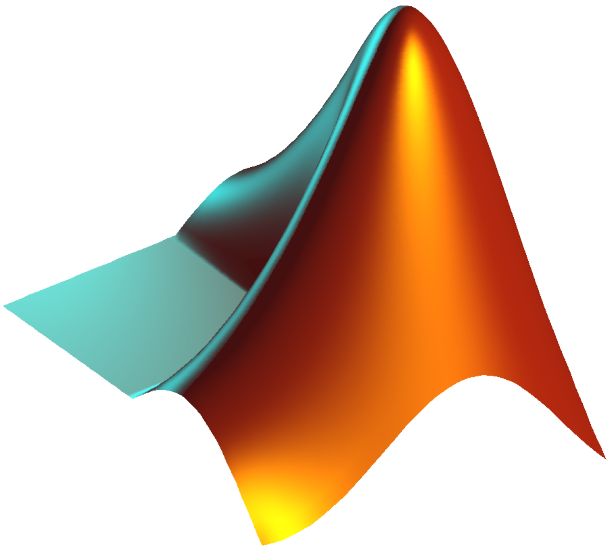
Tutorial: MATLAB



André R. Backes

Universidade Federal de Uberlândia - Faculdade de computação

Sumário

Capítulo 1

Fundamentos de MATLAB

1.1 Interface

1.2 Comandos Iniciais

1.2.1 Criando variáveis e atribuindo valores

Uma das operações mais utilizadas em programação é a operação de atribuição (=). É ela quem permite que armazenemos um valor em uma variável. No exemplo abaixo, o valor 5 é atribuído à variável x:

```
>> x = 5
x =
     5
```

Importante: no MATLAB não é necessário criar uma variável com antecedência. Sempre que atribuímos um valor a uma variável, se ela não existir, o MATLAB irá criá-la.

O nome de uma variável é um conjunto de caracteres que podem ser letras, números ou *underscores* (_). Porém, esse nome deve sempre iniciar com uma letra ou o *underscore* (_), nunca com um número. Além disso, o MATLAB é **case-sensitive**, ou seja, uma palavra escrita utilizando caracteres maiúsculos é diferente da mesma palavra escrita com caracteres minúsculos.

Importante: apesar de não termos definido, toda variável possui um **tipo**. É ele quem determina o conjunto de valores e de operações que uma variável aceita, ou seja, que ela pode executar.

Os tipos mais comuns no MATLAB são:

- **double**: valores númericos em geral. São valores reais de dupla precisão (64

bits). Podem assumir valores de 10^{-308} a 10^{308}, com 15 a 16 algarismos significativos.

```
>> x = 5
x =
     5
>> y = 2.1
y =
    2.1000
```

Note que em números reais a parte decimal usa ponto, e não vírgula.

- **char**: são valores escalares de 16 bits, representando um caractere simples.

```
>> letra = 'a'
letra =
a
>> texto = 'matlab'
texto =
matlab
```

Note que caracteres sempre ficam entre aspas simples.

Importante: se colocarmos um ponto e vírgula (;) após um comando, aquele comando será executado, mas nenhuma saída será apresentada na Janela de Comando (**Command window**).

```
>> x = 4;
>> y = 3.2
y =
    3.2000
```

1.2.2 A variável ans

Vimos que uma variável é criada sempre que atribuímos um valor a ela. Porém, podemos querer calcular uma operação, saber o seu resultado, mas não armazená-la. Sempre que realizamos uma operação mas não definimos a variável onde o resultado será armazenado, o MATLAB salva o resultado na variável **ans**, como mostra o exemplo a seguir:

```
>> 2 + 3
ans =
     5

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

1.2.3 Trabalhando com matrizes

1.2.3.1 Definindo uma matriz usando uma lista de valores

Podemos definir uma matriz utilizando uma lista explícita de valores delimitada por colchetes ([ ]). Na matriz, os elementos de uma mesma linha são separados por espaços ou vírgulas (,). Duas linhas são separadas utilizando um ponto e vírgula (;).

Exemplo de matriz 3x3

```
>> A = [16 3 13; 5 11 8; 9 6 7; 4 14 1]
A =
    16     3    13
     5    11     8
     9     6     7
     4    14     1
```

Exemplo de matriz 1x5

```
>> B = [16 3 2 13 1]
B =
    16     3     2    13     1
```

Exemplo de matriz 2x1

```
>> C = [2; 1]
C =
     2
     1
```

1.2.3.2 Funções para a criação de uma matriz

O MATLAB possui várias rotinas que permitem a criação automática de uma matriz com valores pré-definidos. Abaixo são apresentadas algumas dessas funções:

zeros(M,N,P,...): cria uma matriz preenchida com 0's. As dimensões da matriz são definidas pela quantidade de parâmetros informados. Exemplos:

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

```
     0     0
>> m = zeros(2,1)
m =
     0
     0
```

ones(M,N,P,...): cria uma matriz preenchida com 1's. As dimensões da matriz são definidas pela quantidade de parâmetros informados. Exemplos:

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

rand(M,N,P,...): cria uma matriz preenchida com números aleatórios seguindo uma distribuição **uniforme**. As dimensões da matriz são definidas pela quantidade de parâmetros informados. Exemplos:

```
>> m = rand(1,4)
m =
    0.9572    0.4854    0.8003    0.1419
>> m = rand(2,4)
m =
    0.4218    0.7922    0.6557    0.8491
    0.9157    0.9595    0.0357    0.9340
>> m = rand(3,1)
m =
    0.6787
    0.7577
    0.7431
```

randn(M,N,P,...): cria uma matriz preenchida com números aleatórios seguindo uma distribuição **normal**. As dimensões da matriz são definidas pela quantidade de parâmetros informados. Exemplos:

```
>> m = randn(1,5)
m =
   -0.3034    0.2939   -0.7873    0.8884   -1.1471
```

```
>> m = randn(2,5)
m =
   -1.0689   -2.9443    0.3252    1.3703   -0.1022
   -0.8095    1.4384   -0.7549   -1.7115   -0.2414
>> m = randn(2)
m =
    0.3192   -0.8649
    0.3129   -0.0301
```

eye(N): cria uma matriz identidade de ordem N. Exemplos:

```
>> m = eye(2)
m =
     1     0
     0     1
>> m = eye(3)
m =
     1     0     0
     0     1     0
     0     0     1
```

1.2.3.3 Acessando os elementos da matriz

Para acessar o valor de um determinado elemento da matriz, devemos indicar qual índice dela queremos acessar. Isso é feito utilizando o operador de **parênteses** () após o nome da matriz. Dentro dos parênteses, devemos indicar um número inteiro para cada dimensão da matriz e separá-los por vírgulas (,). Exemplos:

Acessando elementos de uma matriz 2x4:

```
>> m = rand(2,4);
>> m(1,1)
ans =
    0.4456
>> m(2,3)
ans =
    0.6797
```

Acessando elementos de uma matriz 2x3x4:

```
>> m = rand(2,3,4);
>> m(1,1,2)
ans =
    0.2785
>> m(2,3,1)
ans =
    0.0975
```

Importante: os índices da matriz devem ser sempre valores inteiros e positivos. E a primeira posição da matriz é a posição de índice 1.

1.2.3.4 Concatenação de matrizes

A operação de concatenação consiste em agrupar diferentes variáveis e/ou matrizes em uma única matriz. A concatenação é feita da mesma maneira como é feita a definição de uma matriz: por meio de uma lista explícita de valores delimitada por colchetes ([ ]). As variáveis e/ou matrizes que iram ocupar uma mesma linha são separados por espaços ou vírgulas (,). Duas linhas são separadas utilizando um ponto e vírgula (;).

Para entender esse processo, considere duas matrizes A e B:

```
>> A = [1 2]
A =
     1     2

>> B = [3 4]
B =
     3     4
```

Podemos concatenar A e B em uma mesma linha (nesse caso, todas as matrizes concatenadas devem ter o mesmo número de linhas):

```
>> C = [A B]
C =
     1     2     3     4
```

Também podemos concatenar A e B em linhas diferentes (nesse caso, todas as matrizes concatenadas devem ter o mesmo número de colunas):

```
>> C = [A; B]
C =
     1     2
     3     4
```

Podemos ainda utilizar rotinas prontas para a criação de matrizes, assim como o aninhamento de concatenações

```
>> C = [[A; B] zeros(2,2)]
C =
     1     2     0     0
     3     4     0     0
```

1.2.3.5 O operador dois pontos (:)

Os dois pontos (:) é um operador importante no MATLAB. Por meio dele é possível fazer a enumeração intervalada de dados, ou seja, ele permite que selecionemos um intervalo de valores com um determinado espaçamento.

Esse operador pode ser utilizado de duas maneiras:

- **valor_inicial:incremento:valor_final** - cria uma matriz com os elemento de **valor_inicial** a **valor_final**, espaçados de **incremento**. Se **incremento** for positivo, **valor_final** deverá ser maior do que **valor_inicial**. Se **incremento** for negativo, **valor_final** deverá ser menor do que **valor_inicial**.
- **valor_inicial:valor_final** - cria uma matriz com os elementos de **valor_inicial** a **valor_final** usando **incremento** igual a 1. Nesse caso, **valor_final** deverá ser maior do que **valor_inicial**.

Abaixo podemos ver alguns exemplos:

```
>> v = 1:5
v =
     1     2     3     4     5
>> v = 5:-1:-5
v =
     5     4     3     2     1     0    -1    -2    -3    -4    -5
>> v = 0:pi/4:3
v =
         0    0.7854    1.5708    2.3562
```

1.2.3.6 Submatrizes e o comando end

No MATLAB, podemos extrair uma submatriz de outras matrizes maiores. Esse procedimento é bastante simples: basta criar um vetor (matriz linha ou coluna) com os índices a serem selecionados. Esse vetor pode ser criado utilizando uma lista de valores ou o operador dois pontos(:).

```
>> A = [0 2 4 6 8 10 12 14 16];
>> B = A([1 2 3])
B =
     0     2     4

>> B = A(4:6)
B =
     6     8    10

>> B = A(4:end)
```

```
B =
     6     8    10    12    14    16
```

Perceba, no último exemplo, que o comando **end** indica o último índice daquela dimensão da matriz. No caso de nossa matriz possuir mais de uma dimensão, devemos informar os índices a serem selecionados para cada dimensão da matriz.

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

O operador dois pontos (:) também pode ser utilizado para indicar que todos os elementos de uma linha (ou coluna) da matriz serão selecionados. Por exemplo, o comando abaixo seleciona todos os elementos da segunda linha:

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

1.2.3.7 Excluindo linhas ou colunas da matriz

A exclusão de linhas ou colunas segue a idéia de seleção de submatrizes. Uma vez definida a submatriz, atribuímos a ela uma matriz valia,

. O exemplo abaixo mostra como excluir elementos de um vetor:

```
>> A = [0 2 4 6 8 10 12 14 16];
>> A(3:5) = []
A =
     0     2    10    12    14    16
```

Podemos também excluir linhas (ou colunas) inteiras de uma matriz:

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

1.2.3.8 Calculando a transposta e outras propriedades

Para calcular a transposta de uma matriz, basta acrescentar o caracter apóstrofo, "'", após o nome da matriz, como mostra o exemplo a seguir:

```
>> X = [1 2]
X =
     1     2

>> Y = X'
Y =
     1
     2
```

O MATLAB também possui várias outras funções que permitem calcular diversas propriedades de uma matriz. Abaixo são apresentadas algumas dessas funções:

- **diag(X)**: retorna os elementos da diagonal da matriz;
- **inv(X)**: retorna a inversa de uma matriz;
- **dot(V1,V2)**: produto escalar de V1 por V2;
- **cross(V1,V2)**: produto vetorial de V1 por V2;
- **trace(M)**: traço da matriz M (soma dos elementos na diagonal principal);
- **det(M)**: determinante da matriz M;
- **[A,B] = eig(M)**: retorna em A os auto-vetores e em B os autovalores de M;

1.2.4 Trabalhando com textos

Para o MATLAB, um texto é tratado como sendo um matriz contendo uma única linha preenchida com caracteres. Por esse motivo, a maioria dos comandos vistos até agora, como acesso a elementos e seleção de submatrizes, são aplicáveis no caso dos textos.

Exemplo: acessando e substituindo um elemento

```
>> palavra = 'teste';
>> palavra(1) = 'l'
palavra =
leste
```

Exemplo: selecionando um trecho do texto

```
>> palavra = 'matlab';
>> palavra(4:end)
ans =
lab
```

Apesar de o MATLAB ser um software voltado para cálculos matemáticos, ele possui uma série de rotinas para a manipulação de textos. Essas rotinas nos ajudam em tarefas de comparação de textos, localização de subtextos, entre outras tarefas. Abaixo podemos ver algumas dessas rotinas.

- **ind = findstr(str1,str2)**: retorna uma matriz com os índices onde se iniciam todas as ocorrências do texto menor dentro do texto maior.

  ```
  >> texto = 'O espaço separa uma palavra de outra palavra';
  >> ind = findstr(texto,'palavra')
  ind =
      21    38
  ```

- **res = strcmp(str1,str2)**: compara dois textos. A função retorna o valor 1 se forem iguais, e 0 se forem diferentes. É case-sensitive, ou seja, o tamanho das letras afeta o resultado.

  ```
  >> palavra = 'matlab';
  >> res = strcmp(palavra,'software')
  res =
       0
  ```

- **res = strcmpi(str1,str2)**: compara dois textos. A função retorna o valor 1 se forem iguais, e 0 se forem diferentes. Não é case-sensitive, ou seja, o tamanho das letras não afeta o resultado.

  ```
  >> palavra = 'matlab';
  >> res = strcmpi(palavra,'MATLAB')
  res =
       1
  ```

- **[ini,fim] = strtok(texto,caractere)**: separa um texto em duas partes usando como separador um caractere. A função retorna na variável **ini** todo o texto encontrado antes do caractere e em **fim** o restante do texto.

  ```
  >> texto = 'O espaço separa uma palavra de outra palavra';
  >> [ini,fim] = strtok(texto,'a')
  ini =
  O esp

  fim =
  aço separa uma palavra de outra palavra
  ```

- **s = strcat(str1,str2,...)**: concatena dois ou mais textos e armazena na variável s.

```
>> s1 = 'bom';
>> s2 = 'dia';
>> s = strcat(s1,'-',s2)
s =
bom-dia
```

- **s = sprintf(formato,A,...)**: escreve na variável **s** um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o **formato** especificado.

```
>> x = 1.5;
>> y = 2;
>> s = sprintf('Nro real: %f \nNro inteiro: %d',x,y)
s =
Nro real: 1.500000
Nro inteiro: 2
```

- **A = sscanf(texto,formato)**: lê da variável **texto** um conjunto de valores, caracteres e/ou sequência de caracteres de acordo com o **formato** especificado e salva na variável **A**.

```
>> S = '2.7183  3.1416';
>> A = sscanf(S,'%f')
A =
    2.7183
    3.1416
```

As funções **sprintf()** e **sscanf()** necessitam de um parâmetro **formato** para funcionarem corretamente. Esse parâmetro informa como o dado deve ser transformado durante a leitura ou escrita. Entre os possíveis formatos, estão:

%c	caractere
%d ou %i	números inteiros
%f	número reais
%s	texto (vários caracteres)

Além disso, essas funções também suportam caracteres especiais como nova linha (\n), retorno de carro (\r), tabulação horizontal (\t), retrocesso (\b), alimentação de folha (\f) e barra invertida (\\).

1.2.5 Convertendo valores para outros tipos

O MATLAB possui também uma série de funções para a conversão de valores de um tipo para outro tipo. Abaixo podemos ver algumas dessas rotinas.

- **y = double(x)**: converte um valor **x** para a precisão **double**.
- **y = uint8(x)**: converte um valor **x** para inteiro 8 bits sem sinal.
- **y = int8(x)**: converte um valor **x** para inteiro 8 bits com sinal.
- **y = uint32(x)**: converte um valor **x** para inteiro 32 bits sem sinal.
- **y = int32(x)**: converte um valor **x** para inteiro 32 bits com sinal.
- **y = uint8(x)**: converte um valor **x** para inteiro 8 bits sem sinal.
- **y = num2str(x)**: converte um valor **x** para o formato texto.
- **y = str2num(x)**: converte um texto **x** para o seu valor numérico.
- **y = str2double(x)**: converte um texto **x** para o seu valor numérico com a precisão **double**.

1.2.6 Funções diversas

- **log(x)**: logaritmo natural de x;
- **log10(x)**: logaritmo base 10 de x;
- **log2(x)**: logaritmo base 2 de x;
- **exp(x)**: exponencial de x;
- **sqrt(x)**: raiz quadrada de x;
- **pow2(x)**: 2 elevado a potência x;
- **sin(x)**: seno de x (x em radianos);
- **asin(x)** : seno inverso de x;
- **cos(x)**: coseno de x (x em radianos);
- **acos(x)**: cosseno inverso de x;
- **tan(x)**: tangente de x (x em radianos);
- **atan(x)**: tangente inverso de x;
- **sec(x)**: secante de x (x em radianos);
- **asec(x)**: secante inverso de x;
- **csc(x)**: cosecante de x (x em radianos);
- **acsc(x)**: cosecante inverso de x;
- **ceil(x)**: Arredonda x na direção $+\infty$;

- **round(x)**: arredonda x para o inteiro mais próximo;
- **floor(x)**: arredonda x na direção $-\infty$;
- **fix(x)**: arredonda x na direção de 0;
- **norm(X)**: norma da matriz X;
- **min(X)**: menor elemento da matriz X;
- **max(X)**: maior elemento da matriz X;
- **sort(X)**: organiza os elementos da matriz X em ordem crescente;
- **mean(X)**: média de um vetor;
- **std(X)**: desvio padrão de um vetor;
- **size(X)**: retorna as dimensões da matriz;
- **i**: número imaginário, igual a $\sqrt{-1}$;
- **j**: número imaginário, igual a $\sqrt{-1}$;
- **real(C)**: retorna a parte real de um número complexo C;
- **imag(C)**: retorna a parte imaginária de um número complexo C;
- **conj(C)**: retorna o conjugado de um número complexo C;
- **angle(C)**: retorna a fase de um número complexo C;
- **abs(C)**: retorna o módulo de um número complexo C;
- **find(condição da matriz)**: retorna os índices da matriz que concordem com uma determinada condição;
- **disp(texto)**: exibe um texto na Command Window;
- **mod(x,y)**: retorna o resto da divisão inteira x/y;

1.3 Operadores

1.3.1 Operadores aritméticos

Os operadores aritméticos são aqueles que operam sobre números (**valores**, **variáveis** ou **matrizes**) e/ou expressões e tem como resultado valores numéricos. O MATLAB possui um total de seis operadores aritméticos, como mostra a Tabela 1.1.

Importante: caso os valores operados sejam duas matrizes, suas dimensões deverão satisfazer as regras da operação escolhida. POr exemplo, soma e subtração exigem

Operador	Significado	Exemplo
+	adição de dois valores	z = x + y
-	subtração de dois valores	z = x - y
*	multiplicação de dois valores	z = x * y
/	quociente de dois valores (a direita)	z = x / y
\	quociente de dois valores (a esquerda)	z = x \ y
∧	exponenciação	z = x ∧ y

Tabela 1.1:

matrizes de tamanhos iguais. A multiplicação, que o número de colunas da primera seja igual ao de linhas da segunda.

Abaixo são apresentados alguns exemplos desses operadores:

```
>> x = 5;
>> y = x /2
y =
    2.5000

>> A = [1 2 3; 4 5 6];
>> B = A + 1
B =
     2     3     4
     5     6     7

>> C = B - A
C =
     1     1     1
     1     1     1
```

Perceba que temos duas operações de divisão: / (divisão a direita) e \ (divisão a esquerda). A primeira é a divisão tradicional, onde o elemento da esquerda é dividido pelo elemento da direita. Já a segunda, \, o elemento da direita é dividido pelo elemento da esquerda, ou seja, **(x \ y)** equivale a **(y / x)**

```
>> 4 / 2
ans =
     2

>> 4 \ 2
ans =
    0.5000

>> 2 / 4
ans =
    0.5000
```

1.3.1.1 Operadores aritméticos para conjuntos

O MATLAB permite forçar que uma operação aritmética seja executada para todos os elementos de uma matriz. Isso é bastante útil em operações como a multiplicação, onde a operação ocorre de forma diferenciada para matrizes. Com o operador de conjuntos, podemos fazer a multiplicação de um elemento com o elemento localizado na mesma posição da outra matriz. A única restrição é de que as matrizes tenham as mesmas dimensões.

Esses operadores são os mesmos operadores aritméticos já conhecidos, porém precedidos por um ponto, como mostra a Tabela 1.2.

Operador	Significado	Exemplo
.+	adição de dois valores	z = x + y
.-	subtração de dois valores	z = x - y
.*	multiplicação de dois valores	z = x * y
./	quociente de dois valores (a direita)	z = x / y
.\	quociente de dois valores (a esquerda)	z = x \ y
.^	exponenciação	z = x ^ y

Tabela 1.2:

Abaixo são apresentados alguns exemplos desses operadores:

```
>> A = [1 2; 4 5];
>> B = [3 1; 6 8];
>> C = A ./ B
C =
    0.3333    2.0000
    0.6667    0.6250

>> C = A .* B
C =
     3     2
    24    40
```

1.3.2 Operadores relacionais

Os operadores relacionais são aqueles que operam sobre dois valores (variáveis ou matrizes) e/ou expressões e verificam a magnitude (quem é maior ou menor) e/ou igualdade entre eles. São operadores que servem para comparar dois elementos no MATLAB.

O MATLAB possui um total de seis operadores relacionais, como mostra a Tabela 1.3. Como resultado, esse tipo de operador retorna:

- o valor **UM** (1), se a expressão relacional for considerada **verdadeira**;

- o valor **ZERO** (0), se a expressão relacional for considerada **falsa**.

Operador	Significado	Exemplo
>	Maior do que	x > 5
>=	Maior ou igual a	x >= 10
<	Menor do que	x < 5
<=	Menor ou igual a	x <= 10
==	Igual a	x == 0
=	Diferente de	x = 0

Tabela 1.3:

Importante: caso os valores comparados sejam duas matrizes, elas deverão ter as mesmas dimensões. Nesse caso, a comparação será elemento a elemento.

Abaixo são apresentados alguns exemplos desses operadores:

```
>> x = 5;
>> x > 0
ans =
     1

>> A = [1 2;3 4];
>> B = [1 3;3 5];
>> C = A == B
C =
     1     0
     1     0
```

1.3.3 Operadores lógicos

Os operadores lógicos são aqueles que permitem representar situações lógicas, unindo duas ou mais expressões relacionais simples em uma composta. O MATLAB possui um total de três operadores lógicos, como mostra a Tabela 1.4.

Operador	Significado	Exemplo
&&	Operador **E**	(x >= 0 && x <= 9)
||	Operador **OU**	(a == 'F' ||b = 32)
	Operador **NEGAÇÃO**	(x == 10)

Tabela 1.4:

Esses operadores atuam apenas sobre valores lógicos produzidos por duas ou mais expressões relacionais. Como resultado, eles retornam também um valor lógico de acordo com o operador utilizado:

- Operador **E** (&&): a expressão resultante somente será verdadeira se **ambas** as expressões unidas por esse operador também forem;

- Operador **OU** (||): a expressão resultante é verdadeira se **alguma** das expressões unidas por esse operador também for;
- Operador **NEGAÇÃO** ( ): inverte o valor lógico da expressão na qual se aplica.

Abaixo são apresentados alguns exemplos desses operadores:

```
>> x = 5;
>> y = 3;
>> r = (x > 2) && (y < x)
r =
     1

>> r = (x > 2) || (y > x)
r =
     1

>> r = ~(x > 2)
r =
     0
```

1.4 Arquivos .m

1.4.1 Definição

De modo geral, os comandos do MATLAB são digitados na Janela de Comando (**Command Window**). Nela, cada linha de comando digitada é processada imediatamente. Porém, muitas vezes é necessário definir e executar um sequência de comandos. Felizmente, o MATLAB permite criar arquivos de **scripts** chamados arquivos ".m"devido a sua extensão.

Um arquivo ".m"é um arquivo escrito no formato texto (ASCII) contendo uma sequências de comandos do MATLAB. Esse arquivo pode conter qualquer comando do MATLAB ou função definida pelo usuário. Ele também pode ser utilizado para criar novas funções, como veremos na seção seguinte.

Importante: em um arquivo ".m"todas as variáveis são globais. Ou seja, a execução desse arquivo altera os valores das variáveis que já existem no **Workspace** e possuem mesmo nome.

Abaixo podemos ver um exemplo de arquivo ".m":

```
clear all;
close all;
x= 1:pi/16:8*pi;
plot(x,sin(x));
```

```
title('gráfico seno');
xlabel('ângulo');
ylabel('seno');
```

1.4.2 Criando suas próprias funções

Quando criamos um arquivo ".m", podemos definir se este será apenas um arquivo de script de comandos ou se ele será uma função. Uma função nada mais é do que um bloco de comandos (ou seja, declarações e outros comandos) que pode ser nomeado e chamado de dentro de um script ou pela janela de comando. Trata-se de uma ferramenta bastante útil para a estruturação dos programas e reutilização de código.

Para criar uma função, é preciso que a primeira linha do arquivo siga a seguinte forma geral

function [var1,var2,...,varN] = nome_função(param1,param2,...,paramM);

em que **nome_função** é o nome da função, **var1,var2,...,varN** são os valores retornados pela função e **param1,param2,...,paramM** são os valores passados para dentro da função. Pode-se definir para um função qualquer quantidade de parâmetros de entrada. O mesmo vale para os valores de saída. Além disso, é aconselhavel que se utilize o mesmo nome da função para o arquivo ".m".

Importante: em uma função as variáveis são sempre locais. Porém, é possível declarar uma variável como global. Mais detalhes em **help global**.

Abaixo podemos ver um exemplo de função que recebe um vetor (matriz linha ou coluna) como parâmetro e retorna a sua média

```
function me = media(vetor)
me = 0;
n = length(vetor);
for y=1:n
    me = me + vetor(y);
end
me = me/n;
end
```

Um arquivo de função pode ter mais de uma função. A função primária é aquela que leva o mesmo nome do arquivo e deve ser a primeira. As demais, chamadas de subfunções, devem aparecer após a primeira, em qualquer ordem. Além disso, subfunções somente podem ser chamadas pela função primária ou por outras subfunções deste arquivo. Abaixo podemos ver um exemplo de arquivo ".m"com função e uma subfunção:

```
function de = desvio_padrao(vetor)%função primária
me = media(vetor);
de = 0;
n = length(vetor);
for y=1:n
    de = de + (vetor(y)-me)^2;
end
de = sqrt(de/(n-1));
end

function me = media(vetor)%subfunção
me = 0;
n = length(vetor);
for y=1:n
    me = me + vetor(y);
end
me = me/n;
end
```

1.4.3 Comentários

Um comentário, como o próprio nome diz, é um trecho de texto incluído dentro do arquivo ".m"para descrever alguma coisa, por exemplo, o que aquele pedaço do script faz. Os comentários não modificam o funcionamento do script porque são ignorados pelo MATLAB e servem, portanto, apenas para ajudar o programador a organizar o seu código.

Um comentário pode ser adicionado em qualquer parte do código. Para tanto, basta colocar um símbolo de % na frente da linha que será o comentário, como mostra o exemplo a seguir:

```
%apagando as variáveis existentes
clear all;
%fechando as janelas existentes
close all;
%cria o eixo x
x= 1:pi/16:8*pi;
%desenha o gráfico
plot(x,sin(x));
%altera as propriedades do gráfico
title('gráfico seno');
xlabel('ângulo');
ylabel('seno');
```

1.5 Comandos de condição e repetição

Os comandos de condição e repetição controlam o fluxo e especificam a ordem em que a computação é feita dentro de um script MATLAB. Esses comando funcionam de forma semelhante aos usados na linguagem C, mas com uma estrutura um pouco diferente, como veremos a seguir.

1.5.1 Comando if-else

O comando **if** avalia uma expressão condicional e executa uma sequência de comandos se esta condição for verdadeira. Por expressão condicional se entende qualquer expressão que resulte em uma resposta do tipo **verdadeiro** ou **falso** e construída utilizando operadores matemáticos, relacionais ou lógicos. A forma geral de um comando **if** é:

```
if expressão_condicional
    sequência de comandos
end
```

Abaixo podemos ver um exemplo do comando **if**:

```
if x > 0
    y = 10;
end
```

Já o comando **if-else** avalia uma expressão condicional considerando a possibilidade de ela ser falsa. Se o comando **if** diz o que fazer quando a condição é verdadeira, o comando **else** permite executar uma sequência de comandos quando a condição é falsa. A forma geral de um comando **if-else** é:

```
if expressão_condicional
    sequência de comandos do if
else
    sequência de comandos do else
end
```

Abaixo podemos ver um exemplo do comando **if-else**:

```
if x > 0
    y = 10;
else
    y = -10;
end
```

1.5.2 Comando switch

Parecido com o comando if-else, este comando permite que se escolha uma opção entre várias dependendo do resultado de uma variável ou expressão.

O comando switch é um comando de seleção múltipla e é indicado quando se deseja testar uma variável (ou expressão) em relação a diversos valores (ou listas de valores) pré-estabelecidos. A forma geral de um comando **switch** é:

```
switch variável
    case valor1
        sequência de comandos
    case {valor2, valor3, ...}
        sequência de comandos
    ...
    otherwise
        sequência de comandos
end
```

Na execução do comando **switch**, o valor da *variável* é comparado, na ordem, com cada um dos valores definidos pelo comando **case**. Se um desse valores for igual ao valor da variável, a sequência de comandos daquele comando **case** é executado pelo programa. O comando *otherwise* é opcional e sua sequência de comandos somente será executada se o valor da variável que está sendo testada pelo comando **switch** não for igual a nenhum dos valores dos comandos **case**.

Abaixo podemos ver um exemplo do comando **switch**:

```
switch A(2,2)
    case 0
        A(2,2) = A(2,2) + 1;
    case 1
        A(2,2) = A(2,2) + 3;
    case 2
        A(2,2) = A(2,2) + 5;
    otherwise
        A(2,2) = 0;
end
```

1.5.3 Comando while

O comando **while** é um comando de repetição que funciona de forma parecida com o comando **if**. Como o **if**, esse comando avalia uma expressão condicional e executa uma sequência de comandos enquanto essa condição for verdadeira. Ou seja, ao final da sequência de comandos, o **while** testa novamente a expressão condicional para

saber se deve ou não executar novamente aquela sequência de comandos. Por expressão condicional se entende qualquer expressão que resulte em uma resposta do tipo **verdadeiro** ou **falso** e construída utilizando operadores matemáticos, relacionais ou lógicos. A forma geral de um comando **while** é:

```
while expressão_condicional
    sequência de comandos
end
```

Abaixo podemos ver um exemplo do comando **while**:

```
v = zeros(1,10);
x = 1;
while x <= 10
    v(x) = x^2;
end
```

```
for varíavel = valor_inicial:incremento:valor_final
    sequência de comandos
end
```

1.5.4 Comando for

O comando **for** é o controlador de fluxo mais simples e usado na programação com MATLAB. Basicamente, esse comando é usado para repetir uma sequência de comandos diversas vezes. A forma geral de um comando **for** é

```
for varíavel = valor_inicial:incremento:valor_final
    sequência de comandos
end
```

O **incremento** é opcional. Se o seu valor não for definido, ele assumirá o valor de 1. Abaixo podemos ver um exemplo do comando **if-else**:

```
v = zeros(1,10);
for x=1:10
    v(x) = x^2;
end
```

1.5.5 Comando continue e break

Esses dois comandos servem para quebrar a continuidade dos comandos de repetição, como o **for** e o **while**. O funcionamento deles é idêntico ao dos comandos correspondentes na linguagem C:

- **continue** interrompe a execução atual do laço e faz com que ele avance para a sua próxima iteração.

- **break**: interrompe a execução do laço e o termina, fazendo com que o MATLAB siga para a primeira instrução fora do laço.

1.6 Desenhando gráficos

1.6.1 Gráfico de linhas: plot

Para criar um gráfico de linha no MATLAB, utilizamos a função **plot**. Essa função pode ser utilizada de várias maneiras diferentes. Apresentamos aqui três delas:

- **plot(Y)**: cria um gráfico dos elementos de **Y** versos os seus índices;

- **plot(X,Y)**: cria um gráfico dos elementos de **X** versos os elementos de **Y**. **X** e **Y** são vetores com as mesmas dimensões;

- **plot(X,Y,str)**: cria um gráfico dos elementos de **X** versos os elementos de **Y**. **X** e **Y** são vetores com as mesmas dimensões. O texto **str** define a combinação de estilos escolhida pelo usuário para desenhar o gráfico.

Ao definir o estilo de um gráfico, o usuário pode escolher até três elementos: estilo de cor, de marcador e de linha. A Tabela 1.5 apresenta os estilos possíveis.

Cores		Marcadores		Linhas	
símbolo	descrição	símbolo	descrição	símbolo	descrição
y	amarelo	.	ponto	-	linha contínua
m	lilás	*	asterisco	–	linha tracejada
c	azul claro	o	círculo	-.	traços e pontos
r	vermelho	+	+	:	linha pontilhada
g	verde	x	x		
b	azul escuro	s	quadrado		
w	branco	d	losango		
k	preto	v	triângulo p/ baixo		
		∧	triângulo p/ cima		
		<	triângulo p/ esquerda		
		>	triângulo p/ direita		
		p	pentagrama		
		h	hexagrama		

Tabela 1.5:

Para exemplificar o uso dessa função, iremos utilizar do seguinte conjunto de comandos:

```
x = 1:pi/16:8*pi;
y = sin(x);
```

As janelas produzidas para diferentes usos da função são mostradas na Figura 1.1.

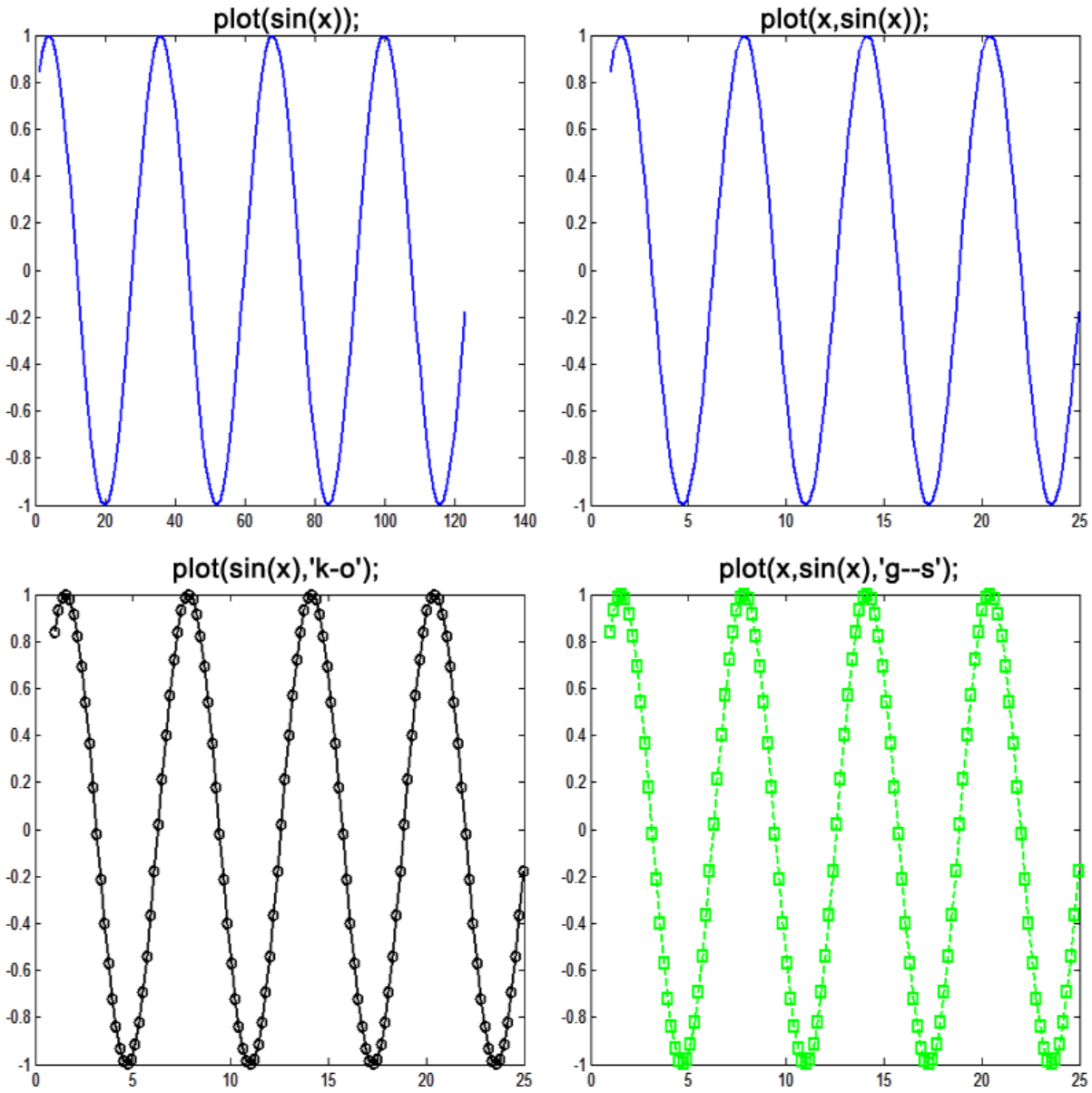


Figura 1.1:

1.6.2 Gráfico de barras: bar

Para criar um gráfico de barras no MATLAB, utilizamos a função **bar**. Essa função pode ser utilizada de várias maneiras diferentes. Apresentamos aqui três delas:

- **bar(Y)**: cria um gráfico de barras dos elementos de **Y** versos os seus índices;

- **bar(X,Y)**: cria um gráfico de barras dos elementos de **X** versos os elementos de **Y**. **X** e **Y** são vetores com as mesmas dimensões;
- **bar(X,Y,largura)**: cria um gráfico de barras dos elementos de **X** versos os elementos de **Y**. **X** e **Y** são vetores com as mesmas dimensões. O valor **largura** define a largura de cada barra. Valores maiores do que 1 causam sobreposição das barras.

A Figura 1.2 mostra dois exemplos de gráficos de barras.

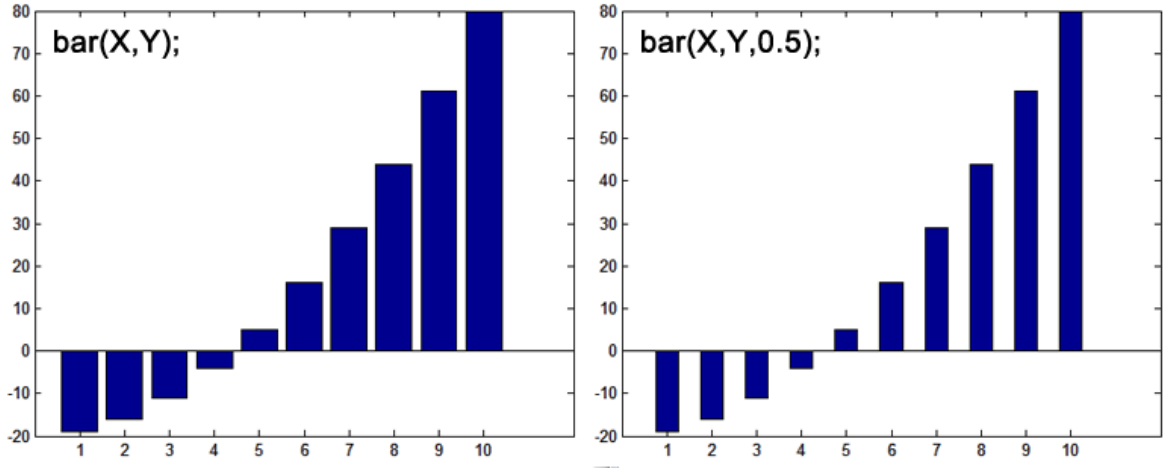


Figura 1.2:

1.6.3 Desenhando superfícies

O MATLAB possui várias funções para desenhar superfícies. Entre essas funções podemos destacar as seguintes: **mesh**, **surf**, **contour** e **contour3**. De modo geral, essas funções permitem desenhar uma superfície que esteja contida em uma matriz. Todas as quatro funções (**mesh**, **surf**, **contour** e **contour3**) podem ser utilizadas de duas formas básicas:

- **nome_função(Z)**: a superfície contida em **Z** será desenhada considerando os seus índices como os eixos **X** e **Y**;
- **nome_função(X,Y,Z)**: a superfície contida em **Z** será desenhada considerando os eixos definidos em **X** e **Y**.

Para exemplificar o uso dessas funções, iremos utilizar do seguinte conjunto de comandos:

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

As janelas produzidas para cada uma das funções são mostradas na Figura 1.3. A função **meshgrid** define um espaço de coordenadas para ser trabalhado. Mais detalhes em **help meshgrid**.

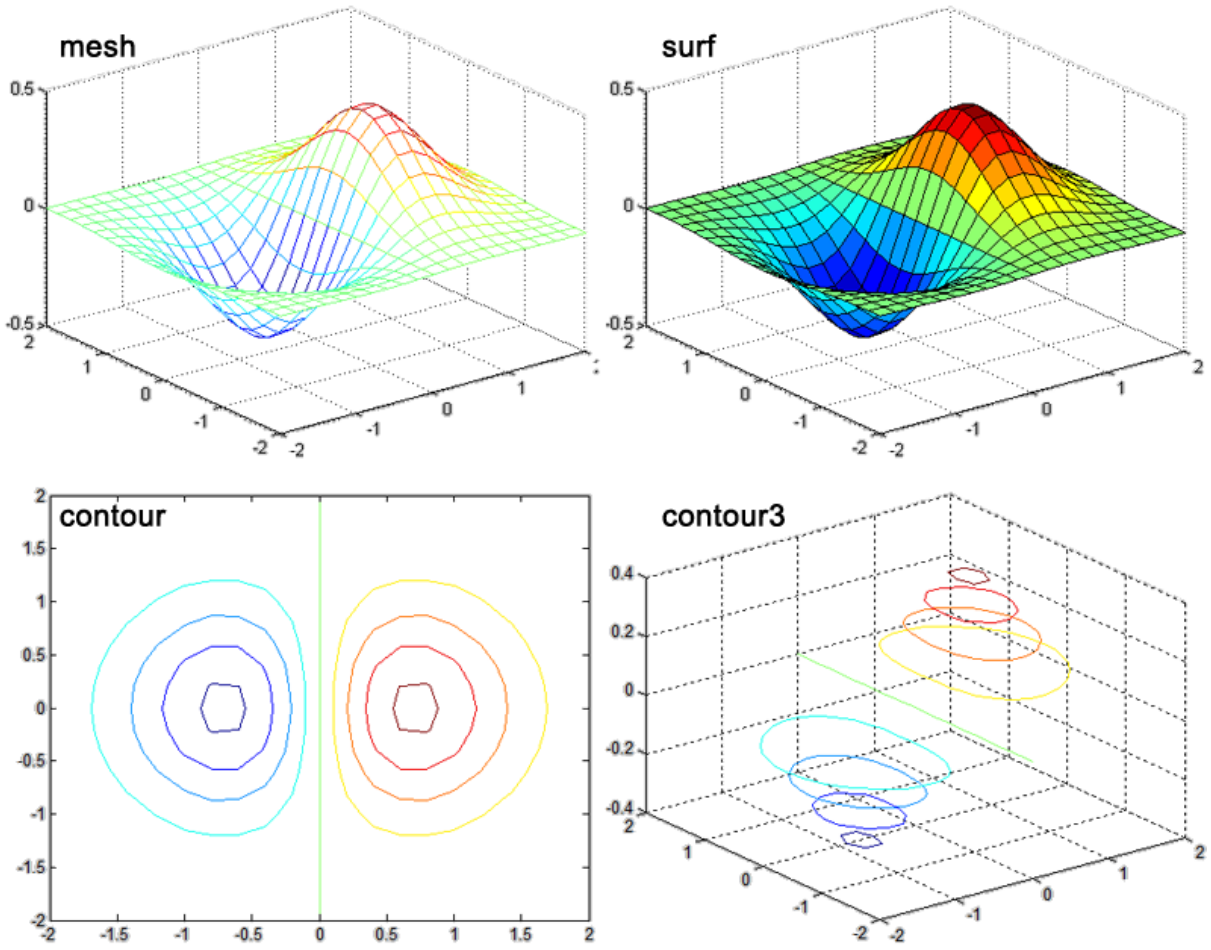


Figura 1.3:

1.6.4 Inserindo anotações em um gráfico

O MATLAB possui várias funções para fazer anotações em um gráfico. Abaixo podemos ver algumas delas:

- **title(str)**: coloca o texto armazenado em **str** como o título acima do gráfico.
- **xlabel(str)**: coloca o texto armazenado em **str** como o título do eixo X.
- **ylabel(str)**: coloca o texto armazenado em **str** como o título do eixo Y.
- **zlabel(str)**: coloca o texto armazenado em **str** como o título do eixo Z.
- **text(X,Y,str)**: coloca o texto armazenado em **str** na posição (X,Y) do gráfico. Caso o gráfico seja tridimensional use **text(X,Y,Z,str)**.

- :

Podemos também manipular outras propriedades do gráfico, como:

- **grid on**: exibe uma grade na janela do gráfico.
- **grid off**: apaga a grade exibida na janela do gráfico.
- **axis on**: exibe os eixos de coordenadas na janela do gráfico.
- **axis off**: apaga os eixos de coordenadas da janela do gráfico.
- **axis([xmin xmax ymin ymax zmin zmax])**: ajusta os limites das coordenadas da janela onde está o gráfico. X variando de xmin à xmax, o mesmo para Y e Z.

Esses comandos podem ser utilizados tanto para desenho de gráficos quanto de superfícies. Abaixo é apresentado um exemplo dessa função sendo as janelas produzidas mostradas na Figura 1.4:

```
>> x= 1:pi/16:8*pi;
>> plot(x,sin(x))
>> title('gráfico seno')
>> xlabel('ângulo')
>> ylabel('seno')
>> axis([-5 30 -2 2])
```

1.6.5 Criando novas janelas

O MATLAB utiliza a mesma janela de desenho sempre que desenhamos um gráfico. Nesse processo o novo gráfico se sobrepõe ao último que foi desenhado. Entretanto, se quisermos criar uma nova janela para a exibição de gráficos, podemos usar a função **figure**.

A função **figure** pode ser utilizada de duas maneiras:

- **figure**: cria uma nova janela;
- **figure(H)**: torna a janela de índice **H** ativa para o desenho do gráfico.

Abaixo é apresentado um exemplo dessa função, sendo as janelas produzidas mostradas na Figura 1.5:

```
>> x = 1:10;
>> y = x.^2;
>> z = 2*x;
>> plot(x,y);
>> figure;
>> plot(x,z);
```

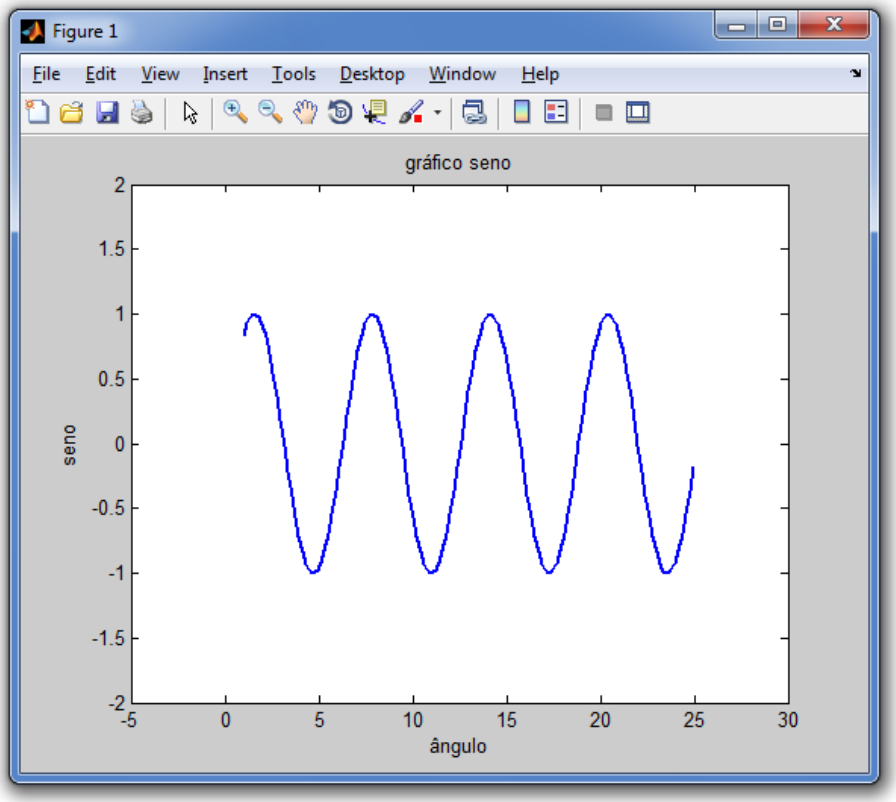


Figura 1.4:

1.6.6 Particionando a janela

Às vezes é necessário mostrar vários gráficos em uma mesma janela de desenho. Para realizar essa tarefa utilizamos a função **subplot**, cuja forma geral é

subplot(R,C,P)

Essa função divide a janela de desenho em **R x C** subjanelas, sendo **P** um número de 1 a **RxC** que representa a posição selecionada. O valor de **P** aumenta em uma unidade à medida que andamos no sentido das colunas das subjanelas. Abaixo é apresentado um exemplo dessa função, sendo as janelas produzidas mostradas na Figura 1.6:

```
>> x= 1:pi/16:8*pi;
>> subplot(2,1,1),plot(cos(x));
>> subplot(2,1,2),plot(sin(x));
```

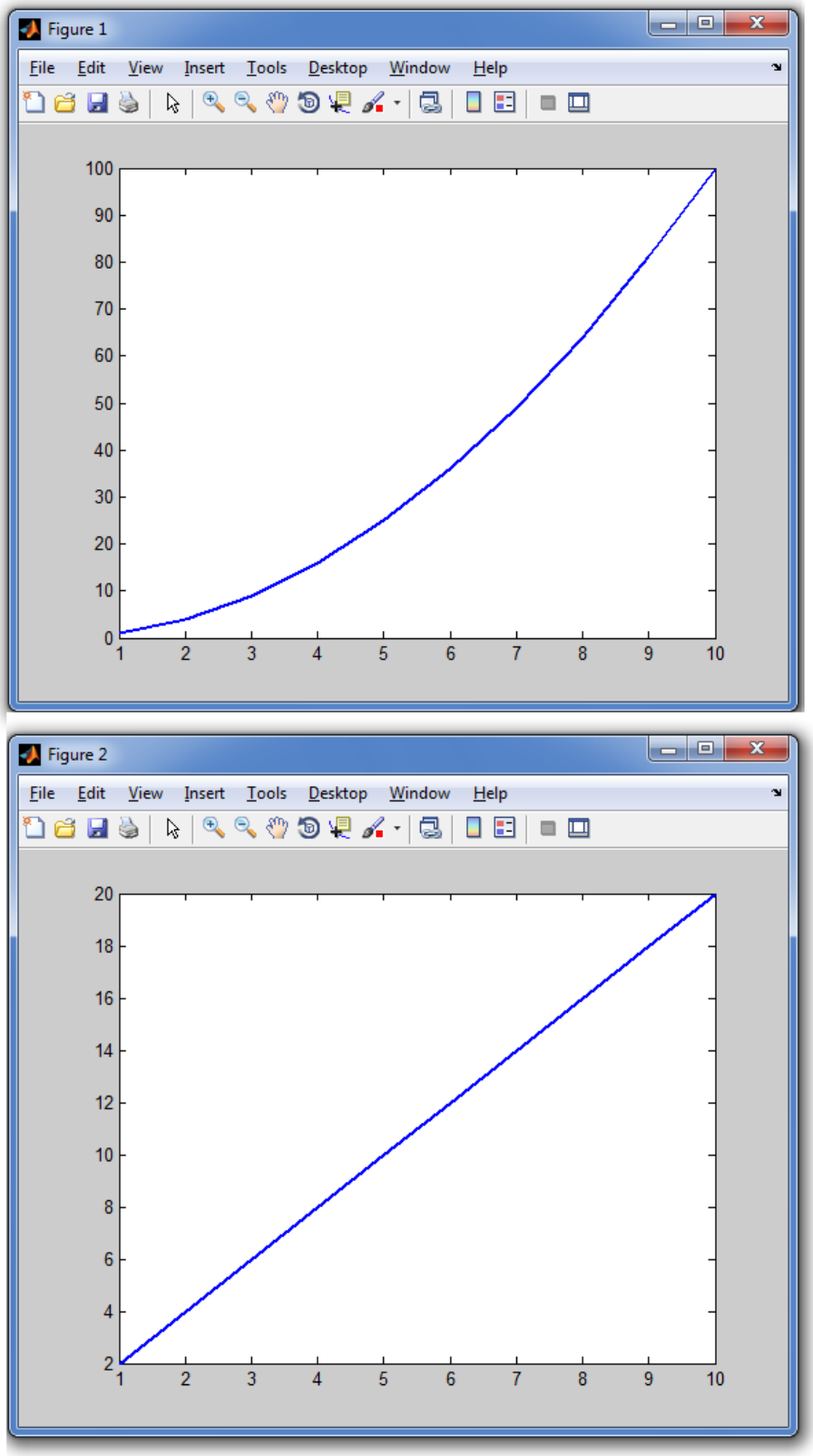


Figura 1.5:

1.7 Trabalhando com imagens

Para trabalhar com imagens, o MATLAB conta com a ajuda do Toolbox de Processamento de Imagens. Este toolbox considera as imagens como matrizes, de modo que

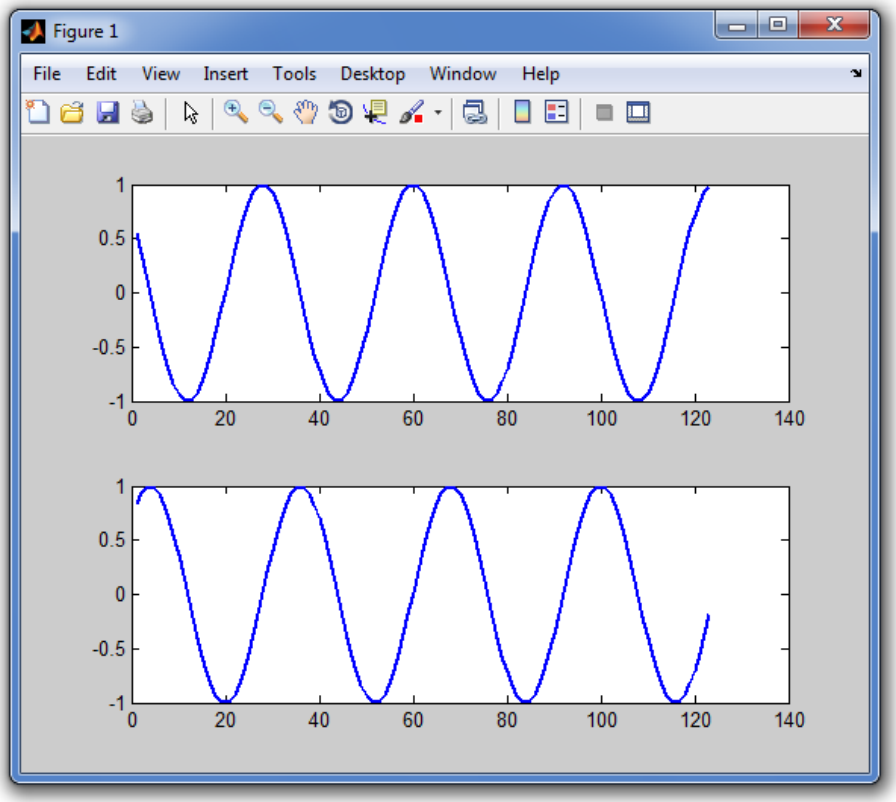


Figura 1.6:

toda e qualquer operação matricial é válida com imagens. Além disso, ele permite trabalhar com quatro tipos de imagens. São eles:

- **imagens de intensidades**: os valores da matriz representam as intensidades em cada ponto (pixel) da imagem. O intervalo destes valores dependem do tipo de dado: $[0, 255]$ para **uint8**, $[0, 65535]$ para **uint16** e $[0, 1]$ para **double**;
- **imagens binárias**: uma matriz composta de zero's e um's;
- **imagens indexadas**: utiliza duas matrizes, **X** e **map**. A matriz **X** armazena um valor numérico representando um índice para a matriz **map**, que armazena as cores existentes;
- **imagens RGB**: uma matriz com três dimensões, onde a terceira dimensão indica o canal de cor (R, G e B). Segue as mesmas regras das imagens de intensidade.

1.7.1 Abrindo e exibindo uma imagem

Para abrir uma imagem no MATLAB utilizamos a função **imread**. Essa função pode ser utilizada de várias maneiras diferentes. Apresentamos aqui duas delas:

- **im = imread(arquivo)**: abre uma imagem especificada pelo seu nome ou endereço em **arquivo** e armazena na forma de uma matriz em **im**.

- **[im,mapa] = imread(arquivo)**: abre uma imagem indexada especificada pelo seu nome ou endereço em **arquivo**. A matriz **im** armazenará os índices da cores contidas em **mapa**.

Abaixo podemos ver um exemplo de uso da função **imread**:

```
>> im = imread('d:\lena_cor.bmp');
>> size(im)
ans =
   256   256     3
```

Perceba que a variável **im** contém uma matriz com três dimensões. Isso ocorre, pois trata-se de uma imagem RGB e cada um dos seus canais é armazenado separadamente na terceira dimensão. É a combinação desses canais que produz todas as cores possíveis.

Uma vez aberta a imagem, podemos exibí-la. Para visualizar uma imagem utilizamos a função **imshow**. Essa função também pode ser utilizada de várias maneiras diferentes. Apresentamos aqui apenas duas delas:

- **imshow(im)**: exibe a imagem amrazenada em **im**. Pode ser uma imagem de intensidades, binária ou RGB;
- **imshow(im,mapa)**: exibe uma imagem indexada, onde a matriz **im** contém os índices das cores contidas em **mapa**.

Abaixo podemos ver um exemplo de uso da função **imread** enquanto a Figura 1.7 exibe a janela de visualização produzida:

```
>> im = imread('d:\lena_cor.bmp');
>> imshow(im);
```

1.7.2 Salvando uma imagem

Para salvar uma imagem do MATLAB utilizamos a função **imwrite**. Essa função pode ser utilizada de várias maneiras diferentes. Apresentamos aqui apenas três delas:

- **imwrite(im,arquivo)**: salva a imagem armazenada em **im** no local especificado em **arquivo**. **arquivo** deve conter uma das extensões suportadas pela função **imread**: 'bmp', 'png', 'jpg', etc;
- **imwrite(im,mapa,arquivo)**: salva a imagem indexada **im**, a qual contém os índices das cores contidas em **mapa**, no local especificado em **arquivo**;

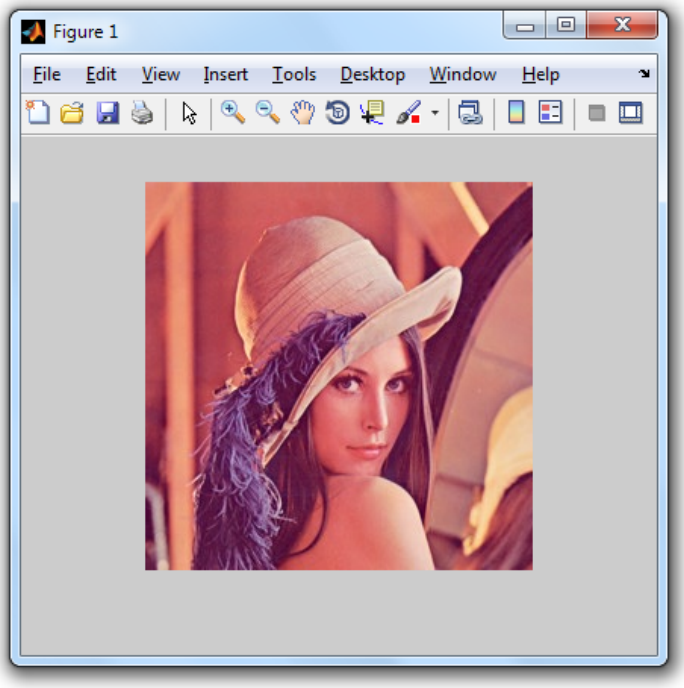


Figura 1.7:

- **imwrite(im,'arquivo.jpg','quality',valor)**: salva a imagem armazenada em **im** no local especificado em **arquivo**, cuja extensão deve ser 'jpg'. O parâmetro **valor** é um número de 0 (menos qualidade) a 100 (mais qualidade) indicando a qualidade da compressão.

Abaixo podemos ver alguns exemplos de uso da função **imwrite**:

```
>> imwrite(im, 'd:\arquivo1.jpg','quality', 50);
>> imwrite(im, 'd:\arquivo2.jpg','quality', 10);
>> imwrite(im, 'd:\arquivo3.bmp');
```