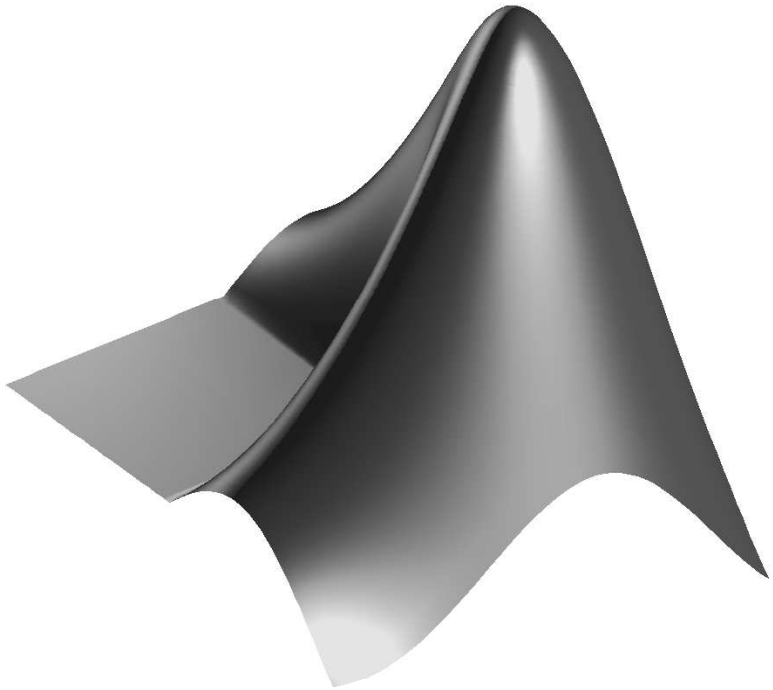
Universidade Federal de Minas Gerais

Departamento de Ciência da Computação do ICEx

Introdução ao MATLAB



Frederico F. Campos, filho

fevereiro de 2000

Sumário

Prefácio

MATLAB (R), acronismo de MATrix LABoratory, é um sistema interativo e linguagem de programação para computação numérica e visualização para as áreas técnicas e científicas. Seu elemento básico de dados é uma matriz, a qual não requer dimensionamento. O MATLAB permite a solução de muitos problemas numéricos em uma fração do tempo que seria necessário para escrever um programa em uma linguagem como FORTRAN, Pascal ou C. Além do mais, em MATLAB as soluções dos problemas são expressas de um modo bem próximo do que elas são escritas matematicamente. MATLAB é marca registrada da
The MathWorks, Inc.
24 Prime Park Way
Natick, Massachusetts 01760-1500 – USA
URL `http://www.mathworks.com`.

Existem versões profissionais do MATLAB para computadores pessoais baseados em microprocessadores 80x86 e Macintosh, estações de trabalho da Digital, Hewlett-Packard, IBM, Silicon Graphics e Sun, além de versões para alguns supercomputadores. É também disponível uma versão de estudante para ambiente Windows e Macintosh.

O objetivo deste texto é apresentar o MATLAB (versão 5.3) como uma linguagem de programação dotada de funções não disponíveis nas linguagens convencionais. Por isto, este material pode ser utilizado em disciplinas, tais como, Programação de Computadores, Cálculo Numérico, Análise Numérica, Álgebra Linear e quaisquer outras dos cursos de Engenharia e das áreas de Ciências Exatas. Também, serão exploradas algumas características próprias que mostrarão por que o MATLAB é uma poderosa ferramenta de apoio ao aprendizado e utilização da Computação Científica.

No capítulo 1 é descrito o ambiente de programação do MATLAB mostrando a janela de comandos e como obter informações de comandos durante a sessão. O capítulo 2 mostra os elementos fundamentais do MATLAB, como constantes e variáveis, expressões e gráficos. No capítulo 3, o MATLAB é apresentado como uma linguagem de programação, sendo descritos comandos de entrada e saída, estruturas condicionais e de repetição e arquivos. O capítulo 4 apresenta funções do MATLAB para resolver problemas de álgebra linear, cálculo de zero e mínimo de função, interpolação, ajuste de curvas e integração numérica.

Sugestões para aprimorar o presente texto, bem como, para efetuar correções serão bem-vindas pelo *e-mail*: `ffcampos@dcc.ufmg.br`.

Belo Horizonte, fevereiro de 2000.

Frederico Ferreira Campos, filho
DCC.ICEx.UFMG

Capítulo 1

Ambiente de programação

A interação entre o MATLAB e o usuário é feita por intermédio de uma janela, na qual um comando é fornecido e o resultado exibido. No caso de dúvida sobre a sintaxe ou mesmo a existência de um comando, o MATLAB possui meios de auxiliar o usuário durante a sessão.

1.1 Janela de comando

Para executar o MATLAB em ambiente Windows ou Macintosh, o cursor deve ser colocado sobre o seu ícone e pressionado o botão da esquerda do *mouse* ou então entrar com o comando `matlab` em caso de ambiente Unix. Aparecerá a janela de comando pela qual será feita a interação entre o MATLAB e o usuário. O sinal de que o programa está pronto para receber um comando é indicado pelos caracteres `>>`, ou por `EDU>>` no caso da versão de estudante. A figura 1.1 mostra a janela de comando da versão de estudante do MATLAB para ambiente Windows.

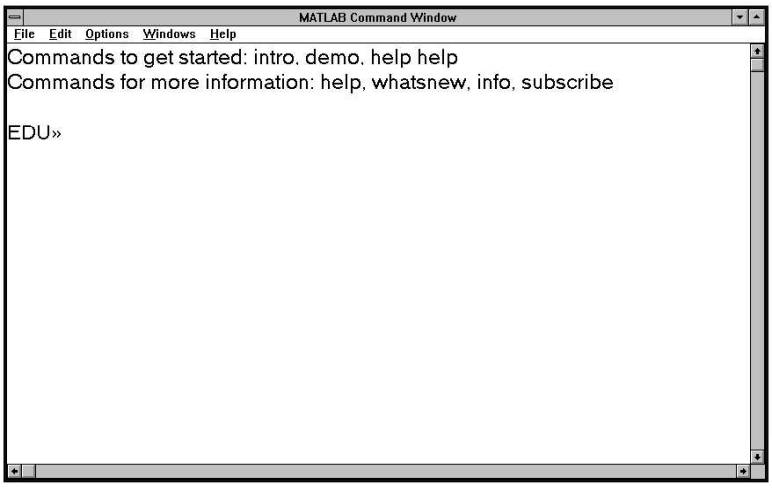


Figura 1.1: Janela de comando da versão de estudante do MATLAB.

Um comando é finalizado acionando-se a tecla **Enter** ou **Return**. Se um comando for muito longo, então três pontos (`...`) seguidos do pressionamento da tecla **Enter** ou **Return** indica

que o comando continuará na próxima linha. Comandos múltiplos podem ser colocados em uma mesma linha se eles forem separados por vírgulas ou ponto-e-vírgulas. Além disto, as vírgulas dizem ao MATLAB para mostrar os resultados e os ponto-e-vírgulas para suprimir a exibição.

Todo texto após o sinal de `%` é considerado comentário, e portanto ignorado, o que torna fácil a documentação de um programa.

As teclas $\uparrow$ e $\downarrow$ podem ser usadas para listar os comandos previamente usados e as teclas $\rightarrow$ e $\leftarrow$ movem o cursor na linha de comandos possibilitando assim a sua modificação.

O número de linhas a serem exibidas de cada vez na janela de comandos pode ser redefinida pelo comando `more`. Sua sintaxe é

```
more(<número-de-linhas>)
```

onde `<número-de-linhas>` define o número de linhas a serem exibidas. O comando é desativado por `more off`, enquanto que `more on` definide um número de linhas de modo a ocupar toda a janela de comandos.

A execução de um comando em MATLAB pode ser interrompida a qualquer instante bastando para isto pressionar as teclas **Control** e **C**, simultaneamente. O comando `clc` é usado para limpar a janela de comandos e `home` posiciona o cursor no canto superior esquerdo da janela de comando. O término de execução do MATLAB é feito pelo comando `quit` ou `exit`.

Quando o MATLAB é ativado, os comandos contidos no arquivo `matlabrc.m` são automaticamente executados para que sejam atribuidos valores a alguns parâmetros. Nesta fase também são executados os comandos do arquivo `startup.m`, caso ele exista. O usuário pode criar, em seu próprio diretório, o arquivo `startup.m` contendo, por exemplo, definição de constantes matemáticas e físicas, formatos de exibição ou quaisquer comandos do MATLAB para personalizar a sua janela de comando.

A abrangência e potencialidade do MATLAB está muito além do que será mostrado neste texto, por isso é aconselhável executar o comando `demo` para visualizar uma demonstração e se ter uma idéia dessa potencialidade.

1.2 Comandos de auxílio ao usuário

O MATLAB possui muito mais comandos do que aqueles apresentados neste texto, o que torna mais difícil lembrá-los. Com o intuito de auxiliar o usuário na procura de comandos, o MATLAB provê assistência por intermédio de suas extensivas capacidades de auxílio direto. Estas capacidades estão disponíveis em três formas: o comando `help`, o comando `lookfor` e interativamente por meio de um menu de barras, no caso de versão para ambiente Windows e Macintosh.

1.2.1 Comando `help`

O comando `help` é a maneira mais simples de obter auxílio no caso de se conhecer o tópico em que se quer assistência. Sua sintaxe é

```
help <tópico>
```

onde `<tópico>` é o nome da função ou de diretório. Por exemplo,

```
>> help sqrt

 SQRT Square root.
  SQRT(X) is the square root of the elements of X. Complex
   results are produced if X is not positive.

  See also SQRTM.
```

Assim são produzidas informações sobre a função `sqrt` para extrair raiz quadrada. O comando `help` funciona a contento quando se conhece exatamente o tópico sobre o qual se quer assistência. Considerando que muitas vezes este não é o caso, o `help` pode ser usado sem `<tópico>` para listar todos os tópicos de auxílio primário

```
>> help

HELP topics:

matlab/general       -  General purpose commands.
matlab/ops           -  Operators and special characters.
matlab/lang          -  Programming language constructs.
matlab/elmat         -  Elementary matrices and matrix manipulation.
matlab/elfun         -  Elementary math functions.
matlab/specfun       -  Specialized math functions.
matlab/matfun        -  Matrix functions - numerical linear algebra.
matlab/datafun       -  Data analysis and Fourier transforms.
matlab/polyfun       -  Interpolation and polynomials.
matlab/funfun        -  Function functions and ODE solvers.
matlab/sparfun       -  Sparse matrices.
matlab/graph2d       -  Two dimensional graphs.
matlab/graph3d       -  Three dimensional graphs.
matlab/specgraph     -  Specialized graphs.
matlab/graphics      -  Handle Graphics.
matlab/uitools       -  Graphical user interface tools.
matlab/strfun        -  Character strings.
matlab/iofun         -  File input/output.
matlab/timefun       -  Time and dates.
matlab/datatypes     -  Data types and structures.
matlab/demos         -  Examples and demonstrations.

For more help on directory/topic, type "help topic".
```

É importante notar que os tópicos listados irão depender da instalação do MATLAB em um dado computador. Estes tópicos podem variar de acordo com esta instalação. Por exemplo, para obter auxílio sobre as funções matemáticas especializadas

```
>> help specfun

  Specialized math functions.

  Specialized math functions.
    airy        - Airy functions.
    besselj     - Bessel function of the first kind.
    bessely     - Bessel function of the second kind.
    besselh     - Bessel functions of the third kind (Hankel function).
    besseli     - Modified Bessel function of the first kind.
    besselk     - Modified Bessel function of the second kind.
    beta        - Beta function.
    betainc     - Incomplete beta function.
    betaln      - Logarithm of beta function.
    ellipj      - Jacobi elliptic functions.
    ellipke     - Complete elliptic integral.
    erf         - Error function.
    erfc        - Complementary error function.
    erfcx       - Scaled complementary error function.
    erfinv      - Inverse error function.
    expint      - Exponential integral function.
    gamma       - Gamma function.
    gammainc    - Incomplete gamma function.
    gammaln     - Logarithm of gamma function.
    legendre    - Associated Legendre function.
    cross       - Vector cross product.

  Number theoretic functions.
    factor      - Prime factors.
    isprime     - True for prime numbers.
    primes      - Generate list of prime numbers.
    gcd         - Greatest common divisor.
    lcm         - Least common multiple.
    rat         - Rational approximation.
    rats        - Rational output.
    perms       - All possible permutations.
    nchoosek    - All combinations of N elements taken K at a time.
    factorial   - Factorial function.

  Coordinate transforms.
    cart2sph    - Transform Cartesian to spherical coordinates.
    cart2pol    - Transform Cartesian to polar coordinates.
    pol2cart    - Transform polar to Cartesian coordinates.
    sph2cart    - Transform spherical to Cartesian coordinates.
    hsv2rgb     - Convert hue-saturation-value colors to red-green-blue.
    rgb2hsv     - Convert red-green-blue colors to hue-saturation-value.
```

O comando `help` é mais conveniente se o usuário conhecer exatamente o tópico em que deseja auxílio. No caso de não saber soletrar ou mesmo desconhecer um tópico as outras duas formas de assistência são muitas vezes mais produtivas.

1.2.2 Comando `lookfor`

O comando `lookfor` provê assistência pela procura em todas as primeiras linhas dos tópicos de auxílio do MATLAB e retornando aquelas que contenham a palavra-chave especificada. O mais importante é que a palavra-chave não precisa ser um comando do MATLAB. Sua sintaxe é

```
lookfor <palavra-chave>
```

onde `<palavra-chave>` é a cadeia de caracteres que será procurada nos comandos do MATLAB. Por exemplo, para informações sobre matriz identidade

```
>> lookfor identity
EYE Identity matrix.
SPEYE  Sparse identity matrix.
```

Apesar da palavra `identity` não ser um comando do MATLAB, ela foi encontrada na descrição de dois comandos, para uma dada instalação. Tendo esta informação, o comando `help` pode ser usado para exibir informações a respeito de um comando específico, como por exemplo

```
>> help eye

 EYE Identity matrix.
    EYE(N) is the N-by-N identity matrix.

    EYE(M,N) or EYE([M,N]) is an M-by-N matrix with 1's on
    the diagonal and zeros elsewhere.

    EYE(SIZE(A)) is the same size as A.

    See also ONES, ZEROS, RAND, RANDN.
```

Em um PC, um auxílio por intermédio de menu-dirigido é disponível selecionando **Table of Contents** ou **Index** no menu **Help**. Como o uso desta forma de assistência é bem intuitivo, o melhor a fazer é experimentar!

1.3 Exercícios

Seção 1.1

1.1. Executar o programa MATLAB.

1.2. Verificar a diferença entre `,` e `;` nos comandos `pi+1, pi*10` e `pi+1; pi*10;`.

1.3. Testar o funcionamento das teclas ↑ e ↓.

1.4. Ver o funcionamento das teclas $\rightarrow$ e $\leftarrow$.

1.5. Verificar a diferença entre `more(20)`, `more off` e `more on`.

Seção 1.2

1.6. Quantos parâmetros tem a função erro `erf`?

1.7. Quais são os comandos utilizados para interpolação?

1.8. Qual o comando usado para calcular o determinante (*determinant*) de uma matriz?

1.9. Qual o comando para achar raiz (*root*) de uma equação polinomial?

1.10. Comparar a soma dos autovalores (*eigenvalues*) com o traço (*trace*) de uma matriz de Pascal de ordem qualquer. Fazer a mesma comparação usando uma matriz com elementos aleatórios.

Capítulo 2

Elementos fundamentais

Neste capítulo serão apresentados alguns ítens básicos, como constantes, variáveis, expressões e gráficos, o que tornará possível o uso imediato do MATLAB no modo interativo. Além disto, estes elementos são fundamentais para a elaboração de programas.

2.1 Constantes e variáveis

Uma constante numérica no MATLAB é formada por uma seqüência de dígitos que pode estar ou não precedida de um sinal positivo (`+`) ou um negativo (`-`) e pode conter um ponto decimal (`.`). Esta seqüência pode terminar ou não por uma das letras `e`, `E`, `d` ou `D` seguida de outra seqüência de dígitos precedida ou não de um sinal positivo (`+`) ou um negativo (`-`). Esta segunda seqüência é a potência de 10 pela qual a primeira seqüência deve ser multiplicada. Por exemplo, `1.23e-1` significa 0,123 e `4.567d2` é 456,7.

Em modo interativo, o MATLAB é um interpretador de expressões. A expressão fornecida é analisada sintaticamente e se estiver correta será avaliada. O resultado pode ser atribuído à uma variável por intermédio do sinal (`=`), ou seja,

```
<variável>=<expressão>
```

Por exemplo

```
>> g=8.6+6.8
g =
   15.4000
```

Se o nome da variável e o sinal de atribuição (`=`) forem omitidos então o resultado será dado à variável *default* `ans`.

```
>> 8/5
ans =
    1.6000
```

Na seção 2.2 serão apresentados os tipos de expressões possíveis no MATLAB. Como qualquer outra linguagem de programação, o MATLAB tem regras a respeito do nome de variáveis, conforme mostrado na tabela 2.1.

regra sobre nome de variável	comentário
conter até 31 caracteres	caracteres além do 31º são ignorados,
começar com letra seguida de letras, números ou (_)	caracteres de pontuação não são permitidos,
Tamanho da letra é diferenciador	`raiz`, `Raiz` e `RAIZ` são variáveis distintas.

Tabela 2.1: Regras para nomes de variáveis.

Algumas linguagens de programação requerem um tratamento especial para números complexos, o que não é o caso do MATLAB. Operações matemáticas com números complexos são escritas do mesmo modo como para números reais.

```
>> a=3+2i
a =
   3.0000 + 2.0000i
>> b=5-7j
b =
   5.0000 - 7.0000i
```

Note que `i` e `j` são usados para indicar a parte imaginária do número complexo. As variáveis reais e complexas em MATLAB ocupam 8 e 16 *bytes* de memória, respectivamente.

2.1.1 Arranjos

Um arranjo ou vetor é um conjunto de variáveis homogêneas (conteúdo de mesmo tipo) identificadas por um mesmo nome e individualizadas por meio de índices. No MATLAB o arranjo pode ser definido elemento por elemento,

`a = [` a_1`, `a_2`, ... , `a_n` ]`

separados por espaço em branco ou vírgula.

```
>> a=[5 1.5,-0.3]
a =
    5.0000    1.5000   -0.3000
```

O vetor `a` possui os elementos `a(1)=5`, `a(2)=1.5` e `a(3)=-0.3`. Se os valores do arranjo forem igualmente espaçados ele pode ser definido por intermédio da expressão

`b = <valor_inicial>:<incremento>:<valor_final>`

Para gerar um vetor com o primeiro elemento igual a 10, o segundo igual a 15, o terceiro igual a 20 e assim sucessivamente até o último igual a 40 basta definir

```
>> b=10:5:40
b =
    10    15    20    25    30    35    40
```

Se o incremento desejado for igual a 1 então ele poderá ser omitido

```
>> u=5:9
u =
     5     6     7     8     9
```

Ao invés de usar incremento, um arranjo pode também ser construído definindo o número desejado de elementos na função `linspace`, cuja sintaxe é

`linspace(<valor_inicial>,<valor_final>,<número_de_elementos>)`

Assim, para criar um arranjo com o primerio elemento igual a 10, o último igual a 40 e possuindo 7 elementos,

```
>> c=linspace(10,40,7)
c =
    10    15    20    25    30    35    40
```

Isto produz um vetor idêntico a `b=10:5:40` definido acima. Se no comando `linspace` o parâmetro `<número_de_elementos>` for omitido então serão gerados 100 pontos. Os elementos também podem ser acessados individualmente,

```
>> c(2)   % segundo elemento de c
ans =
    15
```

ou em blocos usando os comandos de definição de arranjos

```
>> c(3:5)   % terceiro ao quinto elemento
ans =
    20    25    30
```

Lembrando que `5:-2:1` gera a seqüência `5, 3, 1`, então

```
>> c(5:-2:1) % quinto, terceiro e primeiro elementos
ans =
    30    20    10
```

O endereçamento indireto é também possível, permitindo referenciar os elementos em qualquer ordem

```
>> c([4 1])   % quarto e primeiro elementos
ans =
    25    10
```

Nos exemplos acima os arranjos possuem uma linha e várias colunas, por isto são também chamados vetores linha. Do mesmo modo, podem existir vetores coluna, ou seja, arranjos com várias linhas e uma única coluna. Para criar um vetor coluna elemento por elemento estes devem estar separados por (;)

`v = [` v_1`;` v_2`; ... ;` v_n `]`

Deste modo, para gerar um vetor coluna com os elementos `1.5, -3.2, -8.9`,

```
>> v=[1.5;-3.2;-8.9]
v =
    1.5000
   -3.2000
   -8.9000
```

Por isto, separando os elementos de um arranjo por brancos ou vírgulas são especificados os elementos em diferentes colunas (vetor linha). Por outro lado, separando os elementos por ponto-e-vírgula especifica-se os elementos em diferentes linhas (vetor coluna). Para transformar um vetor linha em vetor coluna e vice-versa, o operador de transposição (') é usado

```
>> x=v'
x =
    1.5000   -3.2000   -8.9000
>> y=(1:3)'
y =
     1
     2
     3
```

No caso do vetor ser complexo o operador de transposição (.') deve ser usado para obter a transposição pois o uso do operador (') resultará em um complexo conjugado transposto. Por exemplo, seja o vetor complexo

```
>> z=(1:3)+(0.1:0.1:0.3)*i
z =
   1.0000 + 0.1000i   2.0000 + 0.2000i   3.0000 + 0.3000i
```

o transposto é

```
>> t=z.'
t =
   1.0000 + 0.1000i
   2.0000 + 0.2000i
   3.0000 + 0.3000i
```

e o complexo conjugado transposto

```
>> cc=z'
cc =
   1.0000 - 0.1000i
   2.0000 - 0.2000i
   3.0000 - 0.3000i
```

Assim, no caso do vetor não ser complexo a transposição pode ser feita usando (') ou (.'). A função `length` deve ser usada para saber o comprimento de um vetor

```
>> l=length(z)
l =
     3
```

2.1.2 Matrizes

As matrizes são arranjos bidimensionais e constituem as estruturas fundamentais do MATLAB e por isto existem várias maneiras de manipulá-las. Uma vez definidas, elas podem ser modificadas de várias formas, como por inserção, extração e rearranjo.

Similarmente aos vetores, para construir uma matriz os elementos de uma mesma linha devem estar separados por branco ou vírgula e as linhas separadas por ponto-e-vírgula ou **Enter** (ou **Return**),

`A = [` $a_{11}\ a_{12}\ \ldots\ a_{1n}$ `;` $a_{21}\ a_{22}\ \ldots\ a_{2n}$ `; ...;` $a_{m1}\ a_{m2}\ \ldots\ a_{mn}$ `]`

Para criar uma matriz `A` com 2 linhas e 3 colunas

```
>> A=[3 2 -5; 4 7 9]
A =
     3     2    -5
     4     7     9
```

Para modificar um elemento basta atribuir-lhe um novo valor,

```
>> A(1,2)=8
A =
     3     8    -5
     4     7     9
```

Se for atribuído um valor a um elemento não existente, ou seja, além dos elementos da matriz então o MATLAB aumenta esta matriz automaticamente preenchendo-a com valores nulos de forma a matriz permanecer retangular

```
>> A(3,6)=1
A =
     3     8    -5     0     0     0
     4     7     9     0     0     0
     0     0     0     0     0     1
```

Seja agora a matriz quadrada `B` de ordem 3

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

De modo similar aos arranjos, os elementos de uma matriz podem ser referenciados individualmente, tal como

```
>> B(2,3)
ans =
     6
```

ou em conjuntos, neste caso usando a notação de arranjo. Por exemplo, os elementos das linhas 1 e 3 e coluna 2,

```
>> B([1 3],2)
ans =
     2
     8
```

A notação de vetor, `<valor_inicial>:<incremento>:<valor_final>`, também pode ser usada ou até mesmo `linspace`. Lembrando que `3:-1:1` gera a seqüência `3, 2, 1` e `1:3` produz `1, 2, 3`, então o comando

```
>> C=B(3:-1:1,1:3)
C =
     7     8     9
     4     5     6
     1     2     3
```

cria uma matriz `C` a partir das linhas 3, 2 e 1 e colunas 1, 2 e 3 de `B`, ou seja, cria uma matriz `C` a partir das linhas de `B` em ordem contrária.

Considerando que são referenciadas todas as 3 colunas de `B`, a notação simplificada `(:)` pode ser igualmente usada ao invés de `1:3`

```
>> C=B(3:-1:1,:)
C =
     7     8     9
     4     5     6
     1     2     3
```

Para criar uma matriz `D` a partir das linhas 1 e 2 e colunas 1 e 3 de `B`, faz-se

```
>> D=B(1:2,[1 3])
D =
     1     3
     4     6
```

Para construir uma matriz `E` a partir da matriz `B` seguida da coluna 2 de `C` seguida ainda de 1 coluna com os elementos iguais a 2

```
>> E=[B C(:,2) [2 2 2]']
E =
     1     2     3     8     2
     4     5     6     5     2
     7     8     9     2     2
```

Para remover uma linha ou coluna de uma matriz usa-se a matriz vazia `[]`. Para remover a coluna 3 de `E`

```
>> E(:,3)=[]
E =
     1     2     8     2
     4     5     5     2
     7     8     2     2
```

E posteriormente para remover a linha 1

```
>> E(1,:)=[]
E =
     4     5     5     2
     7     8     2     2
```

A função `size` é usada para fornecer o número de linhas e colunas de uma matriz. Ela pode ser usada de duas formas:

```
>> t=size(E)
t =
     2     4
```

onde a variável `t` é um vetor linha com duas posições, contendo o número de linhas e colunas de `E`, respectivamente. A outra forma é

```
>> [l,c]=size(E)
l =
     2
c =
     4
```

onde as variáveis simples `l` e `c` contém o número de linhas e colunas de `E`, respectivamente. Se a função `length` for usada em uma matriz ela fornecerá o maior valor entre número de linhas e colunas

```
>> c=length(E)
c =
     4
```

O MATLAB tem funções que se aplicam individualmente à cada coluna da matriz produzindo um vetor linha com elementos correspondentes ao resultado de cada coluna. Se a função for aplicada à transposta da matriz ter-se-ão resultados relativos à cada linha da matriz. Se o argumento da função for um vetor ao invés de uma matriz então o resultado será um escalar. Algumas destas funções são mostradas na tabela 2.2.

função	descrição
`sum`	soma dos elementos
`prod`	produto dos elementos
`mean`	média aritmética
`std`	desvio padrão
`max`	maior elemento
`min`	menor elemento
`sort`	ordenação em ordem crescente

Tabela 2.2: Exemplos de funções do MATLAB.

Seja um quadrado mágico que é uma matriz de ordem $n, n \geq 3$ com valores entre 1 e n^2 tal que a soma das linhas, colunas e diagonal são iguais

```
>> A=magic(4)
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
     4    14    15     1
>> sum(A)
ans =
    34    34    34    34
>> prod(A)
ans =
        2880        2156        2700        1248
```

```
>> mean(A)
ans =
    8.5000    8.5000    8.5000    8.5000
>> std(A)
ans =
    5.4467    5.1962    5.1962    5.4467
>> max(A)
ans =
    16    14    15    13
>> max(max(A))   % maior elemento da matriz
ans =
    16
>> min(A)
ans =
     4     2     3     1
>> sort(A)
ans =
     4     2     3     1
     5     7     6     8
     9    11    10    12
    16    14    15    13
```

O MATLAB possui também várias funções para manipulação de matrizes dentre as quais destacam-se aquelas mostradas na tabela 2.3.

função	descrição
`diag`	se o argumento for um vetor então cria uma matriz diagonal com os elementos do vetor; se o argumento for uma matriz então produz um vetor coluna contendo os elementos da diagonal.
`tril`	obtém a parte triangular inferior de uma matriz.
`triu`	obtém a parte triangular superior de uma matriz.

Tabela 2.3: Algumas funções para operações de matrizes.

Para a matriz `A` dada acima

```
>> d=diag(A)
d =
    16
    11
     6
     1
>> D=diag(d)
D =
    16     0     0     0
     0    11     0     0
     0     0     6     0
     0     0     0     1
```

```
>> L=tril(A)
L =
    16     0     0     0
     5    11     0     0
     9     7     6     0
     4    14    15     1
>> U=triu(A)
U =
    16     2     3    13
     0    11    10     8
     0     0     6    12
     0     0     0     1
```

O MATLAB fornece várias matrizes elementares de grande utilidade, como as mostradas na tabela 2.4. Se um único parâmetro for provido então a matriz será quadrada de ordem igual ao valor do parâmetro. Se forem dois parâmetros então ela será retangular com as dimensões iguais aos valores destes parâmetros.

função	descrição da matriz
`zeros`	nula.
`ones`	elementos iguais a `1`.
`eye`	identidade ou parte dela.
`rand`	elementos aleatórios uniformemente distribuidos entre 0 e 1.
`randn`	elementos aleatórios com distribuição normal com média 0 e desvio padrão 1.

Tabela 2.4: Algumas matrizes elementares do MATLAB.

Por exemplo,

```
>> Z=zeros(2)
Z =
     0     0
     0     0
>> U=ones(2,3)
U =
     1     1     1
     1     1     1
>> I=eye(2,4)
I =
     1     0     0     0
     0     1     0     0
>> R=rand(3)
R =
    0.9501    0.4860    0.4565
    0.2311    0.8913    0.0185
    0.6068    0.7621    0.8214
>> N=randn(3,5)
```

```
N =
   -0.4326    0.2877    1.1892    0.1746   -0.5883
   -1.6656   -1.1465   -0.0376   -0.1867    2.1832
    0.1253    1.1909    0.3273    0.7258   -0.1364
```

As funções `rand` e `randn` produzem números pseudo-aleatórios e a seqüência gerada é determinada pelo estado do gerador. Quando o MATLAB é ativado, é atribuído um mesmo valor inicial ao estado, e portanto, a seqüência de números gerados será a mesma, a não ser que o estado seja mudado. Para colocar o gerador no estado inicial quando estas funções já tiverem sido executadas usa-se

```
>> rand('state',0)
>> randn('state',0)
```

2.1.3 Variáveis literais

Uma variável pode conter uma cadeia de caracteres ao invés de um número. Estes caracteres são manipulados iguais aos vetores linha. A cadeia de caracteres deve estar delimitada por apóstrofos para ser atribuida a uma variável literal. Assim,

```
>> a='MATLAB', b='cadeia de caracteres'
a =
MATLAB
b =
cadeia de caracteres
```

Para obter os caracteres das posições 4, 5 e 6 da variável `a`,

```
>> c=a(4:6)
c =
LAB
```

Os caracteres podem ser concatenados

```
>> d=[a ' manipula ' b]
d =
MATLAB manipula cadeia de caracteres
```

As cadeias de caracteres contendo mais de uma linha devem ter o mesmo número de colunas como acontece com as matrizes.

```
>> e=['O MATLAB'
      'permite '
      'matrizes'
      'literais']
e =
O MATLAB
permite
matrizes
literais
```

2.1.4 Variáveis especiais

O MATLAB tem diversas variáveis especiais, as quais estão listadas na tabela 2.5. Com excessão de `ans`, estas variáveis não devem ser redefinidas. Para mais informações sobre matrizes e outras variáveis use `help elmat`.

variável	valor
`ans`	nome de variável *default* usado para resultados.
`pi`	$\pi \approx 3,1416$.
`eps`	menor número tal que quando adicionado a 1 cria um número de ponto flutuante maior que 1. Seu valor é `eps` $= 2^{-52} \approx 2{,}2204 \times 10^{-16}$ em computadores com aritmética de ponto flutuante IEEE.
`inf`	infinito, por exemplo, 1/0.
`NaN`	não é um número (Not-a-Number), por exemplo, 0/0.
`i` e `j`	$i = j = \sqrt{-1}$.
`realmin`	menor número positivo de ponto flutuante em um computador particular. Em máquinas com formato de ponto flutuante IEEE `realmin` $\approx 2{,}2251 \times 10^{-308}$.
`realmax`	maior número de ponto flutuante representável em um computador particular. Em máquinas com formato de ponto flutuante IEEE `realmax` $\approx 1{,}7977 \times 10^{308}$.

Tabela 2.5: Variáveis especiais do MATLAB.

2.2 Expressões

Uma expressão é uma combinação de constantes, variáveis e operadores, cuja avaliação resulta em um valor único. O MATLAB possui três tipos de expressões: aritméticas, lógicas e literais.

2.2.1 Expressões aritméticas

É disponível no MATLAB as operações aritméticas básicas mostradas na tabela 2.6.

operação	expressão	operador	exemplo
adição	$a + b$	`+`	`1+2`
subtração	$a - b$	`-`	`5.1-4.7`
multiplicação	$a \times b$	`*`	`6*9.98`
divisão	$a \div b$	`/` ou `\`	`6/7  5\3`
potenciação	a^b	`^`	`2^10`

Tabela 2.6: Operações aritméticas básicas do MATLAB.

As expressões são avaliadas da esquerda para a direita, tendo a potenciação maior ordem de precedência, seguida pela multiplicação e divisão (ambas tendo igual precedência) e seguida pela adição e subtração (com igual precedência). Os parênteses podem ser usados para alterarem a ordem de precedência.

As operações básicas entre vetores só são definidas quando estes tiverem o mesmo tamanho e orientação (linha ou coluna). Estas operações básicas são apresentadas na tabela 2.7. As operações de multiplicação, divisão e potenciação envolvendo vetores são antecedidas pelo caracter (`.`), o que significa que estas operações são efetuadas elemento por elemento. Considere

Seja a = [a_1 a_2 ... a_n], b = [b_1 b_2 ... b_n] e c um escalar		
operação	expressão	resultado
adição escalar	a+c	[a_1+c a_2+c ... a_n+c]
adição vetorial	a+b	[a_1+b_1 a_2+b_2 ... a_n+b_n]
multiplicação escalar	a*c	[a_1*c a_2*c ... a_n*c]
multiplicação vetorial	a.*b	[a_1*b_1 a_2*b_2 ... a_n*b_n]
divisão à direita	a./b	[a_1/b_1 a_2/b_2 ... a_n/b_n]
divisão à esquerda	a.\b	[b_1/a_1 b_2/a_2 ... b_n/a_n]
potenciação	a.^c	[a_1^c a_2^c ... a_n^c]
	c.^a	[c^a_1 c^a_2 ... c^a_n]
	a.^b	[a_1^b_1 a_2^b_2 ... a_n^b_n]

Tabela 2.7: Operações vetoriais básicas.

```
>> a=1:5, b=10:10:50, c=2
a =
     1     2     3     4     5
b =
    10    20    30    40    50
c =
     2
>> a+c
ans =
     3     4     5     6     7
>> a+b
ans =
    11    22    33    44    55
>> a./b
ans =
    0.1000    0.1000    0.1000    0.1000    0.1000
>> a.^c
ans =
     1     4     9    16    25
>> c.^a
ans =
     2     4     8    16    32
```

De modo similar às operações vetorias, existem as operações matriciais básicas, as quais estão compiladas na tabela 2.8.

O operador \ envolvendo matrizes e vetores está relacionado com solução de sistemas lineares, conforme será visto na seção 4.2.2. Sejam as matrizes A e B de ordem 3 e o escalar c

```
>> A=[1 2 3; 4 5 6; 7 8 9], B=[11 12 13; 14 15 16; 17 18 19], c=3
A =
     1     2     3
```

Seja `c` um escalar e

`A =` [$a_{11}\ a_{12}\ \ldots\ a_{1n}$; $a_{21}\ a_{22}\ \ldots\ a_{2n}$; $\ldots$; $a_{m1}\ a_{m2}\ \ldots\ a_{mn}$]

`B =` [$b_{11}\ b_{12}\ \ldots\ b_{1n}$; $b_{21}\ b_{22}\ \ldots\ b_{2n}$; $\ldots$; $b_{m1}\ b_{m2}\ \ldots\ b_{mn}$]

operação	expressão	resultado
adição escalar	`A+c`	$a_{ij} + c$
adição matricial	`A+B`	$a_{ij} + b_{ij}$
multiplicação escalar	`A*c`	$a_{ij} * c$
multiplicação por elemento	`A.*B`	$a_{ij} * b_{ij}$
multiplicação matricial	`A*B`	AB
divisão por elemento	`A./B`	a_{ij}/b_{ij}
divisão à esquerda	`A.\B`	b_{ij}/a_{ij}
potenciação	`A.^c`	a_{ij}^{c}
	`A^c`	A^{c}
	`c.^A`	$c^{a_{ij}}$
	`c^A`	c^{A}
	`A.^B`	$a_{ij}^{b_{ij}}$

Tabela 2.8: Operações matriciais básicas.

```
     4     5     6
     7     8     9
B =
    11    12    13
    14    15    16
    17    18    19
c =
     3
>> A+c
ans =
     4     5     6
     7     8     9
    10    11    12
>> A+B
ans =
    12    14    16
    18    20    22
    24    26    28
>> A*c
ans =
     3     6     9
    12    15    18
    21    24    27
```

A diferença no resultado das expressões quando os operadores contém o caracter (.) deve ser observada. Na multiplicação,

```
>> A.*B   % multiplicacao de matriz elemento a elemento
ans =
    11    24    39
    56    75    96
   119   144   171
>> A*B  % multiplicacao matricial ordinaria
ans =
    90    96   102
   216   231   246
   342   366   390
```

E na potenciação,

```
>> A.^c   % elemento de uma matriz elevado a uma constante
ans =
     1     8    27
    64   125   216
   343   512   729
>> A^c   % multiplicacao sucessiva de matriz ( A^3 = A*A*A )
ans =
         468         576         684
        1062        1305        1548
        1656        2034        2412
>> c.^A   % constante elevada a elemento da matriz
ans =
           3           9          27
          81         243         729
        2187        6561       19683
```

Como pode ser esperado de uma linguagem para aplicações nas áreas técnicas e científicas, o MATLAB oferece várias funções importantes para matemática, ciência e engenharia. A tabela 2.9 apresenta algumas funções matemáticas elementares do MATLAB, e uma lista mais completa pode ser obtida usando o comando `help elfun`. As funções matemáticas especializadas podem ser listadas usando `help specfun`.

Se a variável for um vetor ou uma matriz a avaliação de uma função se dá para cada elemento da variável

```
>> a=1:5
a =
     1     2     3     4     5
>> b=sqrt(a)
b =
    1.0000    1.4142    1.7321    2.0000    2.2361
```

Os resultados acima podem ser apresentados na forma de uma tabela por intermédio do comando

```
>> [a;b]'
ans =
```

função	descrição	função	descrição
trigonométricas			
acos	arco co-seno	cos	co-seno
acosh	arco co-seno hiperbólico	cosh	co-seno hiperbólico
acot	arco co-tangente	cot	co-tangente
acoth	arco co-tangente hiperbólica	coth	co-tangente hiperbólica
acsc	arco co-secante	csc	co-secante
acsch	arco co-secante hiperbólica	csch	co-secante hiperbólica
asec	arco secante	sec	secante
asech	arco secante hiperbólica	sech	secante hiperbólica
asin	arco seno	sin	seno
asinh	arco seno hiperbólico	sinh	seno hiperbólico
atan	arco tangente	tan	tangente
atanh	arco tangente hiperbólica	tanh	tangente hiperbólica
atan2	arco tangente de 4 quadrantes		
exponenciais			
exp	exponencial	log10	logaritmo decimal
log	logaritmo natural	sqrt	raiz quadrada
complexas			
abs	valor absoluto	imag	parte imaginária do complexo
angle	ângulo de fase	real	parte real do complexo
conj	complexo conjugado		
numéricas			
ceil	arredonda em direção a $+\infty$	lcm	mínimo múltiplo comum
fix	arredonda em direção a 0	rem	resto de divisão
floor	arredonda em direção a $-\infty$	round	arredonda em direção ao inteiro mais próximo
gcd	máximo divisor comum	sign	sinal

Tabela 2.9: Funções matemáticas elementares.

```
1.0000    1.0000
2.0000    1.4142
3.0000    1.7321
4.0000    2.0000
5.0000    2.2361
```

2.2.2 Expressões lógicas

Uma expressão se diz lógica quando os operadores são lógicos e os operandos são relações e/ou variáveis do tipo lógico. Uma relação é uma comparação realizada entre valores do mesmo tipo. A natureza da comparação é indicada por um operador relacional conforme a tabela 2.10.

O resultado de uma relação ou de uma expressão lógica é *verdadeiro* ou *falso*, contudo no MATLAB o resultado é numérico, sendo que `1` significa *verdadeiro* e `0` significa *falso*. Note que (`=`) é usado para atribuição de um valor a uma variável enquanto que (`==`) é usado para comparação de igualdade. No MATLAB os operadores relacionais podem ser usados para comparar vetores do mesmo tamanho ou comparar um escalar com os elementos de um vetor. Em ambos os casos o resultado será um vetor de mesmo tamanho. Sejam os vetores `a` e `b` e o escalar `c`

operador relacional	descrição
`>`	maior que
`>=`	maior ou igual a
`<`	menor que
`<=`	menor ou igual a
`==`	igual a
`~=`	diferente de

Tabela 2.10: Operadores relacionais do MATLAB.

```
>> a=1:10, b=9:-1:0, c=5
a =
     1     2     3     4     5     6     7     8     9    10
b =
     9     8     7     6     5     4     3     2     1     0
c =
     5
```

Assim

```
>> d=a>=c
d =
     0     0     0     0     1     1     1     1     1     1
```

produz um vetor que contém o valor `1` (*verdadeiro*) quando o elemento correspondente do vetor `a` for maior ou igual a `5`, caso contrário contém o valor `0` (*falso*). Como no MATLAB o resultado da relação lógica é um valor numérico, este pode fazer parte de uma expressão aritmética

```
>> e=a+(b<=3)
e =
     1     2     3     4     5     6     8     9    10    11
```

Portanto, quando um elemento de `b` for menor ou igual a `3`, o valor `1` (resultado da relação lógica) é adicionado ao correspondente valor de `a`.

Os operadores lógicos permitem a combinação ou negação das relações lógicas. Os operadores lógicos do MATLAB são listados na tabela 2.11.

operador lógico	descrição	uso
`&`	**e**	conjunção
`|`	**ou**	disjunção
`~`	**não**	negação

Tabela 2.11: Operadores lógicos do MATLAB.

Para os vetores `a` e `b` definidos anteriormente

```
>> f=(a>3)&(a<=8)
f =
     0     0     0     1     1     1     1     1     0     0
```

Os elementos de `f` são iguais a `1` quando os correspondentes elementos de `a` forem maiores do que `3` e menores ou iguais a `8`. Para fazer a negação, ou seja, onde for `1` será `0` e vice-versa basta

```
>> g=~f
g =
     1     1     1     0     0     0     0     0     1     1
```

A ordem de precedência no MATLAB para expressões envolvendo operadores aritméticos e lógicos são indicados na tabela 2.12.

ordem de precedência	operadores
1^a	`^ .^ ' .'`
2^a	`* / \ .* ./ .\`
3^a	`+ - ~` +(unário) -(unário)
4^a	`: > >= < <= == ~=`
5^a	`| &`

Tabela 2.12: Ordem de precedência das operações aritméticas e lógicas.

Os parênteses podem ser usados para alterarem a ordem de precedência. Assim, a avaliação da expressão

```
>> h=-(a-5).^2>-9&3*b<=15
h =
     0     0     0     0     1     1     1     0     0     0
```

é equivalente às várias etapas

```
>> h1=a-5
h1 =
    -4    -3    -2    -1     0     1     2     3     4     5
>> h2=h1.^2
h2 =
    16     9     4     1     0     1     4     9    16    25
>> h3=3*b
h3 =
    27    24    21    18    15    12     9     6     3     0
>> h4=-h2
h4 =
   -16    -9    -4    -1     0    -1    -4    -9   -16   -25
>> h5=h4>-9
h5 =
     0     0     1     1     1     1     1     0     0     0
>> h6=h3<=15
h6 =
     0     0     0     0     1     1     1     1     1     1
>> h=h5&h6
h =
     0     0     0     0     1     1     1     0     0     0
```

2.2.3 Expressões literais

O MATLAB dispõe de várias funções para manipulação de cadeia de caracteres conforme pode ser obtido por `help strfun`. Algumas delas são mostradas a seguir. O comando `n = abs(s)` retorna no vetor `n` os valores numéricos dos caracteres ASCII (*American Standard Code for Information Interchange*) contidos no vetor `s` de acordo com a tabela 2.13.

representação decimal dos caracteres															
33	!	45	-	57	9	69	E	81	Q	93	]	105	i	117	u
34	"	46	.	58	:	70	F	82	R	94	^	106	j	118	v
35	#	47	/	59	;	71	G	83	S	95	_	107	k	119	w
36	$	48	0	60	<	72	H	84	T	96	‘	108	l	120	x
37	%	49	1	61	=	73	I	85	U	97	a	109	m	121	y
38	&	50	2	62	>	74	J	86	V	98	b	110	n	122	z
39	’	51	3	63	?	75	K	87	W	99	c	111	o	123	{
40	(	52	4	64	@	76	L	88	X	100	d	112	p	124	|
41	)	53	5	65	A	77	M	89	Y	101	e	113	q	125	}
42	*	54	6	66	B	78	N	90	Z	102	f	114	r	126	~
43	+	55	7	67	C	79	O	91	[	103	g	115	s		
44	,	56	8	68	D	80	P	92	\	104	h	116	t		

Tabela 2.13: Caracteres em código ASCII.

Por exemplo, para o vetor `s` de tamanho 4,

```
>> s = '5+3i'
s =
5+3i
>> n = abs(s)
n =
    53    43    51   105
```

De modo contrário, `c = setstr(m)` constrói a cadeia de caracteres `c` a partir do vetor numérico `m`. Este comando é particularmente útil para escrever textos em português (ver tabela 2.14).

```
>> m = [ 70 117 110 231 227 111 ]
m =
    70   117   110   231   227   111
>> c = setstr(m)
c =
Função
```

ou de um modo mais simples

```
>> d = ['N',setstr(250),'mero']
d =
```

representação decimal dos caracteres															
192	À	200	È	208	Ð	216	Ø	224	à	232	è	240	ð	248	ø
193	Á	201	É	209	Ñ	217	Ù	225	á	233	é	241	ñ	249	ù
194	Â	202	Ê	210	Ò	218	Ú	226	â	234	ê	242	ò	250	ú
195	Ã	203	Ë	211	Ó	219	Û	227	ã	235	ë	243	ó	251	û
196	Ä	204	Ì	212	Ô	220	Ü	228	ä	236	ì	244	ô	252	ü
197	Å	205	Í	213	Õ	221	Ý	229	å	237	í	245	õ	253	ý
198	Æ	206	Î	214	Ö	222		230	æ	238	î	246	ö	254	
199	Ç	207	Ï	215	×	223	ß	231	ç	239	ï	247	÷	255	ÿ

Tabela 2.14: Caracteres em código ASCII extendido do MATLAB.

`Número`

A função `str2num('<cadeia_de_caracteres>')` converte a representação do valor numérico da `<cadeia_de_caracteres>` escrita em código ASCII para a representação numérica. A `<cadeia_de_caracteres>` deve estar escrita de acordo com as regras de definição de constantes numéricas vistas na seção 2.1

```
>> x = str2num('1.23d2+5.678e-1i')
x =
  1.2300e+02+  5.6780e-01i
```

com este valor numérico é possível, por exemplo, extrair a raiz quadrada

```
>> y = sqrt(x)
y =
  11.0906 + 0.0256i
```

Por outro lado, a função `t = num2str(a)` converte o número `a` para a sua representação em caracteres. Esta função é de grande utilidade quando da escrita de rótulos e títulos em gráficos (ver seção 2.3).

```
>> v = 4.1542784;
>> ['velocidade = ',num2str(v),' m/s']
ans =
velocidade = 4.1543 m/s
```

O resultado de `num2str` é exibido normalmente com 4 dígitos. Se um parâmetro extra for dado então ele indicará o número máximo de dígitos a serem exibidos

```
>> num2str(v,6)
ans =
4.15428
```

2.3 Gráficos

Uma das grandes virtudes do MATLAB é a facilidade que ele oferece para produzir gráficos de boa qualidade. Nesta seção serão vistos como gerar gráficos bi e tridimensionais e os modos de gravá-los em arquivos para que possam ser incluídos em textos.

2.3.1 Gráficos bidimensionais

Para gerar gráficos bidimensionais podem ser usados as versáteis funções `plot` e `fplot`.

Função `plot`

A sintaxe desta função é

```
plot(x1,y1,'<tipo_de_linha_1>',x2,y2,'<tipo_de_linha_2>',...)
```

onde `x` e `y` são vetores contendo as abscissas e ordenadas dos pontos a serem exibidos, respectivamente e `<tipo_de_linha>` é uma cadeia de 1 a 4 caracteres que especifica a cor e o estilo da linha, os quais são mostrados na tabela 2.15.

símbolo	cor	símbolo	estilo de linha
`y`	amarela	`.`	ponto
`m`	lilás	`o`	círculo
`c`	turquesa	`x`	marca x
`r`	vermelho	`+`	mais
`g`	verde	`*`	asterisco
`b`	azul	`-`	linha sólida
`w`	branco	`:`	linha pontilhada
`k`	preto	`-.`	linha de traço e ponto
		`--`	linha tracejada

Tabela 2.15: Alguns tipos de linha da função `plot`.

Considere os vetores

```
>> x = linspace(-8,8,50);
>> y = sin(x);
>> z = cos(x);
```

Para gerar um gráfico de $\text{sen}(x)$ em função de x basta

```
>> plot(x,y)
```

O gráfico produzido é mostrado na figura 2.1(a).

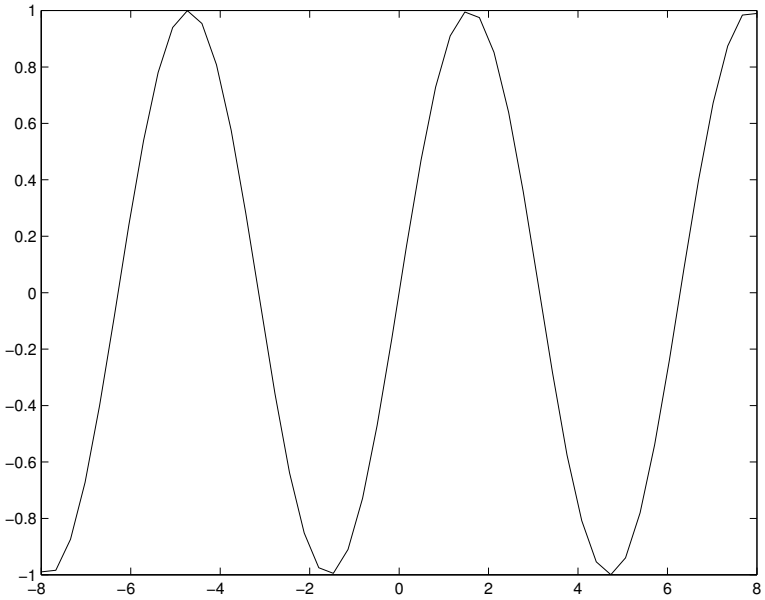
Também pode ser gerado um gráfico um pouco mais complexo, $\text{sen}(x)$ e $\cos(x)$ em função de x com os valores de $\text{sen}(x)$ sendo destacados com (`*`) e os de $\cos(x)$ com (`o`). Para isto o comando

```
>> plot(x,y,'-*',x,z,'-.o')
```

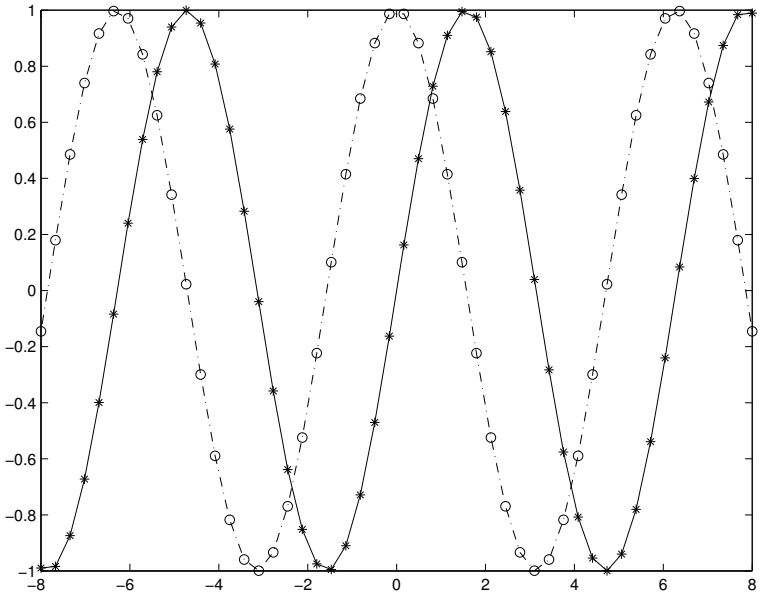
produz a figura 2.1(b). O MATLAB oferece alguns comandos para identificação nos gráficos, como os exibidos na tabela 2.16.

Assim os comandos

```
>> xlabel('eixo x')
```



(a) $f(x) = \text{sen}(x)$



(b) $f(x) = \text{sen}(x)$ e $f(x) = \cos(x)$

Figura 2.1: Gráficos produzidos pelo comando `plot`.

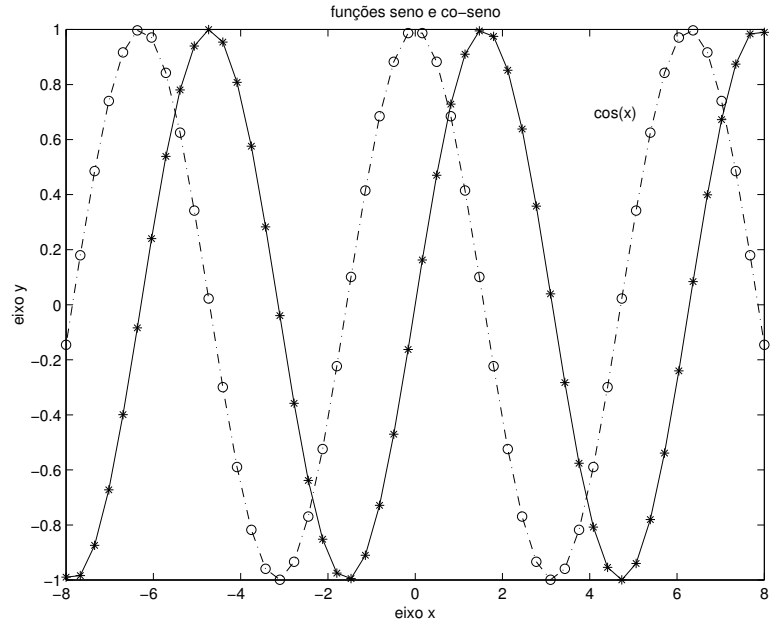
comando	descrição
`xlabel('<texto>')`	escreve `<texto>` abaixo do eixo das abscissas;
`ylabel('<texto>')`	escreve `<texto>` ao lado do eixo das ordenadas;
`title('<texto>')`	escreve `<texto>` no alto da figura;
`text(`x_i`,`y_i`,'<texto>')`	escreve `<texto>` na posição (`$x_i$`,`$y_i$`);
`gtext('<texto>')`	escreve `<texto>` na posição indicada pelo *mouse*.

Tabela 2.16: Funções para identificação de gráficos.

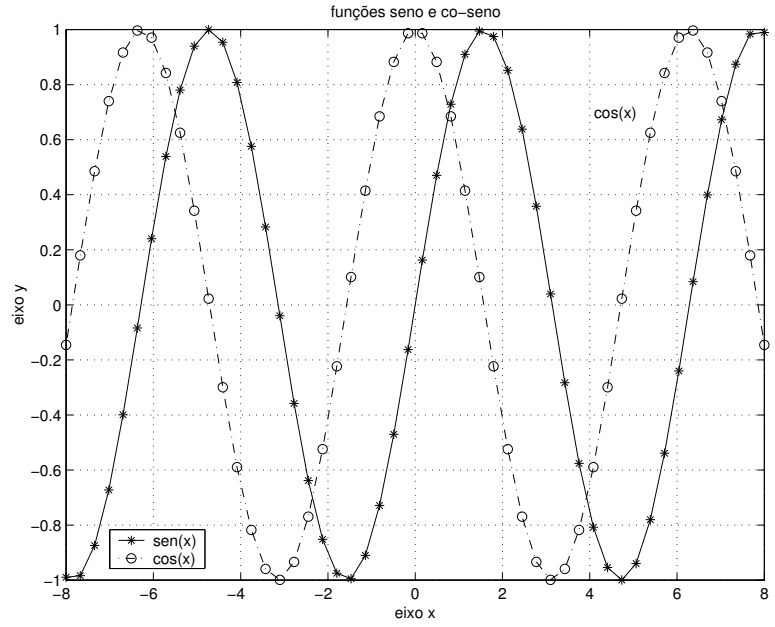
```
>> ylabel('eixo y')
>> title(['fun',setstr(231),setstr(245),'es seno e co-seno'])
```

```
>> text(4.1,0.7,'cos(x)')
```

produzirão os rótulos e títulos no gráfico da figura 2.1(b), como mostrado na figura 2.2(a). Note que a função `setstr` foi usada para gerar os caracteres da língua portuguesa mostrados na tabela 2.14 da página 25.



(a) Rótulos e títulos



(b) Grade e legendas

Figura 2.2: Documentação de gráficos.

O comando `grid on` faz com que apareça uma grade no gráfico produzido. Por sua vez `grid off` faz com que a grade desapareça. O uso de `grid` sem argumento altera o estado, isto é, se não houver grade então `grid` será equivalente à `grid on` e se houver `grid` fará com que a grade desapareça. O comando `legend('<texto_1>','<texto_2>',...)` adiciona uma

caixa de legendas ao esboço da figura atual, usando as cadeias de caracteres `<texto>` como rótulos. Para mover a caixa de legendas basta pressionar o botão esquerdo do *mouse* sobre ela e colocá-la na posição desejada. A figura 2.2(b) mostra o efeito dos comandos

```
>> grid on
>> legend('sen(x)','cos(x)')
```

Para mais informações sobre o uso de `plot` e das funções relacionadas use o comando `help`.

Função `fplot`

O comando `fplot('<função>',[Xmin Xmax])` faz com que a função especificada pela cadeia de caracteres `'<função>'` seja esboçada no intervalo `Xmin` $\leq x \leq$ `Xmax`. Os comandos

```
>> fplot('sin(x)*x',[-20 20]); grid on
>> xlabel('x'); ylabel('f(x)'); title('f(x)=sen(x)*x')
```

produzem o gráfico mostrado na figura 2.3(a). Para que a função especificada pela cadeia de caracteres `'<função>'` seja esboçada somente para `Xmin` $\leq x \leq$ `Xmax` e `Ymin` $\leq f(x) \leq$ `Ymax` deve ser usado o comando `fplot('<função>',[Xmin Xmax Ymin Ymax])`. Deste modo, os comandos abaixo geram o gráfico da figura 2.3(b).

```
>> fplot('sin(x)*x',[-18 18 -10 10]); grid on
>> xlabel('x'); ylabel('f(x)'); title('f(x)=sen(x)*x')
```

Para mais informações sobre os comandos para manipulação de gráficos bidimensionais use `help graph2d`.

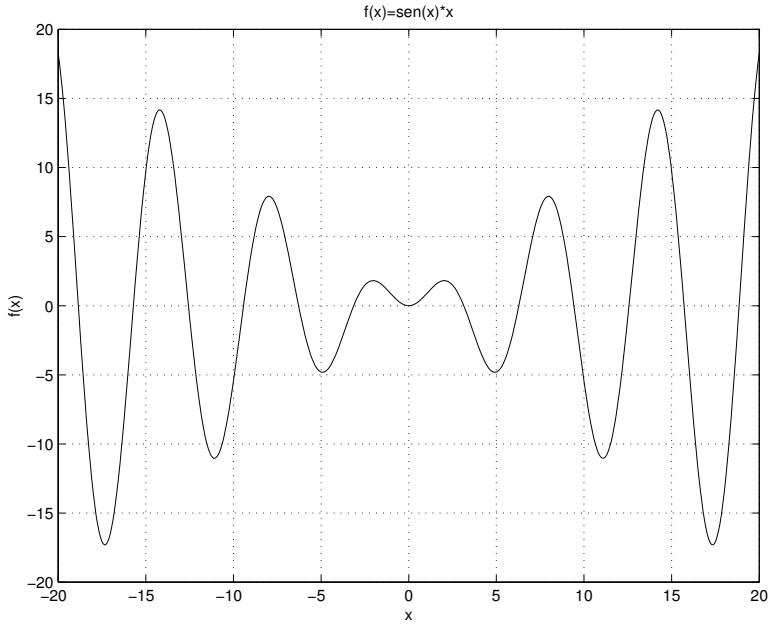
2.3.2 Gráficos tridimensionais

De uma maneira similar aos gráficos bidimensionais, o MATLAB possui vários comandos para que gráficos tridimensionais sejam também facilmente esboçados. Serão descritas a seguir, de modo sucinto, algumas funções com esta finalidade. Portanto, para mais informações sobre essas funções deve ser utilizado o comando `help graph3d`.

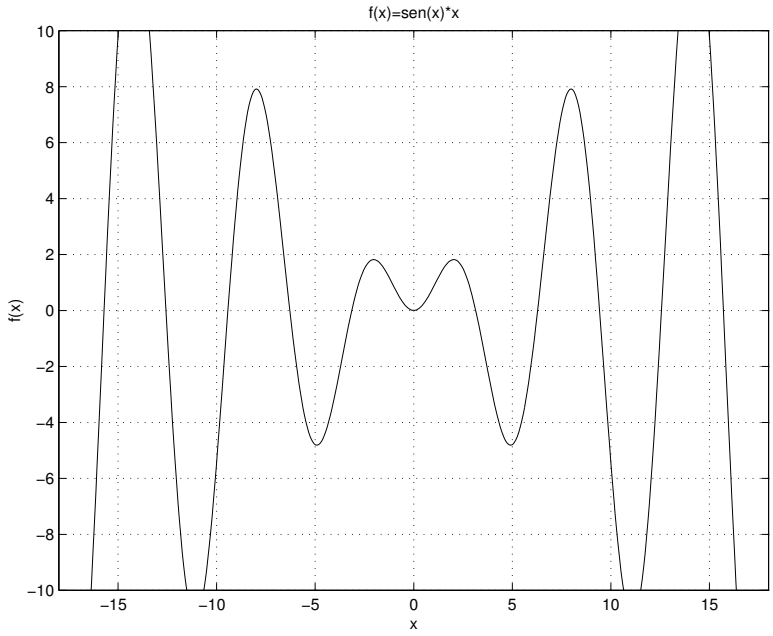
Função `meshgrid`

O comando `[X,Y]=meshgrid(x,y)` transforma o domínio especificado pelos vetores `x` e `y` em matrizes `X` e `Y` que podem ser usadas para avaliar funções de duas variáveis e fazer esboços tridimensionais de malhas e superfícies. As linhas da matriz `X` são cópias do vetor `x` e as colunas da matriz `Y` são cópias do vetor `y`

```
>> x = -1:0.5:2
x =
   -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000
>> y = -1:0.5:1
y =
   -1.0000   -0.5000         0    0.5000    1.0000
>> [X,Y] = meshgrid(x,y)
X =
   -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000
```



(a) Restrição em x



(b) Restrições em x e y

Figura 2.3: Esboços da função $f(x) = \text{sen}(x)x$ pelo comando `fplot`.

```
    -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000
    -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000
    -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000
    -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000
Y =
    -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000   -1.0000
    -0.5000   -0.5000   -0.5000   -0.5000   -0.5000   -0.5000   -0.5000
          0         0         0         0         0         0         0
     0.5000    0.5000    0.5000    0.5000    0.5000    0.5000    0.5000
```

```
    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000    1.0000
>> Z = X+Y
Z =
   -2.0000   -1.5000   -1.0000   -0.5000         0    0.5000    1.0000
   -1.5000   -1.0000   -0.5000         0    0.5000    1.0000    1.5000
   -1.0000   -0.5000         0    0.5000    1.0000    1.5000    2.0000
   -0.5000         0    0.5000    1.0000    1.5000    2.0000    2.5000
         0    0.5000    1.0000    1.5000    2.0000    2.5000    3.0000
```

Função `plot3`

Esta função é uma generalização da função `plot` vista na seção 2.3.1 para esboços tridimensionais. Sua sintaxe é

```
plot3(X1,X1,Z1,'<tipo_de_linha_1>',X2,Y2,Z2'<tipo_de_linha_2>',...)
```

onde `X`, `Y` e `Z` são matrizes de mesmo tamanho contendo as coordenadas tridimensionais dos pontos a serem exibidos e `<tipo_de_linha>` é uma cadeia de 1 a 4 caracteres que especifica a cor e o estilo da linha de acordo com a tabela 2.15 da página 26. Se `X`, `Y` e `Z` forem vetores de mesmo tamanho então é esboçada uma linha no espaço tridimensional cujas coordenadas são os elementos destes vetores. Assim, para produzir o esboço da função $z = \text{sen}(x)\cos(y)^2$ no intervalo $-3 \leq x \leq 3$ e $-2 \leq y \leq 2$, mostrado na figura 2.4(a), utiliza-se os comandos:

```
>> [X,Y] = meshgrid(-3:0.2:3,-2:0.2:2);
>> Z = sin(X).*cos(Y).^2;
>> plot3(X,Y,Z); grid on
>> xlabel('eixo x'); ylabel('eixo y'); zlabel('eixo z')
>> title('plot3')
```

Funções `mesh`, `meshc` **e** `meshz`

A função `mesh(X,Y,Z)` produz o esboço de uma malha na superfície especificada pelas matrizes `X`, `Y` e `Z`. A função `meshc(X,Y,Z)` esboça a malha com contornos desenhados abaixo e `meshz(X,Y,Z)` faz o esboço da malha com um plano de referência. Os comandos

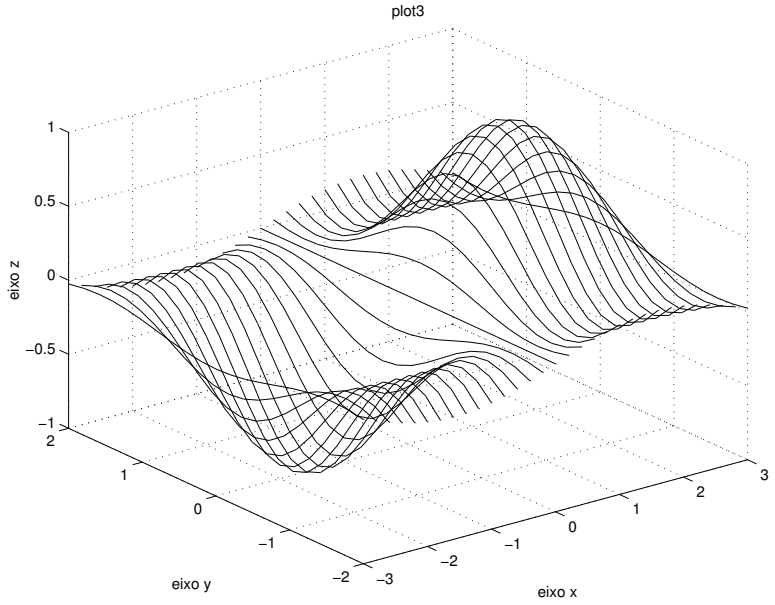
```
>> [X,Y] = meshgrid(-3:0.2:3,-2:0.2:2);
>> Z = sin(X).*cos(Y).^2;
>> mesh(X,Y,Z); grid on
>> xlabel('eixo x'); ylabel('eixo y'); zlabel('eixo z')
>> title('mesh')
```

geram o esboço mostrado na figura 2.4(b). Por outro lado,

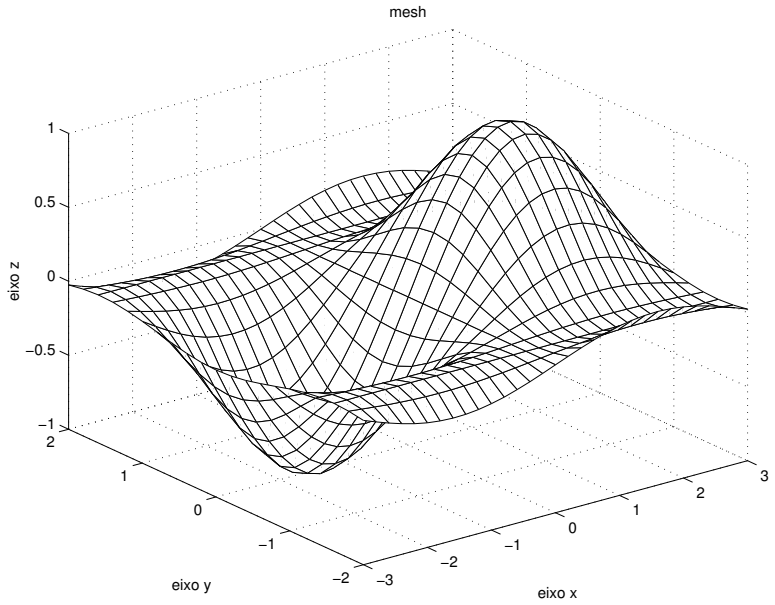
```
>> meshc(X,Y,Z); grid on
>> xlabel('eixo x'); ylabel('eixo y'); zlabel('eixo z')
>> title('meshc')
```

usando a mesma matriz `Z` definida acima e

```
>> meshz(X,Y,Z); grid on
>> xlabel('eixo x'); ylabel('eixo y'); zlabel('eixo z')
>> title('meshz')
```



(a) `plot3`



(b) `mesh`

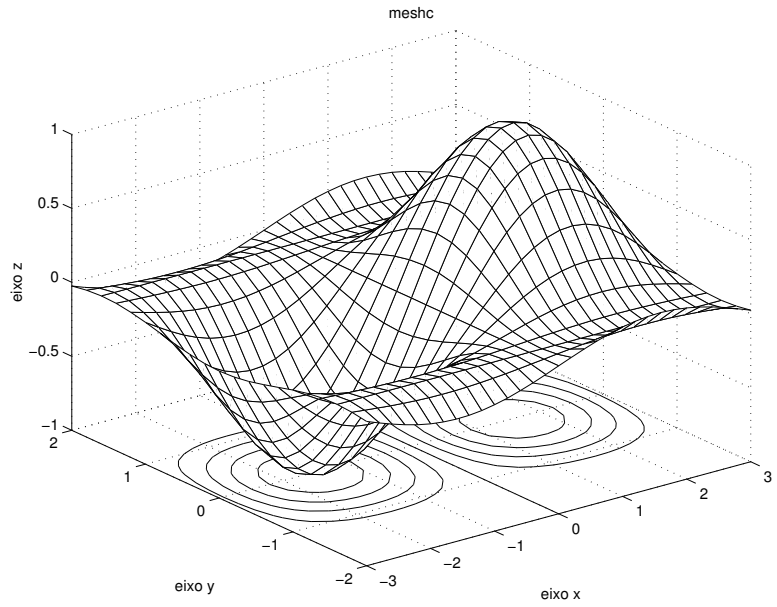
Figura 2.4: Esboços da função $z = \text{sen}(x)\cos(y)^2$ usando os comandos `plot3` e `mesh`.

produzem os esboços apresentados na figuras 2.5(a) e (b).

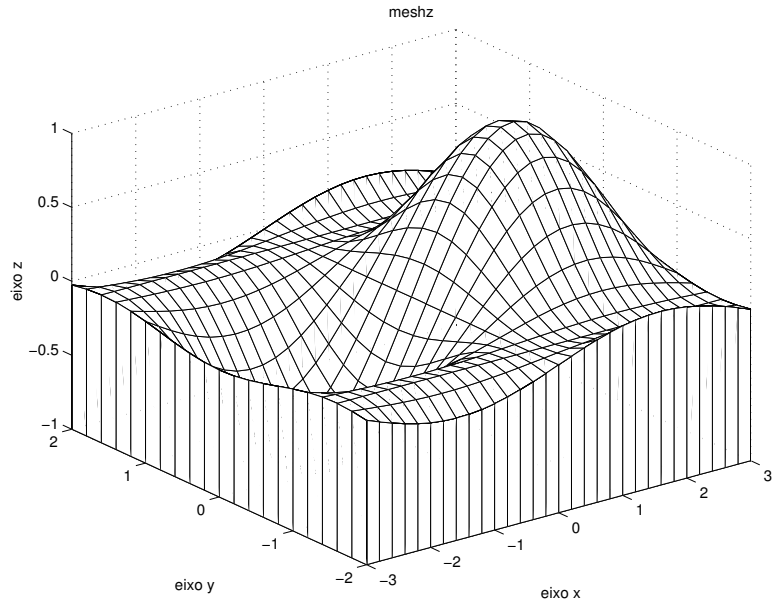
Funções `surf` e `surfl`

A função `surf(X,Y,Z)` produz uma superfície sombreada tridimensional especificada pelas matrizes `X`, `Y` e `Z`. A função `surfl(X,Y,Z)` gera uma superfície sombreada acrescida de um efeito de iluminação. Deste modo, os comandos

```
>> [X,Y] = meshgrid(-3:0.2:3,-2:0.2:2);
>> Z = sin(X).*cos(Y).^2;
```



(a) `meshc`



(b) `meshz`

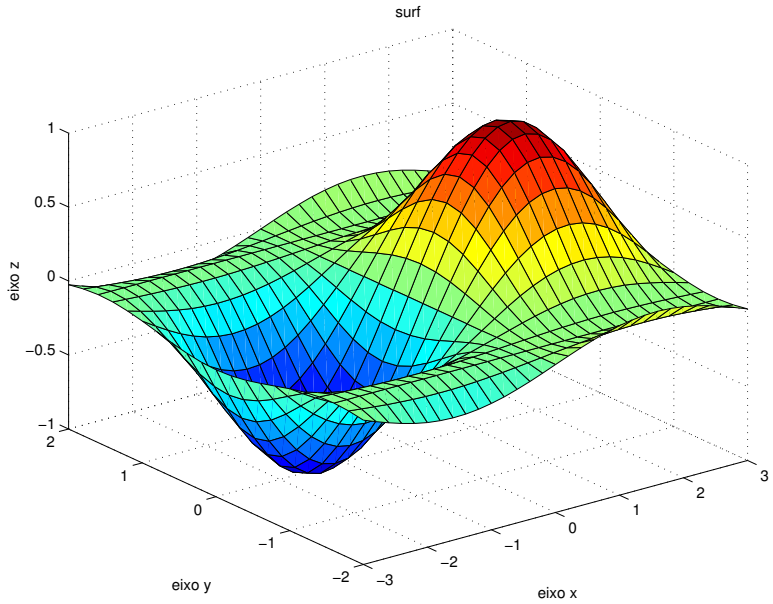
Figura 2.5: Esboços de $z = \operatorname{sen}(x)\cos(y)^2$ usando os comandos `meshc` e `meshz`.

```
>> surf(X,Y,Z); grid on
>> xlabel('eixo x'); ylabel('eixo y'); zlabel('eixo z')
>> title('surf')
```

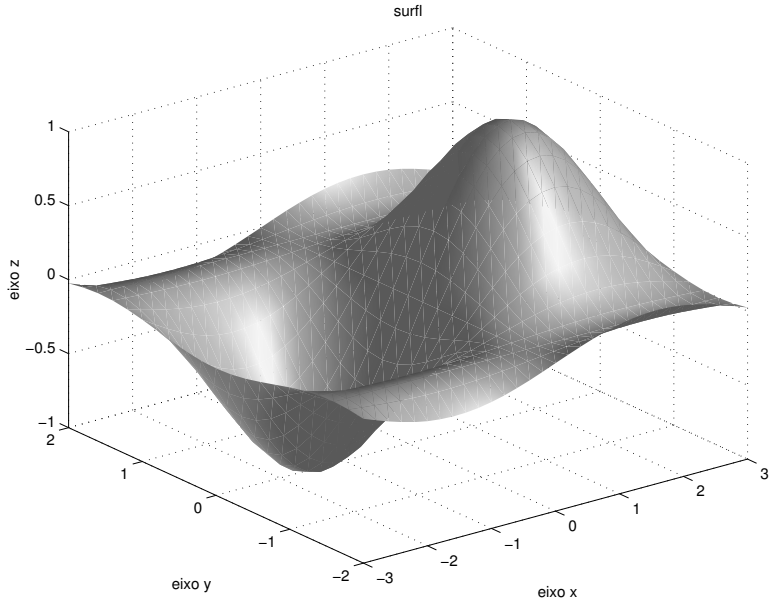
com a matriz `Z` definida acima e

```
>> surfl(X,Y,Z); grid on
>> xlabel('eixo x'); ylabel('eixo y'); zlabel('eixo z')
>> title('surfl')
>> shading interp
>> colormap(gray)
```

produzem as superfícies mostradas nas figuras 2.6(a) e (b). Os comandos `shading interp` e `colormap(gray)` foram usados para realçar o efeito da iluminação.



(a) `surf`



(b) `surfl`

Figura 2.6: Esboços de $z = \text{sen}(x)\cos(y)^2$ usando os comandos `surf` e `surfl`.

2.3.3 Figuras

As figuras geradas pelo MATLAB podem ser facilmente impressas ou gravadas em um arquivo para posterior impressão ou inclusão em um texto. Para tal basta usar o comando `print` cuja sintaxe é

```
print [-d<disp>] <nome_do_arquivo>
```

onde <disp> são dispositivos para especificarem o tipo de arquivo a ser gravado com o nome <nome_do_arquivo>. Alguns destes dispositivos são mostrados na tabela 2.17.

<disp>	descrição	<disp>	descrição
ps	PostScript para impressoras em preto e branco	eps	PostScript encapsulado para impressoras em preto e branco
psc	PostScript para impressoras em cores	epsc	PostScript encapsulado para impressoras em cores
ps2	PostScript nível 2 para impressoras em preto e branco	esp2	PostScript nível 2 encapsulado para impressoras em preto e branco
psc2	PostScript nível 2 para impressoras em cores	epsc2	PostScript nível 2 encapsulado para impressoras em cores

Tabela 2.17: Alguns dispositivos para gravação de arquivos de figuras.

Assim, o comando

```
>> print -deps figplot
```

gera um arquivo PostScript (R) encapsulado com nome `figplot.eps`. O comando `help print` deve ser usado para obter mais informações sobre todos os dispositvos suportados pelo MATLAB.

2.4 Exercícios

Seção 2.1

Observar atentamente e anotar o resultado dos comandos do MATLAB. Apesar dos comandos estarem separados por vírgula, entrar com um de cada vez.

2.1. Atribuir o valor -1,23$\times 10^3$ à variável `a` e 4,17$\times 10^{-2}$ à variável `b`.

2.2. Conferir os nomes permitidos de variáveis

```
arco-seno=0, limite:sup=3, Area=3, area=1
```

2.3. Construir os vetores

```
a=[1 3.5 -4.2 7.5], b=10:5:30 , c=5:-2:-6, d=5:10,
e=linspace(0,10,5), e(3), e(2:4), e(3:-1:1), e([5 1 3]), f=e', g=e(:),
h=(2:3:11)'
```

2.4. Construir as matrizes e verificar o resultado das funções

```
A=[6 -1 4; 0 2 -3; 5 7 8], A(2,1)=1, B=A(2:3,[2 3 1]), C=[A B'],
C(:,[2 4])=[], C(2,:)=[], size(C), sum(C), sum(C'), prod(C), prod(C'),
min(C'), max(C), zeros(3,5), ones(4), eye(5), D=rand(3), diag(diag(D)),
triu(D), tril(D)
```

2.5. Analisar o resultado dos comandos

```
s='MATLAB', [s ' em ordem reversa : ' s(length(s):-1:1)]
```

Seção 2.2

Observar atentamente e anotar o resultado dos comandos do MATLAB. Apesar dos comandos estarem separados por vírgula, entrar com um de cada vez.

2.6. Avaliar as expressões escalares

```
5+3, 4-1, 6*3, 10/2, 2\5, 3^2, 1+2*3-4/5, 6*2^3
```

2.7. Seja $a = 1, b = 2, c = 3, d = 4, e = 5$. Avaliar

$$x = \sqrt[3]{e(e-a)(e-b)+c^d}$$

$$y = a - \frac{b^2-c}{d+e} + \left(\frac{\cos(a)}{d+\operatorname{sen}(b+c)}\right)^3$$

$$z = \log_{10}\left(\frac{b}{d+e^2}\right) + \exp\left(\frac{b+a}{b-a} + \frac{c^2}{1+b^3}\sqrt{1+b^2}\right)$$

2.8. Avaliar as expressões vetoriais

```
u=2:6, v=linspace(10,14,5), k=5,
u-k, u+v, u*k, u.*v, u./v, u.\v, u.^k, k.^u, v.^u
```

2.9. Avaliar as expressões matriciais

```
M=[2 -3 1; 4 6 -1; -5 2 1], N=[1 1 2; 3 1 -1; 3 2 1], x=(1:3)', z=2
M-z, M+N, M*z, M.*N, M*N, M*x, M./N, M.\N, M.^z, M^z, z.^M, z^M, M.^N
```

2.10. Verificar a diferença entre as funções usando $a = \pi$ e $a = -5.6$:

```
ceil(a), fix(a), floor(a), round(a), sign(a)
```

2.11. Para $c = 3 + 4i$ e $c = 5 - 3i$, observar os resultados das funções complexas

```
abs(c), conj(c), real(c), imag(c)
```

2.12. Com relação às variáveis lógicas, dados

```
a=1   % verdadeiro
b=0   % falso
c=sqrt(2)>1
d=exp(0)==0
```

completar a tabela

a	b	c	d	a&b	a&c	b&d	a|b	a|c	b|d	~a	~b

2.13. Avaliar as expressões lógicas para `x=-5:5`

`x>-2 & x<=3, x>3 | abs(x)>2`

2.14. Definir os comandos para gerarem as cadeias de caracteres
`Título`, `variável` e `equações`.

2.15. Qual a diferença entre as funções `str2num` e `num2str`?

Seção 2.3

2.16. Seja a função

$$y = e^{1+x/10} + \cos(x)x.$$

Gerar uma tabela com 20 pontos para $-5 \leq x \leq 5$ e exibir o gráfico da função usando o comando `plot` colocando rótulos e grade.

2.17. Repetir a operação acima usando 50 pontos e sem usar grade.

2.18. Gerar uma tabela de 40 pontos para

$$y = \text{sen}(x)x, \ -10 \leq x \leq 10$$

e exibir o seu gráfico usando o comando `fplot` com rótulos e grade.

2.19. Seja a função

$$z = \text{sen}(x)y^2.$$

Gerar uma superfície com $-5 \leq x \leq 5$ e $-3 \leq y \leq 3$ usando os comandos `meshc` e `meshz`.

2.20. Usando os comandos `surf` e `surfl` exibir a superfície de

$$z = x\cos(x)\,\text{sen}(y)$$

com $-\pi \leq x \leq \pi$ e $-\pi \leq y \leq \pi$.

Capítulo 3

MATLAB como linguagem de programação

No capítulo anterior foram descritos os elementos fundamentais do MATLAB, os quais possibilitam sua utilização em um modo interativo. Neste capítulo serão abordados alguns comandos que tornarão possível desenvolver programas escritos em MATLAB, tais como, comandos para entrada e saída de dados, manipulação de arquivos, estruturas condicionais e estruturas de repetição.

3.1 Comandos de entrada e saída

O MATLAB fornece algumas facilidades para especificar o formato de saída dos resultados, gravação e leitura das variáveis de uma sessão de trabalho para uso posterior e edição de programas em arquivos.

3.1.1 Formato de exibição

Para saber o valor de uma variável basta entrar com o seu nome.

```
>> pi
ans =
    3.1416
```

O comando `disp(<variável>)` é usado para exibir `<variável>` sem mostrar o seu nome ou para exibir uma cadeia de caracteres contida na `<variável>`.

```
>> disp('numeros aleatorios entre 0 e 1'), disp(rand(2,6))
numeros aleatorios entre 0 e 1
    0.9501    0.6068    0.8913    0.4565    0.8214    0.6154
    0.2311    0.4860    0.7621    0.0185    0.4447    0.7919
```

Quando o MATLAB exibe um resultado numérico, ele segue diversas regras. Normalmente, se o resultado é um número inteiro, ele é exibido como tal. Igualmente, quando o resultado é um número real (ponto flutuante), ele é mostrado com aproximadamente quatro dígitos após o ponto decimal. Se os dígitos significativos do resultado estão fora desta faixa então o resultado é exibido em notação científica. O comando `format` é usado para modifcar o formato numérico de exibição, como mostrado na tabela 3.1.

comando	π	resultado
`format short`	3.1416	4 dígitos decimais,
`format bank`	3.14	2 dígitos decimais,
`format long`	3.14159265358979	14 dígitos decimais,
`format short e`	3.1416e+00	4 dígitos decimais e expoente,
`format long e`	3.14159265358793e+00	16 dígitos e expoente,
`format hex`	400921fb54442d18	hexadecimal,
`format +`	+	+, − ou branco para valor positivo, negativo ou nulo,
`format rat`	355/113	aproximação racional,
`format compact`		suprime linhas em branco,
`format loose`		mantém linhas em branco,
`format`		atribui o padrão : `short` e `loose`.

Tabela 3.1: O comando `format`.

3.1.2 Espaço de trabalho

Durante uma sessão, os comandos e variáveis utilizados residem no espaço de trabalho do MATLAB e podem ser armazenados quando desejado. O comando `who` lista o nome das variáveis que estão sendo usadas, ou seja, que estão presentes no espaço de trabalho, enquanto que o comando `whos` fornece maiores informações sobre estas variáveis. As variáveis no espaço de trabalho podem ser removidas incondicionalmente usando o comando `clear`.

`>> clear tempo, raiz` remove as variáveis `tempo` e `raiz`,

`>> clear` remove **todas** as variáveis do espaço de trabalho. **Atenção:** não será solicitada a confirmação. **Todas** as variáveis estarão irremediavelmente removidas.

O comando `save` é usado para gravar as variáveis do espaço de trabalho em um arquivo em disco. Sua sintaxe é

```
save <nome_do_arquivo> <variáveis> <chaves>
```

onde `<nome_do_arquivo>` especifica que as variáveis do espaço de trabalho serão gravadas no arquivo binário `<nome_do_arquivo>.mat`. Por exemplo, `save trab1` faz com que todas as variáveis da sessão sejam gravadas no arquivo `trab1.mat`. Se o nome do arquivo não for dado então todas as variáveis serão gravadas no arquivo *default* `matlab.mat`;
`<variáveis>` definem quais as variáveis do espaço de trabalho a serem gravadas no arquivo `<nome_do_arquivo>.mat`. As variáveis devem estar separadas por brancos;
`<chaves>` especificam as características dos dados que serão gravados no arquivo denominado `<nome_do_arquivo>.mat`. As `<chaves>` podem ser

`-ascii` define que o arquivo será no formato ASCII e seu nome não conterá a extensão `.mat`, que é a forma usada para armazenar arquivos binários.

`-double` antecedido por `-ascii`, define que os números serão gravados com 16 dígitos ao invés de 8.

`-tabs` antecedido por `-ascii`, define que os dados estarão tabulados.

`-append` adiciona variáveis a um arquivo já existente.

As variáveis salvas no arquivo ASCII `<nome_do_arquivo>.<extensão>` são intercaladas em uma única variável denominada `<nome_do_arquivo>`. Ao salvar dados complexos usando a

chave `-ascii` faz com que a parte imaginária dos dados seja perdida pois o MATLAB não pode carregar o dado não numérico `i`.

O comando `load` é usado para recuperar os dados gravados em um arquivo pelo comando `save` e colocá-los no espaço de trabalho. Sua sintaxe é

```
load <nome_do_arquivo>.<extensão>
```

O comando `load` sem argumento recupera os dados do arquivo *default* `matlab.mat`; `load trab1` (sem extensão) recupera os dados do arquivo binário `trab1.mat`. A extensão pode ser usada para recuperar arquivos ASCII. Neste caso, o arquivo deve conter uma matriz de dados com m linhas e n colunas em cada linha. O resultado é uma matriz $m \times n$ com o mesmo nome do arquivo sem a extensão. Os dados em formato ASCII tem que estar na forma de matriz senão o comando `load` não conseguirá recuperá-los.

No exemplo a seguir, as variáveis `a` (1×3), `b` (2×3) e `c` (3×3) são gravadas no arquivo ASCII de nome `trab.dat` usando o comando `save`. Em seguida, o comando `load` é utilizado para recuperar estes dados gerando a variável `trab` (6×3) no espaço e trabalho.

```
>> a=1:3
a =
     1     2     3
>> b=eye(2,3)
b =
     1     0     0
     0     1     0
>> c=magic(3)
c =
     8     1     6
     3     5     7
     4     9     2
>> who   % variaveis no espaco de trabalho
Your variables are:
a         b         c
>> whos   % variaveis no espaco de trabalho
  Name      Size         Bytes  Class
  a         1x3             24  double array
  b         2x3             48  double array
  c         3x3             72  double array
Grand total is 18 elements using 144 bytes
>> save trab.dat a b c -ascii   % grava o arquivo ASCII trab.dat
>> type trab.dat   % mostra o conteudo do arquivo trab.dat
   1.0000000e+00   2.0000000e+00   3.0000000e+00
   1.0000000e+00   0.0000000e+00   0.0000000e+00
   0.0000000e+00   1.0000000e+00   0.0000000e+00
   8.0000000e+00   1.0000000e+00   6.0000000e+00
   3.0000000e+00   5.0000000e+00   7.0000000e+00
   4.0000000e+00   9.0000000e+00   2.0000000e+00
>> load trab.dat
```

```
>> whos   % variaveis no espaco de trabalho
  Name        Size          Bytes  Class
  a           1x3              24  double array
  b           2x3              48  double array
  c           3x3              72  double array
  trab        6x3             144  double array
Grand total is 36 elements using 288 bytes
>> trab
trab =
     1     2     3
     1     0     0
     0     1     0
     8     1     6
     3     5     7
     4     9     2
```

Se o arquivo contiver uma variável com o mesmo nome de uma já existente no espaço de trabalho então o comando `load` faz com que a variável do espaço de trabalho seja substituída pela variável existente no arquivo.

3.1.3 Diário

Todos os comandos solicitados pelo usuário e as respostas fornecidas pelo MATLAB (com excessão de gráficos) podem ser gravados em um arquivo ASCII para que possam ser impressos ou mesmo incluídos em algum relatório, como foi feito neste texto!

Para esta gravação basta usar o comando `diary <nome_do_arquivo>`, a partir do qual a interação usuário/MATLAB será registrada. A suspensão do registro é feita por `diary off` e a reinicialização é obtida por `diary on`. Por esta razão, os únicos nomes de arquivos que não são permitidos são `on` e `off`.

3.1.4 Leitura e gravação de dados

A transferência de dados entre o espaço de trabalho e algum dispositivo de entrada e saída (arquivo em disco, impressora, etc.) aumenta a utilização do MATLAB visto tornar possível, por exemplo, até a troca de informações com um outro programa.

Abertura de arquivo

O comando `fopen` abre um arquivo ou obtém informações sobre os arquivos abertos. Sua sintaxe é

```
[fid,'<mensagem>'] = fopen('<nome_do_arquivo>','<permissão>')
```

Deste modo, o comando `fopen` associa o nome externo do arquivo `'<nome_do_arquivo>'` à unidade `fid` que será utilizada nos comandos de entrada e saída no modo especificado pela `'<permissão>'`. Os caracteres permitidos para `'<permissão>'` estão listados na tabela 3.2.

'<permissão>'	especificação
'r'	Abre o arquivo para leitura.
'r+'	Abre o arquivo para leitura e escrita, mas não cria o arquivo.
'w'	Abre o arquivo para escrita e caso necessário cria o arquivo. Porém, remove o conteúdo do arquivo existente.
'w+'	Abre o arquivo para leitura e escrita e se necessário cria o arquivo. Todavia, remove o conteúdo do arquivo existente.
'a'	Cria e abre um arquivo novo ou abre um arquivo já existente para escrita, anexando ao final do arquivo.
'a+'	Cria e abre um arquivo novo ou abre um arquivo já existente para leitura e escrita, anexando ao final do arquivo.

Tabela 3.2: Atributos de arquivo.

Se '<permissão>' for omitida então será assumido o valor 'r'. Se não for especificado, os arquivos serão abertos em modo binário. Para abrir um arquivo texto, o caracter 't' deve ser adicionado ao caracter de permissão, como em, 'wt' e rt+'. Similarmente, o caracter 'b' pode ser usado para reiterar que um arquivo deve ser binário.

Caso o comando fopen tenha sucesso ao abrir o arquivo, ele retornará o identificador de arquivo fid contendo um número inteiro maior do que 2 e o conteúdo de '<mensagem>' será vazio. O fid é usado com outras rotinas de entrada e saída para identificar o arquivo no qual as operações serão realizadas.

No entanto, se o comando fopen não tiver sucesso então fid = -1 e '<mensagem>' conterá uma cadeia de caracteres informando o tipo de erro ocorrido. O comando help fopen deve ser utilizado para obter mais informações sobre este comando.

Fechamento de arquivo

O comando fclose(fid) fecha o arquivo previamente aberto pelo comando fopen, cujo identificador associado a este arquivo seja fid. O valor 0 é retornado em caso de sucesso no fechamento e -1 no caso de insucesso. Por sua vez, o comando fclose('all') fecha todos os arquivos abertos. Quando um arquivo é fechado, a associação entre o identificador fid e o arquivo físico '<nome_do_arquivo>' é desfeita.

Gravação em arquivo

O comando fprintf grava dados formatados em um arquivo e sua sintaxe é

```
<tamanho> = fprintf(fid,'<formato>',<variável>)
```

onde <tamanho> é o número de *bytes* escritos, fid é o identificador associado ao arquivo no qual será feita a gravação dos valores contidos na <variável> com o formato especificado na cadeia de caracteres '<formato>', a qual deve conter caracteres alfanuméricos e/ou especificações de conversão. Estas especificações de conversão são delimitadas pelo caracter % e uma das letras i, e, f, g ou s, de acordo com a tabela 3.3.

Deste modo,

formato	especificação
%*n*i	usado para valores inteiros, sendo n o tamanho do campo de exibição;
%*n*.*d*f	notação na forma [−]888.888, sendo n o tamanho do campo (número total de caracteres exibidos) e d o número de dígitos decimais;
%*n*.*d*e	notação na forma $[-]8.888 \pm 88$, sendo n o tamanho do campo (número total de caracteres exibidos) e d o número de dígitos decimais;
%*n*.*d*g	equivalente a %*n*.*d*e ou %*n*.*d*f, dependendo de qual formato for mais curto, além disso os zeros insignificantes não são exibidos;
%*n*s	exibe caracteres em um campo de tamanho n.

Tabela 3.3: Formatos de exibição.

```
>> fprintf('a precisao deste computador = %12.5e\n',eps/2)
a precisao deste computador =  1.11022e-16
```

onde \n é usado para começar uma nova linha. Quando for necessário ter o caracter (') exibido basta usá-lo duas vezes

```
>> fprintf('hoje e'' dia %9s\n',date)
hoje e' dia 18-Feb-2000
```

O comando fprintf do MATLAB difere de seu homônimo da linguagem C em um aspecto muito importante, ele é *vetorizado* no caso da variável a ser exibida for uma matriz. Neste caso, o formato é aplicado aos elementos da matriz, por coluna, até a última linha. Ele é então usado de modo similar, sem reinicialização, pelos demais elementos das outras colunas da matriz. Veja a diferença entre os dois comandos fprintf usando a matriz M

```
>> x=1:0.5:2;
>> M=[x;sqrt(x)]'
M =
    1.0000    1.0000
    1.5000    1.2247
    2.0000    1.4142
>> fprintf('%5.2f %10.5f \n',M)
 1.00    1.50000
 2.00    1.00000
 1.22    1.41421
>> fprintf('%5.2f %10.5f \n',M')
 1.00    1.00000
 1.50    1.22474
 2.00    1.41421
```

Se fid for omitido do comando fprintf ou se fid = 1 então os valores serão simplesmente exibidos na tela. No exemplo abaixo, uma tabela contendo x, $\sqrt{x}$ e e^{-x} para $1 \leq x \leq 2$ é gerada e gravada no arquivo sqrtexp.dat.

```
>> x = 1:0.2:2;   % define x
>> tab = [x; sqrt(x); exp(-x)]   % gera tabela
tab =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
```

```
    1.0000    1.0954    1.1832    1.2649    1.3416    1.4142
    0.3679    0.3012    0.2466    0.2019    0.1653    0.1353
>> fid = fopen('sqrtexp.dat','w')   % abre o arquivo para escrita
fid =
     3
>> fprintf(fid,'%5.2f%10.5f%10.4f\n',tab)   % escreve tabela no arquivo
ans =
   156
>> fclose(fid)   % fecha o arquivo
ans =
     0
>> type sqrtexp.dat   % lista o arquivo
 1.00   1.00000    0.3679
 1.20   1.09545    0.3012
 1.40   1.18322    0.2466
 1.60   1.26491    0.2019
 1.80   1.34164    0.1653
 2.00   1.41421    0.1353
```

Um manual da linguagem C deve ser consultado para maiores informações sobre o comando `fprintf`.

Leitura em arquivo

O comando `fscanf` efetua a leitura de dados formatados em um arquivo. Sua sintaxe é

```
[<variável>,<tamanho>] = fscanf(fid,'<formato>',<elementos>)
```

onde `fid` é o identificador associado ao arquivo no qual está sendo feita a leitura dos dados escritos no formato especificado na cadeia de caracteres `'<formato>'`. Os dados são convertidos segundo o `'<formato>'` e atribuídos à `<variável>`. As especificações de conversão são mostradas na tabela 3.3 e o parâmetro `<elementos>` é descrito na tabela 3.4. A variável `<tamanho>` retorna o número de elementos que foram lidos com sucesso.

elementos	especificação
`n`	Lê até `n` valores em um vetor coluna.
`inf`	Lê até o fim do arquivo, resultando em um vetor coluna contendo o mesmo número de elementos do arquivo.
`[m,n]`	Lê os valores suficientes para preencher uma matriz de dimensão `m` × `n`, preenchendo esta matriz por coluna; `n` pode ser igual à `inf` mas não pode ser igual à `m`.

Tabela 3.4: Especificação de elementos para leitura.

Quando o MATLAB estiver lendo um arquivo ele tentará combinar os dados no arquivo com a forma especificada em `'<formato>'`. Se a combinação ocorrer então os dados serão atribuídos por coluna à `<variável>`. No entanto, se somente uma combinação parcial ocorrer então apenas os dados combinados com o `'<formato>'` serão atribuídos à `<variável>` e a operação de leitura será interrompida.

De modo similar ao `fprintf`, o comando `fscanf` do MATLAB difere de seu homônimo da linguagem C em um aspecto muito importante, ele é *vetorizado* ao retornar um argumento matriz. Isto significa que o `'<formato>'` é reciclado através do arquivo até o final deste arquivo ser encontrado ou a quantidade de dados definida em `<tamanho>` tiver sido lida.

```
>> fid = fopen('sqrtexp.dat','r')   % abre arquivo para leitura
fid =
     3
>> [A,n] = fscanf(fid,'%5f%10f%10f',[3 inf])   % leitura das 3 colunas
A =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    1.0000    1.0955    1.1832    1.2649    1.3416    1.4142
    0.3679    0.3012    0.2466    0.2019    0.1653    0.1353
n =
    18
```

Se o caracter `*` for colocado entre o `%` e um caracter de conversão (`d`, `e`, `f`, `g` ou `s`) então o correspondente valor combinado não será armazenado na `<variável>`.

```
>> frewind(fid)   % posiciona leitura para o inicio do arquivo
>> [A,n] = fscanf(fid,'%5f%10f%*10f',[2 inf])   % leitura de 2 colunas
A =
    1.0000    1.2000    1.4000    1.6000    1.8000    2.0000
    1.0000    1.0955    1.1832    1.2649    1.3416    1.4142
n =
    12
>> fclose(fid)   % fecha o arquivo
ans =
     0
```

O comando `frewind(fid)` é usado para posicionar o acesso ao primeiro registro do arquivo cujo identificador seja `fid`.

Durante o processo de leitura é importante verificar se o último registro do arquivo já foi lido. O comando `feof(fid)` faz esta verificação no arquivo de identificador `fid`. Se o último registro já foi lido então será retornado o valor `1`, caso contrário `0` será retornado.

Usar o comando `help iofun` para se conhecer outras funções do MATLAB para leitura e gravação de dados.

3.2 Estruturas condicionais

Uma estrutura condicional permite a escolha do grupo de comandos a serem executados quando uma dada condição for satisfeita ou não, possibilitando desta forma alterar o fluxo natural de comandos. Esta condição é representada por uma expressão lógica.

3.2.1 Estrutura `if-end`

A estrutura condicional mais simples do MATLAB é

```
if <condição>
   <comandos>
end
```

Se o resultado da expressão lógica `<condição>` for `1` (*verdadeiro*) então a lista de `<comandos>` será executada. Se o resultado for `0` (*falso*) os `<comandos>` não serão executados. Considere os comandos

```
a = input('Entre com o valor de a : ');
if a >= 0
   b = sqrt(a)
end
```

e a execução para `a = 2`

```
Entre com o valor de a : 2
b =
    1.4142
```

Neste exemplo, a raiz quadrada de `a` será atribuída a `b` se, e somente se, o valor de `a` for maior ou igual a `0`.

3.2.2 Estrutura `if-else-end`

No caso de haver duas alternativas, uma outra estrutura condicional deve ser usada

```
if <condição>
   <comandos_1>
else
   <comandos_0>
end
```

Se o resultado da expressão lógica `<condição>` for `1` (*verdadeiro*) então a lista contendo `<comandos_1>` será executada. Se `<condição>` for `0` (*falso*) então será a lista `<comandos_0>` a ser executada. A seqüência de comandos

```
a = input('Entre com o valor de a : ');
if a > 0
   b = log(a)
else
   b = exp(a)
end
```

quando executada fornecendo `a = 5` produz

```
Entre com o valor de a : 5
b =
    1.6094
```

Se `a` for positivo então o logaritmo natural de `a` será atribuído a `b` e se `a` for negativo ou nulo então `b` será igual ao exponencial de `a`.

3.2.3 Estrutura `if-elseif-end`

Quando houver mais de duas alternativas, a estrutura `if-else-end` do MATLAB torna-se

```
if <condição_1>
   <comandos_1>
elseif <condição_2>
   <comandos_2>
elseif <condição_3>
   <comandos_3>
     .
     .
     .
else
   <comandos_0>
end
```

A lista `<comandos_1>` será executada se `<condição_1>` for igual a `1` (*verdadeiro*); já a lista `<comandos_2>` será executada se `<condição_2>` for `1` e assim para as outras condições. Se nenhuma das condições for `1` então `<comandos_0>` será executada. Quando a primeira `<condição>` for satisfeita e os `<comandos>` executados, a estrutura `if-elseif-end` será abandonada, ou seja, o controle do processamanto será transferido para o comando imediatamente após o `end`. Os comandos a seguir calculam o valor absoluto de um número real ou complexo

```
a = input('Entre com o valor de a : ');
if imag(a) ~= 0
   b = sqrt(real(a).^2+imag(a).^2);
elseif a < 0
   b = -a;
else
   b = a;
end
[a b]
```

Para `a = 6 + 8i`,

```
Entre com o valor de a : 6+8i
ans =
   6.0000+ 8.0000i  10.0000
```

Deste modo foi executado o primeiro comando para o qual a condição `imag(a) ~= 0` foi satisfeita. Assim, na estrutura `if-elseif-end` uma única lista de comandos é executada.

3.3 Estruturas de repetição

A estrutura de repetição faz com que uma seqüência de comandos seja executada repetidamente até que uma dada condição de interrupção seja satisfeita. O MATLAB possui duas estruturas de repetição, as estruturas `for-end` e a `while-end`.

3.3.1 Estrutura `for-end`

A estrutura `for-end` permite que um grupo de comandos seja repetido um número específico de vezes. Sua sintaxe é

```
for <variável>=<arranjo>
   <comandos>
end
```

onde `<variável>` é a variável-de-controle que assume todos os valores contidos no vetor linha `<arranjo>`. Assim, o número de repetições da lista `<comandos>` é igual ao número de elementos no vetor `<arranjo>`. A variável-de-controle não pode ser redefinida dentro da estrutura `for-end`. Os comandos

```
n = input('Valor de n : '); s = 0; n2 = n^2;
for i = 1:2:2*n-1
   s = s + i;
end, n2, s
```

executados para `n = 5`

```
n = input('Valor de n : '); s = 0; n2 = n^2;
Valor de n : 5
n2 =
    25
s =
    25
```

mostram que a soma dos `n` primeiros números ímpares é igual ao quadrado de `n`, pois para `n = 5` a variável-de-controle `i` assume os valores `1 3 5 7 9`. Deve ser observado o uso do (`;`) para suprimir a exibição de resultados intermediários no cálculo de `s`. Para mostrar que as estruturas `for-end` podem estar encadeadas, considere a seqüência de comandos

```
n = input('Ordem do quadrado magico : ');
A = magic(n);
Soma_Linhas = zeros(n,1); Soma_Colunas = zeros(1,n); Soma_Diagonal = 0;
for i = 1:n
   Soma_Diagonal = Soma_Diagonal + A(i,i);
   for j = 1:n
      Soma_Linhas(i) = Soma_Linhas(i) + A(i,j);
      Soma_Colunas(j) = Soma_Colunas(j) + A(i,j);
   end
end
A, Soma_Linhas, Soma_Colunas, Soma_Diagonal
```

Para `n = 4`

```
Ordem do quadrado magico : 4
A =
    16     2     3    13
     5    11    10     8
     9     7     6    12
```

```
     4    14    15     1
Soma_Linhas =
    34
    34
    34
    34
Soma_Colunas =
    34    34    34    34
Soma_Diagonal =
    34
```

Cumpre observar que o MATLAB possui comandos para determinar estes somatórios de um modo mais simples, pelo uso do comando `sum` que fornece a soma das colunas de uma matriz.

3.3.2 Estrutura `while-end`

A estrutura `while-end`, ao contrário da `for-end`, repete um grupo de comandos um número indefinido de vezes. Sua sintaxe é

```
while <condição>
   <comandos>
end
```

Enquanto a expressão lógica `<condição>` for *verdadeiro* a lista `<comandos>` será repetida. Por exemplo, para determinar a precisão de um computador, os comandos

```
n = 0; Epsilon = 1;
while 1 + Epsilon > 1
   n = n + 1;
   Epsilon = Epsilon / 2;
end
n, Epsilon, eps
```

quando executados fornecem

```
n =
    53
Epsilon =
   1.1102e-16
eps =
   2.2204e-16
```

`Epsilon` é a chamada precisão da máquina ϵ, ou seja, o maior número que somado a 1 é igual a 1. Para computadores com aritmética IEEE $\epsilon = 2^{-53}$. Comparada com a variável especial `eps` $= 2^{-52}$ do MATLAB

```
>> 1+eps-1
ans =
   2.2204e-16
>> 1+Epsilon-1
ans =
     0
```

Note que quando `eps` é somado a 1 resulta em um número maior do que 1. O mesmo não ocorre com `Epsilon`, porque qualquer valor igual ou menor do que ele somado a 1 será simplesmente 1, ou seja, o computador já não consegue mais representar $1 + \epsilon$.

3.3.3 Estrutura com interrupção no interior

A estrutura `while-end` permite que um grupo de comandos seja repetido um número indeterminado de vezes, no entanto, a condição de interrupção é testada no início da estrutura. Em várias situações em programação se faz necessário interromper a execução da repetição verificando a condição no interior da estrutura e não no seu início.

O comando `break` interrompe a execução de uma estrutura `while-end` ou `for-end` e transfere a execução para o comando imediatamente seguinte ao `end`. Em repetições aninhadas, o `break` interrompe a execução apenas da estrutura mais interna. Assim, uma repetição com condição de interrupção no interior pode ter a forma

```
while 1
   <comandos_1>
   if <condição>
      break
   end
   <comandos_2>
end
```

A estrutura `while-end` é executada indefinidamente a princípio pois a condição do `while` é sempre *verdadeiro*. Contudo, quando a `<condição>` do `if` for satisfeita o comando `break` será executado causando a interrupção da repetição `while-end`. O conjunto de comandos

```
while 1
   a=input('Entre com a, a>0 : ');
   if a <= 0
      break
   end
   disp(rats(a))
end
```

lista continuamente a representação racional de um número fornecido enquanto este for positivo. Deste modo, para `a =` π e `a =` $\sqrt{2}$

```
Entre com a, a>0 : pi
    355/113
Entre com a, a>0 : sqrt(2)
   1393/985
Entre com a, a>0 : 0
```

3.4 Arquivos M

Além de se poder entrar com cada comando de uma vez, o MATLAB permite, como qualquer outra linguagem de programação, que seja executada uma seqüência de comandos escrita em

um arquivo. Por esta razão este tipo de arquivo é chamado de roteiro (*script*) e deve possuir a extensão `.m`. Os arquivos de roteiro são particularmente úteis para entrar com matrizes e grande quantidade de dados; deste modo, torna-se fácil conferir e modificar os seus valores. Serão apresentados dois tipos de roteiros: programa e função.

3.4.1 Programa

Um programa de arquivo M tem acesso à todas as variáveis no espaço de trabalho e as variáveis criadas por ele farão parte do espaço de trabalho. Alguns comandos do MATLAB são particularmente úteis para facilitar a interação programa-usuário.

O comando `<variável>=input('<texto>')` exibe `<texto>` e espera que um valor seja atribuído à `<variável>` e `error('<mensagem>')` causa a interrupção da execução de um programa e exibe `<mensagem>`.

Considere o programa no arquivo `decsomat.m` para gerar uma matriz com elementos aleatórios entre -100 e 100 e decompô-la na soma de três matrizes: uma triangular inferior, uma diagonal e outra triangular superior

```
n = input('Ordem da matriz : ');
A = fix(200*(rand(n)-0.5*ones(n)));
D = diag(diag(A));
L = tril(A) - D;
U = triu(A) - D;
A, L, D, U
```

Estes arquivos M podem ser criados usando um editor de texto qualquer e para executar um programa basta entrar com o nome do arquivo que o contenha

```
>> decsomat
Ordem da matriz : 3
A =
    90    -2    -8
   -53    78   -96
    21    52    64
L =
     0     0     0
   -53     0     0
    21    52     0
D =
    90     0     0
     0    78     0
     0     0    64
U =
     0    -2    -8
     0     0   -96
     0     0     0
```

O comando `echo on` é usado para que os comandos do arquivo sejam mostrados durante a execução; para suprimir a exibição basta usar `echo off`.

Para exibir uma janela contendo um menu de escolha para o usuário é utilizado o comando `menu`. Sua sintaxe é

```
<variável> = menu('<título>','<opção_1>',...,'<opção_n>')
```

A cadeia de caracteres `'<título>'` será apresentada na parte superior da janela do menu e as `n` opções `<opção_i>` serão colocadas à disposição do usuário para que ele escolha uma delas. Se a `k`-ésima opção for escolhida então este valor de `k` será atribuído a `<variável>`. No exemplo a seguir, a escolha do método de Newton (terceira opção do menu) faz com que a variável `m` seja igual a 3.

```
>> m = menu('Escolha o esquema :','secante','regula falsi','Newton')
m =
     3
```

Para interromper a execução de um programa até que qualquer tecla seja acionada basta usar o comando `pause`. Para que a interrupção seja por `n` segundos, ao invés de esperar pelo acionamento de tecla, deve ser usado `pause (n)`. O comando `pause` é especialmente útil quando da exibição de vários gráficos.

```
x = -10:0.1:10;
for n = 1:5
   plot(x,sin(x).*x.^n)
   pause
end
```

Os comandos `pause off` e `pause on` causam a desativação e ativação, respectivamente, de interrupções pelo `pause` ou `pause (n)`.

3.4.2 Comandos para execução de expressões

Antes de ser apresentado o segundo tipo de arquivo M (`function`) se faz necessário mostrar os comandos para execução de expressões. O MATLAB possui dois comandos com este propósito: `feval` e `eval`, os quais são de grande valia principalmente para o usuário criar novas `function`'s. O comando

$$[v_1,\ v_2, \ldots, v_m] \texttt{ = feval('<função>', } a_1,\ a_2, \ldots, a_n)$$

atribui os n argumentos $a_1,\ a_2, \ldots, a_n$ à uma função e retorna o resultado nas m variáveis $v_1,\ v_2, \ldots, v_m$. A função é especificada pela cadeia de caracteres `'<função>'` que pode ser uma expressão ou mesmo o nome de um arquivo contendo uma `function` onde a expressão está definida. Por isto os dois comandos a seguir para calcular resto de divisão entre dois números são equivalentes

```
>> x = rem(5,3)
x =
     2
>> y = feval('rem',5,3)
y =
     2
```

Por outro lado, o comando `eval('<expressão>')` interpreta a expressão definida pela cadeia de caracteres `'<expressão>'`.

```
>> r = eval('sin(pi/2)')
r =
     1
>> x = 1; y = eval('2*x+exp(x-1)')
y =
     3
```

Note que para avaliar uma expressão que possui a variável `x`, esta tem que ser previamente definida. Um uso efetivo de `eval` é mostrado na `function pegaso` da seção 3.4.3. A função `eval` é também capaz de executar tarefas mais complexas, para tal considere o programa contido no arquivo `tabeval.m`

```
Tabela = ['sqrt(x)'
          'exp(x) ' % Observe o caractere branco entre ) e '
          '1/x+5*x'];
n = input('Escolha a expressao : ');
x = input('Defina o argumento  : ');
a = eval(Tabela(n,:))
```

Para executá-lo

```
>> tabeval
Escolha a expressao : 3
Defina o argumento  : 2
a =
   10.5000
```

Neste exemplo, como foi escolhido `n = 3` então a terceira expressão será utilizada. Para `x = 2` tem-se que `a = 1/2+5*2 = 10.5000`. Considerando que a variável `Tabela` é uma matriz de caracteres, ela deve então ter o mesmo número de colunas (ver seção 2.1.3 na página 16).

3.4.3 Subprograma `function`

Um outro tipo de arquivo de roteiro é usado para o próprio usuário criar novas funções para o MATLAB. Por exemplo, seja a função para calcular as duas raízes de uma parábola

```
function raizes = parabola(a,b,c)
%PARABOLA   raizes de uma parabola.
%           PARABOLA(A,B,C) calcula as duas raizes da parabola
%           P(X) = A*X^2 + B*X + C = 0
%           retornando-as no vetor RAIZES.

delta = sqrt(b^2-4*a*c);
raizes(1) = (-b + delta ) / (2*a);
raizes(2) = (-b - delta ) / (2*a);
```

Além de aumentar a flexibilidade, os arquivos de função tem o mesmo status que as outras funções do MATLAB.

```
>> lookfor parabola
PARABOLA   raizes de uma parabola.
```

```
>> help parabola
 PARABOLA   raizes de uma parabola.
            PARABOLA(A,B,C) calcula as duas raizes da parabola
            P(X) = A*X^2 + B*X + C = 0
            retornando-as no vetor RAIZES.
```

O nome da função tem que ser igual ao nome do arquivo M onde ela está definida, mas sem a extensão `.m`, ou seja, a função `parabola` descrita acima deve estar no arquivo `parabola.m`. Um arquivo pode conter mais que uma `function`, sendo que a primeira delas deve ter o nome desse arquivo. As outras `function`'s só podem ser chamadas pela primeira delas, não sendo possível serem chamadas por `function`'s escritas em um outro arquivo. Na realidade várias funções do MATLAB são arquivos M. Ao contrário do programa no qual as variáveis são globais, em uma `function` as variáveis são locais, ou seja, elas não tem acesso e nem podem criar variáveis no espaço de trabalho. Por sua vez, os parâmetros de saída (`raizes` no caso da `function parabola`) são criados no espaço de trabalho do MATLAB.

Os comandos `nargin` e `nargout` retornam o número de argumentos de entrada e saída, respectivamente, especificados pelo usuário para uma `function`. A `function pegaso` calcula o zero pertence ao intervalo `[a b]` da função definida na cadeia de caracteres `Funcao`, utilizando o robusto e eficiente método pégaso. Nesta `function` é utilizado o comando `eval` para avaliação de expressão, o qual é descrito na seção 3.4.2.

```
function [Raiz,Erro,Iter] = pegaso(funcao,a,b,Toler,IterMax,Exibe)
%PEGASO   Calculo de raiz de equacao pelo metodo pegaso.
%
%         PEGASO(FUNCAO,A,B,TOLER,ITERMAX,EXIBE) calcula a raiz de uma
%         equacao F(X)=0 contida no intervalo [A B] com tolerancia TOLER
%         e com no maximo ITERMAX iteracoes, usando o metodo pegaso,
%         sendo a funcao F(X) dada na cadeia de caracteres FUNCAO.
%         EXIBE especifica se os resultados intermediarios serao
%         mostrados, 0: nao exibe e 1: exibe.
%         FUNCAO, A e B sao necessarios enquanto que TOLER, ITERMAX e
%         EXIBE terao valores pre-definidos caso nao sejam fornecidos.
%
if nargin < 3, error('Numero de argumentos insuficientes'), end
if nargin < 4, Toler = 1e-10; end
if nargin < 5, IterMax = 100; end
if nargin < 6, Exibe = 0; end
x = a; Fa = eval(funcao); x = b; Fb = eval(funcao);
if Exibe ~= 0
   disp('        Calculo de raiz de equacao pelo metodo pegaso')
   disp('iter     a          b          x         Fx            delta_x')
end
k = 0; x = b; Fx = Fb;
while 1
   k = k + 1; DeltaX = -Fx / (Fb - Fa) * (b - a);
   x = x + DeltaX; Fx = eval(funcao);
   if Exibe ~= 0
```

```
      fprintf('%3i%11.5f%11.5f%11.5f%14.5e%14.5e\n',k,a,b,x,Fx,DeltaX);
   end
   if ((abs(DeltaX) < Toler & abs(Fx) < Toler) | k >= IterMax), break, end
   if Fx*Fb < 0 a = b; Fa = Fb; else Fa = Fa * Fb / (Fb + Fx); end
   b = x; Fb = Fx;
end
Raiz = x;
if nargout > 1, Erro = abs(DeltaX) >= Toler | abs(Fx) >= Toler; end
if nargout > 2, Iter = k; end
```

Os argumentos `Funcao`, `a` e `b` devem ser fornecidos senão uma mensagem de erro será exibida e a execução da `function` interrompida. No entanto, os argumentos `Toler` (tolerância da raiz), `IterMax` (número máximo de iterações) e `Exibe` (exibe resultados intermediários) são opcionais; caso não sejam incluídos na lista de argumentos de entrada serão atribuídos valores pré-definidos. Se forem especificados mais de seis argumentos de saída haverá a exibição de uma mensagem de erro e a interrupção da `function`.

Se nenhum ou apenas um argumento de saída for especificado então será retornado a raiz da equação na variável `Raiz`. Se forem dois argumentos então além da `Raiz` será retornado a condição de erro na variável `Erro`. Se forem três argumentos de saída então serão retornados `Raiz`, `Erro` e o número de iterações `Iter`. Mais de três argumentos de saída causam a exibição de uma mensagem de erro e a não execução da `function`.

Para calcular a raiz de $f(x) = \cos(x^2 - 1)\sqrt{x+1} = 0$ pertencente ao intervalo [0 2], com tolerância $\epsilon = 10^{-1}$, com no máximo 10 iterações, listando os resultados intermediários e retornado a raiz, a condição de erro e o número de iterações

```
>> [r,e,i] = pegaso('cos(x^2-1)*sqrt(x+1)',0,2,1e-1,10,1)
        Calculo de raiz de equacao pelo metodo pegaso
iter     a          b          x          Fx            delta_x
  1    0.00000    2.00000    0.47920   8.72828e-01  -1.52080e+00
  2    2.00000    0.47920    0.99219   1.41128e+00   5.12995e-01
  3    2.00000    0.99219    1.68045  -4.09987e-01   6.88254e-01
  4    0.99219    1.68045    1.52552   3.83307e-01  -1.54933e-01
  5    1.68045    1.52552    1.60038   1.54647e-02   7.48614e-02
r =
    1.6004
e =
     0
i =
     5
```

Por sua vez, calculando a mesma raiz com os argumentos de entrada opcionais previamente atribuídos, ou seja, tolerância $\epsilon = 10^{-10}$, máximo de 100 iterações, não listando os resultados intermediários e além disto retornado somente a raiz e a condição de erro

```
>> [r,e] = pegaso('cos(x^2-1)*sqrt(x+1)',0,2)
r =
    1.6034
```

```
e =
     0
```

Conforme já mencionado, a variáveis de uma `function` são locais, ou seja, só podem ser referenciadas internamente, não sendo reconhecidas pelo espaço de trabalho e outras `function`'s. No entanto, além do uso de argumentos, um outro modo de trocar informações entre o espaço de trabalho e `functions`'s é pelo uso de variáveis globais. A declaração

```
global <lista_de_variáveis>
```

faz com que as variáveis especificadas na `<lista_de_variáveis>` tornem-se globais e portanto esta declaração deve aparecer no programa e nas `function`'s de interesse. O uso de variáveis globais torna mais difícil o entedimento e a modificação das `function`'s, além de tornar os módulos do programa menos independentes. Por estas razões, a utilização de variáveis globais deve ser evitada.

O comando `return` colocado dentro de uma `function` causa um retorno normal para o comando seguinte àquele que chamou a `function`. Ele é útil quando da consistência dos dados.

3.4.4 Depuração de programa

O comando `keyboard` interrompe a execução de um programa e transfere o controle para o teclado. Este modo especial, indicado pela letra `K` antes dos caracteres `>>`, permite listar variáveis ou mesmo executar qualquer comando do MATLAB. O modo criado pelo `keyboard` termina com a execução do comando `return`.

3.4.5 Gerenciamento de arquivos

O MATLAB fornece diversos comandos para gerenciamento de arquivos, os quais são mostrados na tabela 3.5.

comando	descrição
`what`	lista os arquivos `.m`, `.mat` e `.mex` do diretório corrente;
`dir` ou `ls`	lista os arquivos do diretório corrrente;
`type <nome_do_arquivo>`	exibe o arquivo `<nome_do_arquivo>.m`;
`delete <nome_do_arquivo>`	remove o arquivo `<nome_do_arquivo>`;
`cd <dir>` ou `chdir <dir>`	muda o diretório para `<dir>`;
`cd` ou `chdir` ou `pwd`	mostra o diretório corrente;
`which <nome_do_arquivo>`	mostra em que diretório está `<nome_do_arquivo>.m`.

Tabela 3.5: Comandos para gerenciamento de arquivos.

3.5 Exercícios

Seção 3.1

3.1. Verificar as diferenças entre os formatos de exibição para as variáveis `a=sqrt(2)`, `e=exp(10)`, `x=1:10`, `y=x'` e `M=rand(3)`:

```
a, M, disp(a), disp(M),format bank, x, disp(x), format short e, a, e, x,
format compact, a, e,x, format, fprintf('%10.5f %12.3e\n',a,e),
fprintf('%5.1f\n',x), fprintf('%5.1f\n',y)
```

3.2. Observar os resultados dos comandos para controle do espaço de trabalho utilizando as variáveis do exercício 3.1:

```
who, whos, save, clear, who, load, whos,
save esptrab x y, clear, who, load esptrab, whos, x, y
```

3.3. Gravar os comandos do exercício 3.2 no arquivo `diario` e usar `type diario` para ver o seu conteúdo.

3.4. Gerar uma tabela de x, $\text{sen}(x)$, $0 \leq x \leq \pi/2$ e gravá-la no arquivo `seno.dat`.

3.5. Acrescentar o valor $\pi + 1$, $\text{sen}(\pi + 1)$ na última linha do arquivo `seno.dat` gravado no exercício 3.4.

Seção 3.2

3.6. Qual o valor de `b` da estrutura `if-end` mostrada na seção 3.2.1 para `a = -1`?

3.7. Encontrar o valor de `b` da estrutura `if-else-end` da seção 3.2.2 para `a = 10`.

3.8. Definir o valor de `b` da estrutura `if-else-end` da seção 3.2.2 para `a = 0`.

3.9. Achar o valor de `b` da estrutura `if-elseif-end` da seção 3.2.3 para `a = -3`.

3.10. Qual o valor de `b` da estrutura `if-elseif-end` da seção 3.2.3 para `a = 3+4i`?

Seção 3.3

3.11. Determinar o valor de `s` da estrutura `for-end` mostrada na seção 3.3.1 para `n = 7`.

3.12. Calcular a soma das linhas, das colunas e da diagonal de um quadrado mágico de ordem 3.

3.13. Explicar como é feita a determinação da precisão de um computador.

3.14. Escrever os comandos para calcular a norma-2 de um vetor `x` com `n` elementos e comparar com a função `norm` do MATLAB.

3.15. Definir os comandos para calcular a norma-1 de uma matriz `A` de ordem `n` e comparar com a função `norm`.

Seção 3.4

3.16. Escrever uma `function` para calcular o valor absoluto de um número real ou complexo.

3.17. Dadas as matrizes triangulares $U = \{u_{ij}\}$ e $L = \{l_{ij}\}$ e os vetores b e c, escrever uma `function` para calcular x tal que $Ux = b$ e outra para calcular y tal que $Ly = c$:

$$\begin{pmatrix} u_{11} & u_{12} & u_{13} & \cdots & u_{1n} \\ & u_{22} & u_{23} & \cdots & u_{2n} \\ & & u_{33} & \cdots & u_{3n} \\ & & & & \vdots \\ & & & & u_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_n \end{pmatrix} \quad \text{e} \quad \begin{pmatrix} l_{11} & & & & \\ l_{21} & l_{22} & & & \\ l_{31} & l_{32} & l_{33} & & \\ \vdots & \vdots & \vdots & & \\ l_{n1} & l_{n2} & l_{n3} & \cdots & l_{nn} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_n \end{pmatrix}$$

3.18. Criar o arquivo `almaxmin.m`

```
n=input('Ordem da matriz : ');
A=rand(n);
maior=max(max(A))
menor=min(min(A))
```

3.19. Executar o programa acima para `n` = 5, 10 e 30.

3.20. Gerenciar o diretório usando

```
what, dir, which almaxmin, type almaxmin, delete almaxmin.m, what
```

Capítulo 4

MATLAB para Computação Científica

4.1 Medidas de tempo e operações

O MATLAB provê dois modos de analisar o desempenho de um dado método: o tempo de execução e o número de operações aritméticas.

Usando o `tic-toc` é possível saber o tempo gasto para a execução de um grupo de comandos. A sua sintaxe é

```
tic
   <comandos>
<variável>=toc
```

O `tic` inicia a contagem do tempo e o `toc` fornece o tempo, em segundos, passado desde o último `tic`. A atribuição do tempo gasto a `<variável>` é opcional. No entanto, se o computador estiver executando várias tarefas simultaneamente, `tic-toc` pode não ser uma medida muito confiável. Uma medida mais confiável é obtida por `cputime` que fornece o tempo de CPU (unidade central de processamento) que é o tempo gasto para execução de operações aritméticas e lógicas. A sua sintaxe é

```
<variável_1>=cputime
   <comandos>
<variável_2>=cputime-<variável_1>
```

Por sua vez a função `flops` fornece o número acumulado de operações de ponto flutuante, bastando executar o comando `flops(0)` para zerar o contador. Cumpre observar que `flops` não conta exatamente todas as operações mas sim as que o MATLAB considera mais importantes. Além disso, a avaliação de uma função elementar (seno, exponencial, etc) para argumento real é contado apenas como uma operação aritmética, o que está longe de ser verdade. Portanto, `flops` deve ser usado com cautela. Por exemplo, considere a execução do programa no arquivo `medidas.m`

```
A=magic(350); B=rand(350);
tic; t=cputime; flops(0);
A*B;
operacoes=flops;
tic_toc=toc;
```

```
tempo_de_cpu=cputime-t;
megaflops=operacoes*1.0e-6/tempo_de_cpu;
operacoes, tic_toc, tempo_de_cpu, megaflops
```

Os resultados produzidos em uma estação de trabalho SUN 4 foram

```
>> medidas
operacoes =
    85750000
tic_toc =
   23.8804
tempo_de_cpu =
   22.8167
megaflops =
    3.7582
```

A diferença de tempo produzida por `tic-toc` e `cputime` pode ser bem significativa se o computador estiver executando outros programas ao mesmo tempo. Observe que `toc` deve estar após `flops` pois ele requer 18 operações. É sabido que a multiplicação matricial entre duas matrizes de ordem n demanda $2n^3 - n^2$ operações, no entanto o MATLAB só considera o termo de maior grau ($2 \cdot 350^3 = 85750000$).

4.2 Álgebra linear

O nome MATLAB é um acronismo de MATrix LABoratory (laboratório de matriz), e certamente é na área de álgebra linear que o MATLAB apresenta suas maiores pontencialidades. Nesta seção serão mostrados alguns comandos do programa relativos aos tópicos usualmente abordados em textos de Cálculo Numérico. Para um conhecimento mais amplo dos comandos do MATLAB com respeito à vetores e matrizes usar o comando `help matfun`.

4.2.1 Decomposições e inversa

A função `chol(X)` fatora uma matriz simétrica definida positiva pelo método de Cholesky. Há dois usos, `U=chol(X)` produz uma matriz triangular superior `U` tal que `U'*U=X`. Se `X` não for definida positiva então será emitida uma mensagem de erro. Por outro lado, `[U,p]=chol(X)` nunca produz uma mensagem de erro. Se `X` for definida positiva então `p=0` e `U` será igual ao mencionado acima. Porém, se `X` não for definida positiva então `p` será um inteiro positivo e `U` será uma matriz triangular superior de ordem `p-1` tal que `U'*U=X(1:p-1,1:p-1)`. Por exemplo,

```
>> A=[4 -2 2; -2 10 -7; 2 -7 30];
>> U=chol(A)
U =
     2    -1     1
     0     3    -2
     0     0     5
>> B=[1 -2 4; -2 5 3; 4 3 8]
B =
     1    -2     4
```

```
    -2     5     3
     4     3     8
>> U=chol(B)
 ??? Error using ==> chol
Matrix must be positive definite.

>> [U,p]=chol(B)
U =
     1    -2
     0     1
p =
     3
>> U'*U
ans =
     1    -2
    -2     5
```

A função `lu(X)` faz a decomposição *LU* da matrix `X` usando pivotação parcial. O uso do comando `[L,U,P]=lu(X)` gera uma matriz triangular inferior unitária `L`, uma matriz triangular superior `U` e uma matriz de permutações `P`, tal que `P*X=L*U`. Contudo, o uso de `[E,D]=lu(X)` produz `D=U` e `P*E=L` de modo que `X=E*D`. Por exemplo,

```
>> M=[2 -3 5; 4 1 -1; 1 8 6]
M =
     2    -3     5
     4     1    -1
     1     8     6
>> [L,U,P]=lu(M)
L =
    1.0000         0         0
    0.2500    1.0000         0
    0.5000   -0.4516    1.0000
U =
    4.0000    1.0000   -1.0000
         0    7.7500    6.2500
         0         0    8.3226
P =
     0     1     0
     0     0     1
     1     0     0
>> R=P*M-L*U   % P*M = L*U
R =
     0     0     0
     0     0     0
     0     0     0
>> [E,D]=lu(M)
E =
    0.5000   -0.4516    1.0000
```

```
    1.0000         0         0
    0.2500    1.0000         0
D =
    4.0000    1.0000   -1.0000
         0    7.7500    6.2500
         0         0    8.3226
>> R=M-E*D   % M = E*D
R =
     0     0     0
     0     0     0
     0     0     0
```

A função `inv(X)` calcula a inversa de `X`, tal que `X*inv(X)=eye(size(X))`. Para a matriz M acima

```
>> N=inv(M)
N =
    0.0543    0.2248   -0.0078
   -0.0969    0.0271    0.0853
    0.1202   -0.0736    0.0543
>> round(M*N)
ans =
     1     0     0
     0     1     0
     0     0     1
```

A função `qr(X)` faz a decomposição QR da matriz `X` de dimensão $m \times n$. Esta fatorização pode ser usada tanto para matriz quadrada quanto para matriz retangular. O uso de `[Q,R]=qr(X)` produz uma matriz ortonormal `Q` de ordem m (`Q'*Q=eye(size(Q))`) e uma matriz triangular superior `R` com a mesma dimensão $m \times n$ de `X`, de modo que `X=QR`. Já `[Q,R,P]=qr(X)` produz uma matriz ortogonal `Q`, uma matriz triangular superior `R` com elementos da diagonal decrescentes e uma matriz de permutações `P` de ordem n, tal que `X*P=Q*R`. Por exemplo,

```
>> A=[2 -4; 3 5; 1 -7; 8 6]
A =
     2    -4
     3     5
     1    -7
     8     6
>> [Q,R]=qr(A)
Q =
   -0.2265    0.5326   -0.5446   -0.6071
   -0.3397   -0.3211    0.5754   -0.6711
   -0.1132    0.7754    0.5906    0.1927
   -0.9058   -0.1096   -0.1535    0.3794
R =
   -8.8318   -5.4349
         0   -9.8215
         0         0
```

```
         0         0
>> round(Q'*Q)   % Q e' ortonormal
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> round(A-Q*R)   % A = Q*R
ans =
     0     0
     0     0
     0     0
     0     0
>> [Q,R,P]=qr(A)
Q =
   -0.3563   -0.4560   -0.8085   -0.1071
    0.4454   -0.1417    0.0007   -0.8840
   -0.6236   -0.4745    0.5740   -0.2377
    0.5345   -0.7395    0.1301    0.3880
R =
   11.2250    4.2762
         0   -7.7275
         0         0
         0         0
P =
     0     1
     1     0
>> round(A*P-Q*R)   % A*P = Q*R
ans =
     0     0
     0     0
     0     0
     0     0
```

A função `svd(X)` faz a decomposição do valor singular de uma matriz `X` de dimensão $m \times n$. O comando `[U,S,V]=svd(X)` produz uma matriz ortonormal `U` de ordem m, uma matriz diagonal `S` de dimensão $m \times n$ contendo os valores singulares de `X` em ordem decrescente e uma matriz ortonormal `V` de ordem n, de modo que `X=U*S*V'`. Para se obter apenas os valores singulares de `X` em um vetor `s` basta `s=svd(X)`. O uso de `[U,S,V]=svd(X,0)` produz uma decomposição mais econômica pois para $m > n$ ter-se-á apenas as n primeiras colunas de `U` e `S` será de ordem n. Por exemplo,

```
>> A=[2 -4; 3 5; 1 -7; 8 6];
>> [U,S,V]=svd(A)
U =
   -0.1884   -0.5472   -0.5446   -0.6071
    0.4673    0.0110    0.5754   -0.6711
   -0.4351   -0.6517    0.5906    0.1927
```

```
    0.7462   -0.5251   -0.1535    0.3794
S =
   12.4766         0
         0    6.9523
         0         0
         0         0
V =
    0.5257   -0.8507
    0.8507    0.5257
>> round(U'*U)   % U e' ortonormal
ans =
     1     0     0     0
     0     1     0     0
     0     0     1     0
     0     0     0     1
>> round(V'*V)   % V e' ortonormal
ans =
     1     0
     0     1
>> round(A-U*S*V')   % A = U*S*V
ans =
     0     0
     0     0
     0     0
     0     0
>> s=svd(A)
s =
   12.4766
    6.9523
>> [U,S,V]=svd(A,0)
U =
   -0.1884   -0.5472
    0.4673    0.0110
   -0.4351   -0.6517
    0.7462   -0.5251
S =
   12.4766         0
         0    6.9523
V =
    0.5257   -0.8507
    0.8507    0.5257
```

4.2.2 Solução de sistemas

Dado um sistema $Ax = b$, a solução x pode ser facilmente calculada pelo MATLAB pelo operador \, por exemplo, para um sistema com matriz simétrica

```
>> A=[4 -2 2; -2 10 -7; 2 -7 30]
```

```
A =
     4    -2     2
    -2    10    -7
     2    -7    30
>> b=[8 11 -31]'
b =
     8
    11
   -31
>> x=A\b
x =
     3
     1
    -1
>> r=(b-A*x)'
r =
     0     0     0
```

No caso de uma matriz não-simétrica

```
>> A=[1 -3 2; -2 8 -1; 4 -6 5]
A =
     1    -3     2
    -2     8    -1
     4    -6     5
>> b=[11 -15 29]'
b =
    11
   -15
    29
>> x=A\b
x =
    2.0000
   -1.0000
    3.0000
>> r=(b-A*x)'
r =
     0     0     0
```

4.2.3 Autovalores e autovetores

Os autovalores são as raízes do polinômio característico da matriz. A função `poly` constrói o polinômio característico de uma matriz. Por exemplo,

```
>> A=[4 -1 3; -1 5 2; 3 2 8]
A =
     4    -1     3
    -1     5     2
     3     2     8
```

```
>> p=poly(A)
p =
    1.0000  -17.0000   78.0000  -79.0000
>> e=roots(p)
e =
    9.9734
    5.6162
    1.4104
```

A função `eig(A)` permite o cálculo do autosistema de A (autovalores e autovetores) por um método mais eficiente do que construindo o polinômio característico. O uso de `[V,D]=eig(A)` produz uma matriz diagonal `D` contendo os autovalores λ e uma matriz `V` com os correspondentes autovetores. Para obter apenas os autovalores em um vetor `d` basta `d=eig(A)`. Se a matriz for simétrica então os autovetores são mutuamente ortogonais (`V'*V=eye(size(A))`). Para a matriz simétrica `A` definida acima

```
>> [V,D]=eig(A)
V =
   -0.5463    0.7387    0.3948
    0.8375    0.4732    0.2734
   -0.0151   -0.4799    0.8772
D =
    5.6162         0         0
         0    1.4104         0
         0         0    9.9734
>> d=eig(A)
d =
    5.6162
    1.4104
    9.9734
>> round(V'*V)
ans =
     1     0     0
     0     1     0
     0     0     1
```

4.2.4 Normas e condição

O MATLAB possui uma função para calcular diversas normas de vetores e matrizes, bem como uma função para determinar o número de condição de matrizes. A função `norm(X,p)` fornece a norma `p` de um vetor ou matriz `X`. No caso de um vetor `x`, `p` pode ser qualquer número (inteiro ou real, positivo ou negativo), além dos caracteres `inf` (`norm(x,inf)=max(abs(x))`) e `-inf` (`norm(x,-inf)=min(abs(x))`). Se o parâmetro `p` for omitido então é assumido `p=2`, ou seja, `norm(x)=norm(x,2)`. No caso de uma matriz `X`, `p` pode ser somente `1`, `2`, `inf` ou `'fro'` (Frobenius) e se ele for omitido também é assumido `p=2` (`norm(X)=norm(X,2)`). Por exemplo,

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

```
>> norm(x,2), norm(x)
ans =
    7.4162
ans =
    7.4162
>> norm(x,1)
ans =
    15
>> norm(x,inf)
ans =
     5
>> norm(A,2), norm(A)
ans =
   16.8481
ans =
   16.8481
>> norm(A,'fro')
ans =
   16.8819
```

A função `dot(a,b)` calcula o produto interno entre os vetores `a` e `b`, os quais devem ter o mesmo tamanho. Se `A` e `B` forem matrizes então `dot(A,B)` produzirá um vetor linha contendo o produto interno entre as correspondentes colunas de `A` e `B`. Assim,

```
>> v=1:3;
>> y=-3:-1;
>> d=dot(v,y)
d =
   -10
```

A função `cond(X)` calcula o número de condição da matriz `X` definido em termos da norma-2, ou seja,

$$\texttt{cond(X)} = \kappa_2(X) = \|X\|_2\|X^{-1}\|_2$$

Para a matrix

```
>> B=[5 3 0; -1 8 6; 4 2 9];
>> cond(B)
ans =
    2.6641
```

A função `rcond(X)` fornece uma estimativa do recíproco do número de condição de `X` definido em termos da norma-1. Se `X` for uma matriz bem-condicionada então `rcond(X)` é próximo de 1 e se `X` for mal-condicionada `rcond(X)` será próximo de 0. Para a matriz `B` acima

```
>> r=rcond(B)
r =
    0.3378
>> 1/r
ans =
    2.9605
```

A função `det(X)` calcula o determinante da matriz `X`. Para `A` e `B` definidas previamente,

```
>> det(A)
ans =
     0
>> det(B)
ans =
   399
```

A função `trace(M)` determina o traço da matriz `M`, isto é, a soma dos elementos da sua diagonal principal. Para a matriz `A` definida anteriormente,

```
>> t = trace(A)
t =
    15
```

4.3 Cálculo de zero e mínimo de função

O MATLAB possui funções para manipular polinômios, calcular raízes de equações e determinar os mínimos de funções. Para mais informações sobre estes tópicos usar `help polyfun`.

4.3.1 Operações com polinômios

O MATLAB fornece várias funções que permitem que as operações envolvendo polinômios seja feita de um modo bem simples. No MATLAB, os coeficientes de um polinômio são definidos como elementos de um vetor como por exemplo, $P(x) = c_1x^n + c_2x^{n-1} + \ldots + c_nx + c_{n+1}$ é representado pelo vetor `c`.

Avaliação

Um polinômio pode ser avaliado no MATLAB pela função `polyval(c,x)`, onde o vetor `c` contém os coeficientes do polinômio e `x` contém os pontos nos quais ele deve ser avaliado. Por exemplo, para avaliar $P(x) = 3x^5 - 2x^4 + 5x^3 + 7x^2 - 3x + 1$ nos pontos $x = 1, 2, 3, 4$ e 5

```
>> c = [3 -2 5 7 -3 1];
>> x = 1:5;
>> y = polyval(c,x)
y =
          11          127          757         2981         8911
```

Adição e subtração

Apesar do MATLAB não dispor de uma função para adicionar polinômios, esta operação pode ser efetuada pela adição vetorial. Assim, para somar os polinômios $a(x) = 5x^3 - 4x^2 + 1$ e $b(x) = 2x^3 + 5x^2 - x - 4$ basta

```
>> a = [5 -4 0 1];
```

```
>> b = [2 5 -1 -4];
>> c = a + b
c =
     7     1    -1    -3
```

resultando $c(x) = 7x^3 + x^2 - x - 3$; e para subtrair

```
>> d = a - b
d =
     3    -9     1     5
```

Se os polinômios tiverem graus diferentes então o vetor correspondente ao de menor grau deve ser preenchido com zeros à esquerda de modo que ambos vetores tenham o mesmo tamanho. Por exemplo, para somar $e(x) = 2x^3 + 4x^2 + 5x - 1$ e $f(x) = -3x + 2$

```
>> e = [2 4 5 -1];
>> f = [0 0 -3 2];
>> g = e + f
g =
     2     4     2     1
```

Multiplicação

Se os vetores `a` e `b` contém os coeficientes dos polinômios $a(x)$ e $b(x)$, respectivamente, então os coeficientes do polinômio resultante da multiplicação de $a(x)$ por $b(x)$ pode ser obtido pela função `conv(a,b)`. Por exemplo, a multiplicação de $a(x) = 3x^2 - 5x + 4$ por $b(x) = 2x - 1$ é feita no MATLAB por meio de

```
>> a = [3 -5 4];
>> b = [2 -1];
>> c = conv(a,b)
c =
     6   -13    13    -4
```

resultando no polinômio $c(x) = 6x^3 - 13x^2 + 13x - 4$. Para multiplicar mais de dois polinômios a função `conv` deve ser usada repetidamente.

Divisão

O comando `[q,r]=deconv(a,b)` faz a divisão entre os polinômios cujos coeficientes são os elementos dos vetores `a` e `b`, respectivamente. O vetor `q` contém os coeficientes do polinômio quociente e o vetor `r` contém os coeficientes do polinômio obtido pelo resto da divisão, ou seja, `a=conv(q,b)+r`. Por exemplo, a divisão de $d(x) = 2x^4 - 3x^3 + 4x^2 - 5x + 6$ por $e(x) = x^2 - 3x + 1$ é efetuada por

```
>> d = [2 -3 4 -5 6];
>> e = [1 -3 1];
>> [q,r] = deconv(d,e)
q =
     2     3    11
r =
     0     0     0    25    -5
```

resultado no quociente $q(x) = 2x^2 + 3x + 11$ com resto $r(x) = 25x - 5$.

Derivação

O MATLAB oferece a função `polyder` para a derivação polinomial. Deste modo, para obter a primeira e segunda derivada de $P(x) = x^4 + 2x^3 - 12x^2 + 14x - 5$

```
>> p = [1 2 -12 14 -5];
>> p1 = polyder(p)
p1 =
     4     6   -24    14
>> p2 = polyder(p1)
p2 =
    12    12   -24
```

resultando $P'(x) = 4x^3 + 6x^2 - 24x + 14$ e $P''(x) = 12x^2 + 12x - 24$.

Construção a partir das raízes

Dadas as raízes de uma equação polinomial em um vetor `r`, a função `poly(r)` constrói o polinômio associado. Por exemplo, se $\xi_1 = -1$, $\xi_2 = 3$ e $\xi_3 = 4$ então

```
>> r = [-1 3 4];
>> p = poly(r)
p =
     1    -6     5    12
```

produz o polinômio $P(x) = x^3 - 6x^2 + 5x + 12$.

4.3.2 Cálculo de raízes

O MATLAB possui duas poderosas funções para o cálculo de raiz de equação. A função `roots` obtém todas as raízes de uma equação algébrica e a função `fzero` determina uma raiz de uma equação algébrica ou transcendente.

Função `roots`

Se o vetor linha `c` contém os coeficientes do polinômio $P(x) = c_1x^n + c_2x^{n-1} + \ldots + c_nx + c_{n+1}$ então a função `roots(c)` fornecerá um vetor coluna contendo as n raízes de $P(x) = 0$. Por exemplo, para calcular as 4 raízes de $P(x) = x^4 + 2x^3 - 13x^2 - 14x - 24 = 0$

```
>> c = [1 2 -13 -14 24];
>> r = roots(c)
r =
   -4.0000
    3.0000
   -2.0000
    1.0000
```

Pode ser mostrado que as raízes de $P(x) = 0$ são os autovalores da matriz companheira

$$C = \begin{pmatrix} -c_2/c_1 & -c_3/c_1 & \dots & -c_n/c_1 & -c_{n+1}/c_1 \\ 1 & 0 & \dots & 0 & 0 \\ 0 & 1 & \dots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \dots & 1 & 0 \end{pmatrix}.$$

Deste modo, `r = roots(c)` é equivalente à

```
>> n = length(c) - 1;
>> C = diag(ones(n-1,1),-1);
>> C(1,:) = -c(2:n+1)/c(1)
C =
    -2    13    14   -24
     1     0     0     0
     0     1     0     0
     0     0     1     0
>> r = eig(C)
r =
   -4.0000
    3.0000
   -2.0000
    1.0000
```

Função `fzero`

A função `fzero` permite o cálculo da raiz de uma equação qualquer de uma variável. Sua sintaxe é

`fzero('<função>',x0)` ou

`fzero('<função>',x0,toler)` ou

`fzero('<função>',x0,toler,exibe)`

onde `'<função>'` é o nome de um arquivo M, no qual a função é definida, `x0` é o valor inicial, `toler` é a tolerância para a raiz (se não definida então `eps` $\approx 2{,}2204 \times 10^{-16}$ é usada) e se `exibe` $\neq 0$ as iterações serão mostradas. Por exemplo, calcular uma raiz da equação $f(x) = 3x^2\text{sen}(x)e^{-x} = 0$ definida no arquivo `f.m` abaixo e cujo esboço é apresentado na figura 4.1.

```
function y = f(x)
y = 3*x^2*sin(x)*exp(-x);
```

com `x0`=2, usando tolerância padrão `eps` e sem exibir as iterações.

```
>> raiz = fzero('f',2)
Zero found in the interval: [0.72, 3.28].
raiz =
    3.1416
```

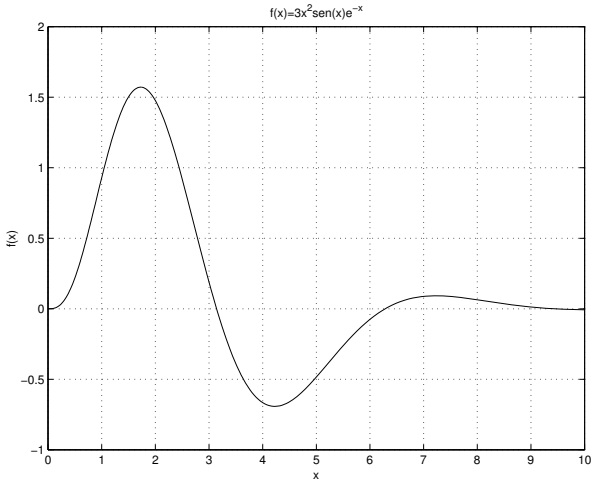


Figura 4.1: Raízes, máximo e mínimo de $f(x) = 3x^2\,\mathrm{sen}(x)e^{-x}$.

A função `fzero` é uma implementação do efetivo e eficiente algoritmo de van Wijngaarden-Dekker-Brent baseado em aproximação quadrática inversa e bisseção.

4.3.3 Determinação dos mínimos de funções

A função `fmin('<função>',a,b)` determina o mínimo de uma função de uma variável que se encontra no intervalo (`a`, `b`). A cadeia de caracteres `'<função>'` pode ser uma expressão ou o nome de um arquivo M no qual a função está definida. Para a função $f(x) = 3x^2\mathrm{sen}(x)e^{-x}$ definida no arquivo `f.m` na seção 4.3.2 existe um mínimo no intervalo (3, 5), conforme a figura 4.1. Este mínimo pode ser obtido usando o arquivo M

```
>> minimo = fmin('f',3,5)
minimo =
  4.2275
```

ou especificando diretamente a expressão

```
>> minimo = fmin('3*x^2*sin(x)*exp(-x)',3,5)
minimo =
  4.2275
```

Considerando que o máximo de $f(x)$ é igual ao mínimo de $-f(x)$, então o máximo de $f(x) = 3x^2\mathrm{sen}(x)e^{-x} \in (1, 2)$ é calculado por

```
>> maximo = fmin('-3*x^2*sin(x)*exp(-x)',1,2)
maximo =
    1.7273
```

O `help` deve ser usado para obter mais informações sobre `fmin` e a função `fmins`, usada para calcular o mínimo de função de várias variáveis.

4.4 Interpolação e ajuste de curvas

O MATLAB possui funções para cálculo das diferenças finitas ascendentes, interpolação unidimensional e bidimensional e para regressão polinomial.

4.4.1 Cálculo das diferenças finitas ascendentes

A função `diff(x)` calcula a diferença entre dois elementos adjacentes do vetor `x`. No caso de `x` ser uma matriz serão tomadas as diferenças em cada uma das colunas. Por exemplo,

```
>> x = [1 2 4 4 7]
x =
     1     2     4     4     7
>> d1 = diff(x)
d1 =
     1     2     0     3
```

Além disso, `diff(x,n)` determina as diferenças finitas de ordem `n`

```
>> d2 = diff(x,2)
d2 =
     1    -2     3
```

A função `diff` pode também ser utilizada para estimar a derivada em um ponto de uma função tabelada.

4.4.2 Interpolação unidimensional

Dada uma tabela com pares (x_i, y_i) especificados nos vetores `x` e `y`, respectivamente, então os valores de z contidos no vetor `z` podem ser interpolados usando a função `interp1`, cuja sintaxe é

```
<resultado> = interp1(x,y,z,'<método>')
```

onde a cadeia de caracteres `'<método>'` especifica o método a ser utilizado na interpolação, sendo

parâmetro	método
`'linear'`	interpolação linear.
`'cubic'`	interpolação por polinômios cúbicos.
`'spline'`	interpolação por *splines* cúbicos.
`'nearest'`	interpolação *vizinho mais próximo*.

Quando o método não for especificado será assumida uma interpolação linear. Os valores do vetor `x` devem crescer ou decrescer monotonicamente. Por exemplo, sejam cinco pontos da função $y = x^4$, para $-1 \leq x \leq 1$ definidos por

```
>> x = linspace(-1,1,5);   % abscissas dos pontos
>> y = x.^4;   % ordenadas
```

Interpolando o valor $z = 0,7$ usando os quatro métodos

```
>> interp1(x,y,0.7,'linear')
ans =
    0.4375
```

```
>> interp1(x,y,0.7,'cubic')
ans =
    0.3325
>> interp1(x,y,0.7,'spline')
ans =
    0.2695
>> interp1(x,y,0.7,'nearest')
ans =
    0.0625
```

Considerando que o valor exato é $0,7^4 \approx 0,2401$, o método de interpolação com *splines* cúbicos produziu o melhor resultado para este caso.

As aproximações da função $y = x^4$ por três funções interpoladoras de `interp1` podem ser visualizadas pelos comandos

```
x1 = linspace(-1,1,50);   % abscissas para a interpolacao
y1 = x1.^4;   % ordenadas da curva real
c = interp1(x,y,x1,'cubic');   % interpolacao cubica
s = interp1(x,y,x1,'spline');   % interpolacao por splines
plot(x,y,'o',x,y,'-.',x1,y1,'-',x1,c,'--',x1,s,':'); grid on
xlabel('x'); ylabel('y'); title(['M',setstr(233),'todos de interp1'])
legend('pontos','linear','y=x^4','cubica','splines',0)   % coloca legenda
```

A partir dos 50 pontos em `x1` foram criados os vetores `c` e `s` que contêm as ordenadas obtidas por interpolação usando polinômios cúbicos e *splines*, respectivamente. A figura 4.2 mostra graficamente os resultados da função `interp1`. Os cinco pontos iniciais estão representados por `o`. Uma interpolação linear é feita implicitamente desde que o comando `plot` interliga os pontos por retas. O gráfico da função $y = x^4$ está representado por uma linha sólida, a aproximação por polinômios cúbicos por uma linha tracejada e os *splines* por uma linha pontilhada. Usualmente os *splines* produzem uma aproximação mais suave da função como pode ser observado neste caso. Deve ser mencionado que a função `interp2` pode ser usada para se obter uma interpolação bidimensional.

4.4.3 Ajuste polinomial

O comando `c = polyfit(x,y,n)` atribui ao vetor `c` os valores dos coeficientes do polinômio de quadrados mínimos de grau `n` que melhor se ajusta aos pontos (x_i, y_i) definidos nos vetores `x` e `y`. O polinômio é na forma

$$P_n(x) = c_1 x^n + c_2 x^{n-1} + c_3 x^{n-2} + \cdots + c_n x + c_{n+1}.$$

Para exemplificar o uso de `polyfit`, sejam os dados relacionando a temperatura em graus Celsius e Farenheit obtidos a partir de um termômetro com pouca exatidão

```
>> c = [15 17 18 20 23 25 26 28 30 32];
>> f = [40 41 42 43 45 46 46 48 49 50];
```

Para obter a reta de quadrados mínimos $\hat{f} = P_1(c) = b_1 c + b_2$ basta

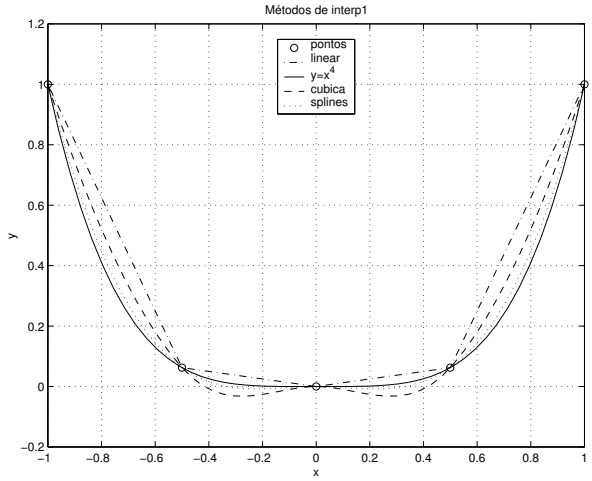


Figura 4.2: Aproximação de $y = x^4$ pela função `interp1`.

```
>> b = polyfit(c,f,1)
b =
    0.5925   31.1345
```

Apesar da pouca exatidão dos dados o resultado não foi muito diferente da relação exata que é $F = \frac{5}{9}C + 32$. Os pontos do polinômio de regressão de grau 1 avaliados para os valores de `c`, isto é, $P(c_i)$ são obtidos pelo comando

```
>> p = polyval(b,c);
```

os quais podem ser mostrados juntamente com os resíduos do ajuste $r_i = f_i - P(c_i)$

```
>> [c;f;p;f-p]'
ans =
   15.0000   40.0000   40.0226   -0.0226
   17.0000   41.0000   41.2077   -0.2077
   18.0000   42.0000   41.8003    0.1997
   20.0000   43.0000   42.9854    0.0146
   23.0000   45.0000   44.7630    0.2370
   25.0000   46.0000   45.9481    0.0519
   26.0000   46.0000   46.5406   -0.5406
   28.0000   48.0000   47.7257    0.2743
   30.0000   49.0000   48.9108    0.0892
   32.0000   50.0000   50.0959   -0.0959
```

O diagrama de dispersão com os pares (c_i, f_i) representados por `o`, bem como a reta de quadrados mínimos são produzidos pela seqüência

```
>> plot(c,f,'o',c,p); grid on
>> xlabel('Graus Celsius'); ylabel('Graus Farenheit');
>> title('Ajuste polinomial de grau 1')
```

gerando o gráfico da figura 4.3. Cumpre observar que uma regressão linear múltipla pode ser obtida usando as funções `qr` ou `svd` descritas na seção 4.2.1.

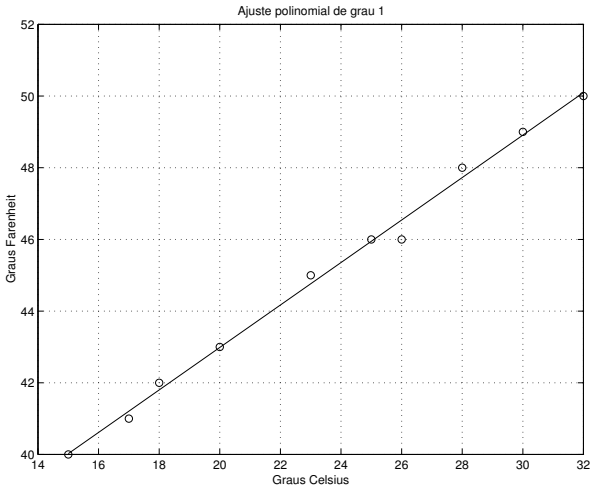


Figura 4.3: Regressão polinomial usando a função `polyfit`.

4.5 Integração numérica

O MATLAB dispõe de algumas funções para calcular numericamente a integral de uma função de uma variável.

Função `trapz`

A função `trapz(x,y)` calcula a integral de $y = f(x)$, com os pares (x_i, y_i) sendo definidos nos vetores `x` e `y`, respectivamente, usando a regra do trapézio. Se os valores x_i tiverem um espaçamento unitário então o parâmetro `x` poderá ser omitido. Por exemplo, seja a função $f(x) = \cos(3x + 1)x^2 + x^{1,5}$, cujo esboço é mostrado na figura 4.4. Para calcular

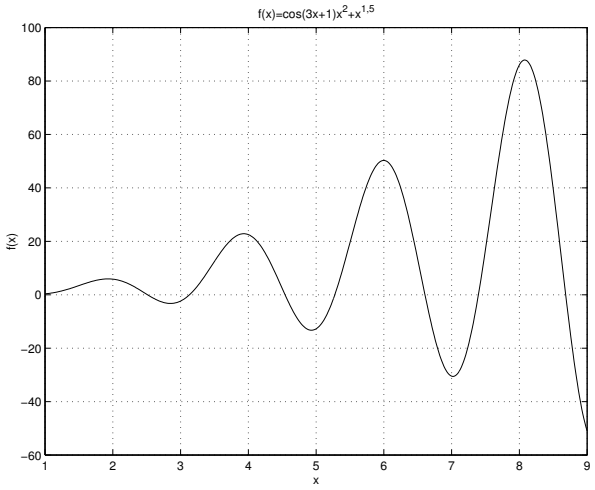


Figura 4.4: Integração de $f(x) = \cos(3x + 1)x^2 + x^{1,5}$.

$$\int_1^9 \cos(3x + 1)x^2 + x^{1,5}\, dx$$

usando `trapz` com espaçamento 0,1

```
>> x = 1:0.1:9;
>> y = cos(3*x+1).*x.^2+x.^1.5;
>> area = trapz(x,y)
area =
  102.4429
```

A função a ser integrada pode também estar definida em uma `function`

```
function y = g(x)
y = cos(3*x+1).*x.^2+x.^1.5;
```

Assim, para calcular a integral acima com espaçamento 0,01

```
>> x = 1:0.01:9;
>> y = g(x);
>> area = trapz(x,y)
area =
  102.5100
```

Considerando que o resultado exato da integral é $\approx$ 102,5106, quando se diminui o espaçamento o resultado da integração numérica torna-se mais exato.

Funções `quad` e `quad8`

Um modo mais eficiente de calcular uma integral numericamente com o MATLAB é pelas funções `quad` e `quad8`. A suas sintaxes são

```
quad('<função>',a,b,<tolerância>,<traça>)

quad8('<função>',a,b,<tolerância>,<traça>)
```

onde a cadeia de caracteres `'<função>'` é o nome de uma `function` na qual a função a ser integrada está definida, `a` e `b` são os limites inferior e superior de integração, respectivamente, `<tolerância>` especifica a tolerância para o cálculo (se não especificada então é assumido o valor 10^{-3}) e `<traça>` é um parâmetro que se tiver um valor não-nulo faz com que seja mostrado um esboço da integração. Tanto `<tolerância>` quanto `<traça>` são parâmetros opcionais. A função `quad8` produz resultados mais exatos do que `quad`. Para calcular a integral mostrada na seção anterior

```
>> area = quad('g',1,9)
area =
  102.5107
>> area = quad8('g',1,9)
area =
  102.5106
```

Uma integração mais exata pode ser obtida por

```
>> area = quad8('g',1,9,1e-10)
area =
  102.5106
```

Deve ser lembrado que as fórmulas de Simpson produzem resultados mais exatos do que a regra do trapézio quando o mesmo número de pontos são usados.