1. CAPÍTULO 1 – INTRODUÇÃO AO MATLAB

TERMOS CHAVE			CONTEÚDO
pronto	intervalo	logaritmo	
programas	caracteres	logaritmo comum	
arquivos de script	cadeias	logaritmo natural	
barra de ferramentas	conversão de tipo	constantes	
variável	aritmética de saturação	números randômicos	
comando de atribuição	default	semente	
operador de atribuição	operador de continuação	pseudo-randômico	
usuário	reticências	intervalo aberto	
inicializando	unário	fluxo global	
incrementando	operando	codificação de caracteres	
decrementando	binário	conjunto de caracteres	
nomes de identificadores	notação científica	expressão relacional	
palavras reservadas	notação exponêncial	expressão booleana	
palavras-chave	precedência	expressão lógica	
mnemônico	associatividade	operadores relacionais	
tipos	parênteses aninhados	operadores lógicos escalares	
classes	parênteses internos	operadores de curto-circuito	
dupla precisão	tópicos de ajuda	tabela de verdade	
ponto flutuante	chamar uma função	comutativo	
sem sinal	argumentos		
	valores retornando		

MATLAB® é um pacote de software muito poderoso que possui muitas ferramentas integradas para resolver problemas e desenvolver ilustrações gráficas. O método mais simples para usar o produto MATLAB é interativo; uma expressão é inserida pelo usuário e o MATLAB responde imediatamente com um resultado. Também é possível escrever scripts e programas no MATLAB, que são essencialmente grupos de comandos executados sequencialmente.

Este capítulo se concentrará no básico, incluindo muitos operadores e funções integradas que podem ser usadas em expressões interativas.

1.1 ENTRANDO NO MATLAB

MATLAB é um pacote de software matemático e gráfico com capacidades numéricas, gráficas e de programação. Ele possui funções integradas para executar muitas operações, e existem caixas de ferramentas que podem ser adicionadas para aumentar essas funções (por exemplo, para processamento de sinal). Existem versões disponíveis para diferentes plataformas de hardware, tanto nas edições profissionais quanto em estudantes.

Quando o software MATLAB é iniciado, é aberta uma janela na qual a parte principal é a Janela de Comandos (veja a Figura 1.1). Na Janela de Comandos, você deve ver:

```
>>
```

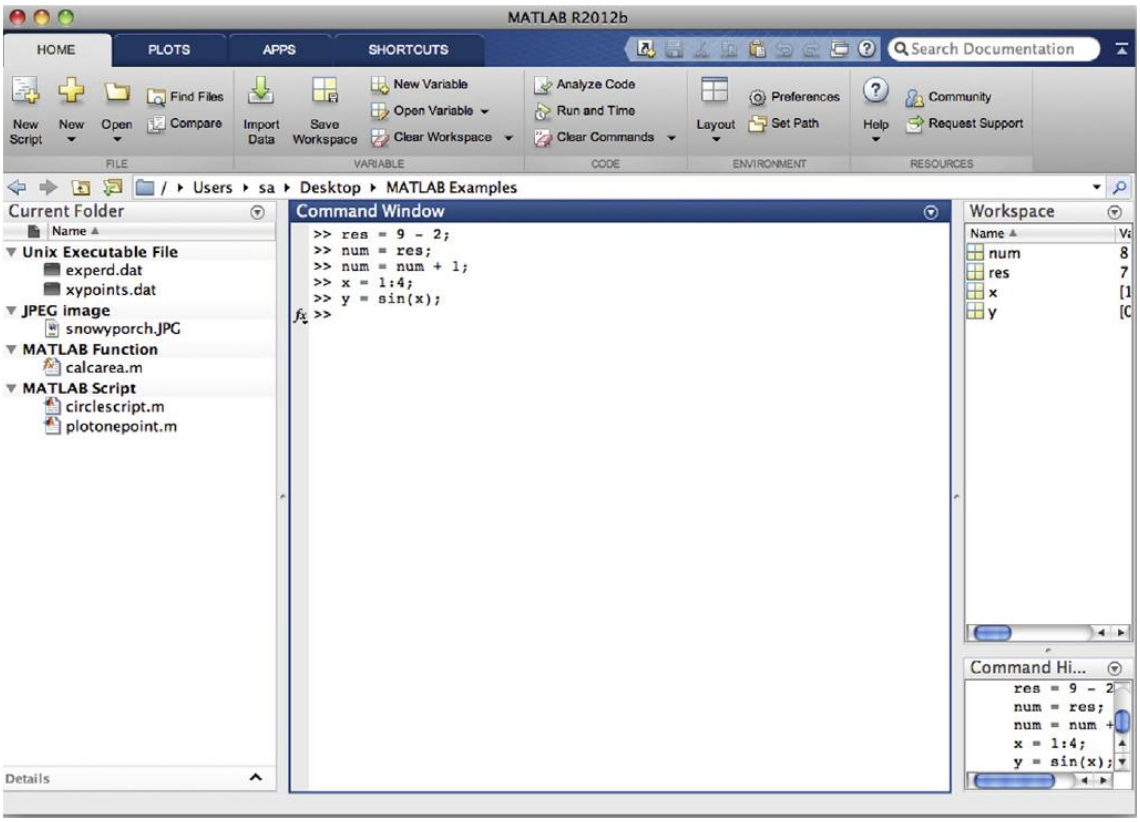

FIGURA 1.1 Janela de Comandos MATLAB

O >> é chamado de ***prompt***. Na edição do estudante, o *prompt* é, em vez disso, o seguinte:

```
EDU >>
```

Na Janela de Comandos, MATLAB pode ser usado de forma interativa. No *prompt*, qualquer comando ou expressão do MATLAB pode ser inserido e o MATLAB responderá imediatamente com o resultado.

Também é possível escrever ***programas*** no MATLAB que estão contidos em ***arquivos de script*** ou ***arquivos-M***. Os programas serão introduzidos no Capítulo 3.

Os seguintes comandos podem servir como uma introdução ao MATLAB e permitem que você obtenha ajuda:

- **demo** apresentará exemplos MATLAB no Navegador de Ajuda, que tem exemplos de alguns dos recursos do MATLAB
- **help** irá explicar qualquer função; **help help** vai explicar como a ajuda funciona
- **lookfor** pesquisa através da ajuda por uma palavra ou frase específica (nota: isso pode levar muito tempo)
- **doc** apresentará uma página de documentação no Navegador de Ajuda.

Para sair do MATLAB, digite **quit** ou **exit** no *prompt*, ou clique em MATLAB e, em seguida, **Quit** no menu do MATLAB.

1.2 O AMBIENTE DE TRABALHO DO MATLAB

Além da Janela de Comandos, existem várias outras janelas que podem ser abertas e podem ser abertas por *default* (configuração padrão). O que é descrito aqui é o *layout default* para

essas janelas na Versão R2012b, embora existam outras configurações possíveis. Diferentes versões do MATLAB podem mostrar outras configurações *default*, e o *layout* sempre pode ser personalizado. Portanto, as principais características serão descritas brevemente aqui.

À esquerda da Janela de Comandos (Command Window) é a Janela da Pasta Atual (current Folder). A pasta que está definida como a *pasta atual* é onde os arquivos serão salvos. Esta janela mostra os arquivos que estão armazenados na *pasta atual*. Estes podem ser agrupados de várias maneiras, por exemplo, por tipo e ordenados, por exemplo, por nome. Se um arquivo for selecionado, as informações sobre esse arquivo são mostradas na parte inferior.

À direita da Janela de Comandos estão a Janela de Espaço de Trabalho (Workspace) e a Janela de Histórico de Comandos (Command History) na parte inferior. A Janela de Histórico de Comandos mostra os comandos que foram inseridos, não apenas na sessão atual (na Janela de Comandos atual), mas anteriormente também. A Janela de Espaço de Trabalho será descrita na próxima seção.

Esta configuração padrão pode ser alterada clicando na seta para baixo no canto superior direito de cada janela. Isso mostrará um menu de opções (diferente para cada janela), incluindo, por exemplo, fechar essa janela em particular e desacoplar essa janela. Uma vez desacoplada, aparecerá o menu e depois, clicar na seta ondulada, que aponta para a parte inferior direita, acoplará a janela novamente. Para fazer qualquer uma dessas janelas, a janela ativa, clique com o mouse nela. Por padrão, a janela ativa é a Janela de Comandos.

A partir da versão 2012b, a aparência do ambiente de trabalho foi completamente alterada. Em vez de menus e barras de ferramentas, o Desktop agora possui uma barra de ferramentas. Por padrão, três guias são mostradas ("HOME", "PLOTS" e "APPS"), embora outros, incluindo "SHORTCUTS", possam ser adicionados.

Sob a aba "HOME", existem muitos recursos úteis, que são divididos em seções funcionais "FILE", "VARIABLE", "CODE", "ENVIRONMENT" e "RESOURCES" (estes rótulos podem ser vistos no fundo da área cinza de ferramentas). Por exemplo, em "ENVIRONMENT", pressionando a seta para baixo sob Layout, permite a personalização das janelas do Ambiente de Trabalho. Outras características da ferramenta serão introduzidas em capítulos posteriores quando o material relevante for explicado.

1.3 VARIÁVEIS E COMANDO DE ATRIBUIÇÃO

Para armazenar um valor em uma sessão do MATLAB, ou em um programa, é usada uma **variável**. A Janela da Área de Trabalho mostra as variáveis que foram criadas e seus valores.

Uma maneira fácil de criar uma variável é usar uma *declaração de atribuição*. O formato de uma declaração de atribuição é

```
nomedavariável = expressão
```

A variável está sempre à esquerda, seguida pelo símbolo =, que é o *operador de atribuição* (ao contrário da matemática, um único sinal de igual não significa igualdade), seguido de uma **expressão**. A expressão é avaliada e, em seguida, esse valor é armazenado na variável. Aqui está um exemplo e como ele apareceria na Janela de Comandos:

```
>> meunum = 6
meunum =
     6
>>
```

Aqui, o usuário (a pessoa que trabalha no MATLAB) digitou "meunum = 6" no *prompt* e MATLAB armazenou o inteiro 6 na variável chamada *meunum* e, em seguida, exibiu o resultado seguido pelo *prompt* novamente. Como o sinal de igualdade é o operador de atribuição e não significa igualdade, a declaração deve ser lida como "*meunum* recebe o valor de 6" (e não "meunum é igual a 6").

Observe que o nome da variável deve estar sempre à esquerda e a expressão à direita. Um erro ocorrerá se estes forem invertidos.

```
>> 6 = meunum
   6 = meunum
     |
Error: The expression to the left of the equals sign is not a
valid target for an assignment.
>>
```

Colocar um ponto-e-vírgula no final de uma declaração suprime a saída. Por exemplo,

```
>> res = 9 e 2;
>>
```

Isso atribuiria o resultado da expressão do lado direito, o valor 7, para a variável *res*; somente não mostra esse resultado. Em vez disso, outro *prompt* aparece imediatamente. No entanto, neste ponto da Janela de Espaço de Trabalho, as variáveis *meunum* e *res* e seus valores podem ser vistas.

Os espaços em uma declaração ou expressão não afetam o resultado, mas facilitam a leitura. A seguinte declaração, que não tem espaços, realizaria exatamente o mesmo resultado que a declaração anterior:

```
>> res=9-2;
>>
```

MATLAB usa uma **variável padrão** chamada *ans*, se uma expressão for digitada no *prompt* e não for atribuída a uma variável. Por exemplo, o resultado da expressão 6 + 3 é armazenado na variável *ans*:

```
>> 6 + 3
ans =
     9
```

Esta *variável padrão* é reutilizada a qualquer momento, quando apenas uma expressão é digitada no *prompt*.

Um atalho para reescrever comandos é pressionar a seta para cima ↑, que retornará ao(s) comando(s) anteriormente digitado(s). Por exemplo, se você decidiu atribuir o resultado da expressão 6 + 3 a uma variável chamada resultado ao invés de usar a variável padrão *ans*, você

poderia pressionar a seta para cima e depois a seta para a esquerda para modificar o comando em vez de redigitar toda a declaração:

```
>> resultado = 6 + 3
resultado =
     9
```

Isso é muito útil, especialmente se uma expressão longa é inserida e contém um erro, e é desejável voltar e corrigi-lo.

Para alterar uma variável, pode-se usar outra *declaração de atribuição*, que atribui o valor de uma expressão diferente para ele. Considere, por exemplo, a seguinte sequência de declarações:

```
>> meunum = 3
meunum =
     3
>> meunum = 4 + 2
meunum =
     6
>> meunum = meunum + 1
meunum =
     7
```

Na primeira *declaração de atribuição*, o valor 3 é atribuído à variável *meunum*. Na próxima declaração de atribuição, *meunum* é alterado para ter o valor da expressão 4 + 2 ou 6. Na terceira *declaração de atribuição*, o *meunum* é alterado novamente, para o resultado da expressão meunum + 1. Como, nesse momento, *meunum* tinha o valor 6, o valor da expressão era 6 + 1 ou 7.

Nesse ponto, se a expressão *meunum* + 3 for inserida, a variável padrão *ans* é usada como resultado dessa expressão e não é atribuída a uma outra variável. Assim, o valor de *ans* se torna 10, mas *meunum* é inalterado (e ainda é 7). Observe que apenas digitar o nome de uma variável exibirá seu valor (claro que o valor também pode ser visto na Janela de Espaço de Trabalho).

```
>> meunum + 3
ans =
     10
>> meunum
meunum =
     7
```

1.3.1 Inicializando, Incrementando e Decrememtando

Freqüentemente, os valores das variáveis mudam, como mostrado anteriormente. Colocar o valor inicial ou inicial em uma variável é chamado de inicialização da variável.

Adicionar a uma variável é chamado de incremento. Por exemplo, a declaração *meunum* = *meunum* + 1

incrementa a variável *meunum* de 1.

PERGUNTA RÁPIDA!

Como 1 pode ser subtraído do valor de uma variável chamada num?

Resposta

```
num = num - 1;
```

Isso é chamado de decrementar a variável.

1.3.2 Nomes de Variáveis

Nomes de variáveis são exemplos de *nomes de identificadores*. Veremos outros exemplos de nomes de identificadores, como nomes de funções, em capítulos futuros. As regras para nomes de identificadores são as seguintes.

- O nome deve começar com uma letra do alfabeto. Depois disso, o nome pode conter letras, dígitos e o caractere de sublinhado (por exemplo, value_1), mas não pode ter espaço.
- Existe um limite para o comprimento do nome; A função interna **namelengthmax** informa o tamanho máximo deste (todos os caracteres extras são truncados).
- MATLAB é *case-sensitive*, o que significa que diferencia entre letras maiúsculas e minúsculas. Então, as variáveis chamadas *meunum*, *MEUNUM* e *Meunum* são diferentes (embora isso seja confuso e não deve ser feito).
- Embora os caracteres de sublinhado sejam válidos em um nome, seu uso pode causar problemas com alguns programas que interagem com MATLAB, então alguns programadores usam maiúsculas e minúsculas em vez disso (por exemplo, pesoDaPeca em vez de peso_da_peca).
- Existem certas palavras chamadas *palavras reservadas*, ou *palavras-chave*, que não podem ser usadas como nomes de variáveis.
- Os nomes das funções integradas (descritos na próxima seção) podem, mas não devem, ser usados como nomes de variáveis.

Além disso, nomes de variáveis sempre devem ser *mnemônicos*, o que significa que eles devem ter algum sentido. Por exemplo, se a variável estiver armazenando o raio de um círculo, um nome como o *raio* faria sentido; *x* provavelmente não.

Os seguintes comandos referem-se a variáveis:

- **who** mostra variáveis que foram definidas nesta Janela de Comandos (isso mostra apenas os nomes das variáveis)
- **whos** mostra variáveis que foram definidas nesta Janela de Comandos (isso mostra mais informações sobre as variáveis, semelhante ao que está na Janela de Espaço de Trabalho)
- **clear** apaga todas as variáveis para que elas não existam mais
- **clear** *nomedevariavel* apaga uma variável particular
- **clear** *nomedevariavel1 nomedevariavel2* ... apaga uma lista de variáveis (nota: separe os nomes com espaços).

Se nada aparecer quando **who** ou **whos** são inseridos, isso significa que não há nenhuma variável. Por exemplo, no início de uma sessão MATLAB, as variáveis podem ser criadas e, em seguida, seletivamente apagadas (lembre-se de que o ponto-e-vírgula suprime a saída).

```
>> who
>> meunum = 3;
>> meunum + 5;
>> who
Your variables are:
ans meunum
>> clear meunum
>> who
```

```
Your variables are:
ans
```

Essas mudanças também podem ser vistas na Janela de Espaço de Trabalho.

1.3.3 Tipos

Cada variável tem um tipo associado a ela. O MATLAB suporta muitos tipos, que são chamados de classes. (Essencialmente, uma classe é uma combinação de um tipo e as operações que podem ser realizadas nos valores desse tipo, mas, por simplicidade, usaremos esses termos de forma intercambiável por enquanto).

Por exemplo, existem tipos para armazenar diferentes espécies de números. Para números reais de ponto flutuante ou números reais, ou, em outras palavras, números com casas decimais (por exemplo, 5.3), existem dois tipos básicos: **single** e **double**. O nome do tipo **double** é o abreviado para ***dupla precisão***; Ele armazena números maiores do que o tipo **single**. MATLAB usa uma representação de ***ponto flutuante*** para esses números.

Existem muitos tipos inteiros, tais como **int8**, **int16**, **int32** e **int64**. Os números nos nomes representam o número de bits usados para armazenar valores desse tipo. Por exemplo, o tipo **int8** usa oito *bits*, no total para armazenar o inteiro e seu sinal. Como um *bit* é usado para o sinal, isso significa que sete *bits* são usados para armazenar o valor do número (0s ou 1s). Existem também tipos inteiros sem sinal **uint8**, **uint16**, **uint32** e **uint64**. Para esses tipos, o sinal não é armazenado, o que significa que o inteiro só pode ser positivo (ou 0).

O *intervalo de um tipo*, que indica os menores e maiores números que podem ser armazenados no tipo, pode ser calculado. Por exemplo, o tipo **uint8** armazena 2^8 ou 256 inteiros, variando de 0 a 255. O intervalo de valores que podem ser armazenados no **int8**, no entanto, é de -128 a +127. O intervalo pode ser encontrado para qualquer tipo passando o nome do tipo como uma *string* (isto é, uma *cadeia de caracteres* entre *apóstrofos*) para as funções **intmin** e **intmax**. Por exemplo,

```
>> intmin ('int8')
ans =
 -128
>> intmax ('int8')
ans =
  127
```

Quanto maior o número no nome do tipo, maior o número que pode ser *armazenado* nele. Nós usaremos, na maioria das vezes, o tipo **int32** quando for necessário um tipo inteiro.

O tipo **char** é usado para armazenar um *único caractere* (por exemplo, 'x') ou *strings* (cadeia de caracteres), que são sequências de caracteres (por exemplo, 'gato'). Ambos, os *caracteres* e as *strings* são colocados entre apóstrofos.

O tipo **lógical** é usado para armazenar valores **true** (verdadeiro) ou **false** (falso).

As variáveis que foram criadas na Janela de Comandos podem ser vistas na Janela de Espaço de Trabalho. Nessa janela, podem ser vistos para cada variável, o nome da variável, o valor e a classe (que é, essencialmente, seu tipo). Outros atributos das variáveis também podem ser vistos na Janela de Espaço de Trabalho. Quais atributos são visíveis por padrão dependem da versão do MATLAB.

No entanto, quando a Janela de Espaço de Trabalho é escolhida, pressionar na seta-para-baixo permite ao usuário escolher quais atributos serão exibidos, modificando Choose Culumns (Escolher Colunas).

Por padrão, os números são armazenados como o tipo **double** no MATLAB. Existem, no entanto, muitas funções que convertem valores de um tipo para outro. O nome dessas funções é o mesmo que os nomes dos tipos exibidos nesta seção. Esses nomes podem ser usados como funções para converter um valor nesse tipo. Isso é chamado de *moldar* (*casting*) o valor para um tipo diferente, ou *conversão de tipo*. Por exemplo, para converter um valor do tipo **double**, que é o padrão, para o tipo **int32**, a função **int32** seria usada. Inserindo a declaração de atribuição

```
>> val = 6 + 3;
```

resultaria no número 9 que está sendo armazenado na variável *val*, com o tipo padrão de **double**, que pode ser visto na Janela de Espaço de Trabalho. Posteriormente, a declaração de atribuição

```
>> val = int32(val);
```

mudaria o tipo da variável para **int32**, mas não mudaria seu valor. Aqui está outro exemplo usando duas variáveis diferentes.

```
>> num = 6 + 3;
>> numi = int32(num);
>> whos
   Name          Size        Bytes      Class      Attributs
   num           1x1             8      double
   numi          1x1             4      int32
```

Observe que **whos** mostra o tipo (classe) das variáveis, bem como o número de *bytes* usados para armazenar o valor de uma variável. Um *byte* é equivalente a oito *bits*, então o tipo **int32** usa quatro *bytes*. A função **class** também pode ser usada para ver o tipo de variável:

```
>> class(num)
ans =
double
```

Um motivo para usar um tipo inteiro para uma variável é economizar espaço na memória.

PERGUNTA RÁPIDA!

O que aconteceria se você for além do intervalo de um tipo específico? Por exemplo, o inteiro maior que pode ser armazenado no **int8** é 127, então o que aconteceria se tentássemos converter um inteiro maior para o tipo **int8**?

```
>> int8(200)
```

Resposta

O valor seria o maior no intervalo, neste caso 127. Se, em vez disso, usamos um número negativo que é menor do que o valor mais baixo do intervalo, seu valor seria -128. Este é um exemplo da chamada ***aritmética de saturação***.

```
>> int8(200)
ans =
  127
>> int8(-130)
```

```
ans =
 -128
```

PRÁTICA 1.1

- Calcule o intervalo de inteiros que podem ser armazenados nos tipos **int16** e **uint16**. Use **intmin** e **intmax** para verificar seus resultados.
- Digite uma declaração de atribuição e veja o tipo da variável na Janela de Espaço de Trabalho. Então, mude seu tipo e veja-o novamente. Veja também usando **whos**.

1.4 EXPRESSÕES NUMÉRICAS

As expressões podem ser criadas usando valores, variáveis que já foram criadas, operadores, funções internas e parênteses. Para os números, estes podem incluir operadores, como a multiplicação e as funções, como as funções trigonométricas. Um exemplo dessa expressão é:

```
>> 2 * sin(1.4)
ans =
    1.9709
```

1.4.1 Função de Formato e Reticências

O *padrão* no MATLAB é exibir números com pontos decimais com quatro casas decimais, como mostrado no exemplo anterior. (O *padrão* significa que se você não especificar o contrário, é isso que você obtém). O comando **format** pode ser usado para especificar o formato de saída das expressões.

Existem muitas opções, incluindo tornar o formato **short** (curto, o padrão) ou **long**. Por exemplo, alterar o formato para **long** resultará em 15 casas decimais. Isso permanecerá em vigor até que o formato seja alterado de novo para **short**, conforme demonstrado no seguinte:

```
>> format long
>> 2 * sin (1.4)
ans =
    1.970899459976920
>> format short
>> 2 * sin (1.4)
ans =
    1.9709
```

O comando **format** também pode ser usado para controlar o espaçamento entre o comando MATLAB, ou a expressão, e o resultado; pode ser **loose** (o padrão) ou **compact**.

```
>> format loose
>> 5 * 33
ans =

   165

>> format compact
>> 5 * 33
ans =
   165
>>
```

Particularmente, expressões longas podem ser continuadas na próxima linha digitando três (ou mais) pontos, que é o ***operador de continuação***, ou ***reticências***. Para fazer isso, digite parte da

expressão seguida de reticências e, em seguida, pressione a tecla ***Enter*** e continue escrevendo a expressão na próxima linha.

```
>> 3 + 55 - 62 + 4 - 5...
+ 22 - 1
ans =
    16
```

1.4.2 Operadores

Existem, em geral, dois tipos de operadores: operadores ***unários***, que operam em um único valor, ou operando, e operadores ***binários***, que operam em dois valores ou operandos. O símbolo "-", por exemplo, é o operador unário para negação e o operador binário para subtração.

Aqui estão alguns dos operadores comuns que podem ser usados com expressões numéricas:

+ adição
- negação, subtração
* multiplicação
/ divisão (dividida por, por exemplo, 10/5 é 2)
\ divisão (dividida em, por exemplo, 5\10 é 2)
^ exponenciação (por exemplo, 5^2 é 25)

Além de exibir números com pontos decimais, os números também podem ser exibidos usando notação científica ou exponencial. Isso usa e para o expoente de 10 aumentado para uma potência. Por exemplo, 2 * 10 ^ 4 poderia ser escrito de duas maneiras:

```
>> 2 * 10 ^ 4
ans =
     20000
>> 2e4
ans =
     20000
```

1.4.2.1 *Regras de Precedência do Operador*

Alguns operadores têm precedência sobre outros. Por exemplo, na expressão 4 + 5 * 3, a multiplicação tem precedência sobre a adição, então, primeiro 5 é multiplicado por 3, então 4 é adicionado ao resultado. Usar parênteses pode alterar a precedência em uma expressão:

```
>> 4 + 5 * 3
ans =
    19
>> (4 + 5) * 3
ans =
    27
```

Dentro de um determinado nível de precedência, as expressões são avaliadas da esquerda para a direita (isto é chamado de ***associatividade***).

Os parênteses aninhados são parênteses dentro de outros; A expressão nos parênteses internos é avaliada primeiro. Por exemplo, na expressão 5 - (6 * (4 + 2)), primeiro a adição é realizada, depois a multiplicação e, finalmente, a subtração, resultando em -31. Parênteses também podem ser usados simplesmente para fazer uma expressão mais clara. Por exemplo, na expressão ((4 + (3 * 5)) - 1), os parênteses não são necessários, mas são usados para mostrar a ordem em que as partes da expressão serão avaliadas.

Para os operadores que foram vistos até agora, a precedência é a seguinte (do mais alto para o mais baixo):

()	parênteses
^	exponenciação
-	negação
*, /, \	multiplicação e divisão
+, -	adição e subtração

PRÁTICA 1.2

Pense sobre quais os resultados das seguintes expressões e digite-as para verificar suas respostas:

```
1 \ 2
- 5 ^ 2
(-5) ^ 2
10 - 6 / 2
5 * 4 / 2 * 3
```

1.4.3 Funções Internas e Ajuda

Existem muitas funções internas no MATLAB. O comando **help** pode ser usado para identificar funções MATLAB, e também como usá-las. Por exemplo, digitar **help** no *prompt* na Janela de Comandos mostrará uma lista de ***tópicos de ajuda*** que são grupos de funções relacionadas. Esta é uma lista muito longa; Os tópicos de ajuda mais elementares aparecem no início. Além disso, se você tiver caixas de ferramentas (Tool-boxes) instaladas, estas serão listadas.

Por exemplo, um dos tópicos de ajuda elementar é listado como **matlab\elfun**; inclui as funções matemáticas elementares. Outro dos primeiros tópicos de ajuda é **matlab\ops**, que mostra os operadores que podem ser usados em expressões.

Para ver uma lista das funções contidas em um tópico de ajuda específico, digite **help** seguida do nome do tópico. Por exemplo,

```
>> help elfun
```

mostrará uma lista das funções matemáticas elementares. É uma lista muito longa, e é dividida em trigonométrica (para o qual o padrão é radiano, mas existem funções equivalentes que em vez de usar graus), exponencial, complexo e arredondamento e funções de resto.

Para descobrir o que uma função específica faz e como chamá-la, digite **help** e depois o nome da função. Por exemplo, o seguinte dará uma descrição da função **sin** (*seno*).

```
>> help sin
```

Note que clicar no *fx* à esquerda do *prompt* na Janela de Comandos também permite navegar pelas funções nos tópicos de ajuda. Escolher o botão Help (Ajuda) sob Resources (Recursos) para exibir a página de Documentação do MATLAB é outro método para encontrar funções por categoria.

Para ***chamar uma função***, o nome da função é fornecido seguido do(s) ***argumento(s)*** que são passados para a função entre parênteses. A maioria das funções ***retorna valor(es)***. Por exemplo, para encontrar o valor absoluto de -4, a seguinte a expressão seria inserida:

```
>> abs(-4)
```

que é uma chamada para a função **abs**. O número entre parênteses, o -4, é o ***argumento***. O valor 4 seria então ***retornado*** como resultado.

PERGUNTA RÁPIDA!

O que aconteceria se você usar o nome de uma função, por exemplo, **sin**, como um nome de variável?

Resposta

Isso é permitido no MATLAB, mas o **sin** não pode ser usado como função interna até que a variável seja apagada. Por exemplo, examine a seguinte sequência:

```
>> sin(3.1)
ans =
    0.0416
>> sin = 45
sin =
    45
>> sin(3.1)
Subscript indices must either be real positive integers or
logical.
>> who
Your variables are:
ans sin
>> clear sin
>> who
Your variables are:
ans
>> sin(3.1)
ans =
    0.0416
```

Além das funções trigonométricas, o tópico de ajuda do **elfun** também possui algumas funções de arredondamento e resto que são muito úteis. Alguns destes incluem **fix**, **floor**, **ceil**, **round**, **mod**, **rem** e **sign**.

As funções **rem** e **mod** retornam o resto de uma divisão; por exemplo, 5 entra em 13 duas vezes com um resto de 3, então o resultado dessa expressão é 3:

```
>> rem(13, 5)
ans =
    3
```

PERGUNTA RÁPIDA!

O que aconteceria se você revertesse a ordem dos argumentos por engano e digitasse o seguinte:

```
rem(5,13)
```

Resposta

A função **rem** é um exemplo de uma função que tem dois argumentos passados para ela. Em alguns casos, a ordem em que eles são passados não é importante, mas para **rem** importa. A função **rem** divide o segundo argumento no primeiro. Neste caso, o segundo argumento, 13, entra em 5 vezes zero com um resto de 5, então 5 é retornado como resultado.

Outra função no tópico de ajuda do **elfun** é a função **sign**, que retorna 1 se o argumento for positivo, 0 se for 0 e -1 se for negativo. Por exemplo,

```
>> sign(-5)
ans =
   -1
>> sign(3)
ans =
    1
```

PRÁTICA 1.3

Use a função **help** para descobrir o que as funções de arredondamento **fix**, **floor**, **ceil** e **round**. Faça experiências com elas passando valores diferentes para as funções, incluindo alguns números negativos, alguns positivos e alguns com frações menores que 0,5 e alguns maiores. *É muito importante ao testar as funções, que você teste completamente, tentando diferentes tipos de argumentos!*

MATLAB tem o operador de exponenciação ^, e também a função **sqrt** para calcular raízes quadradas e **nthroot** para encontrar a ***n-ésima*** raiz de um número. Por exemplo, a seguinte expressão encontra a terceira raiz de 64:

```
>> nthroot(64,3)
ans =
    4
```

Para o caso em que $x = b^y$, y é o **logaritmo** de x para a base b, ou, em outras palavras, $y=\log_b(x)$. As bases utilizadas frequentemente incluem b = 10 (chamado de ***logaritmo comum***), b = 2 (usado em muitas aplicações de computação) e b = e (a ***constante e***, que é igual a 2.7183); Isso é chamado de ***logaritmo natural***. Por exemplo,

$$100 = 10^2 \text{ assim } 2 = \log_{10}(100)$$
$$32 = 2^5 \text{ assim } 5 = \log_2(32)$$

O MATLAB possui funções integradas para retornar logaritmos:

- **log(x)** retorna o logaritmo natural
- **log2(x)** retorna o logaritmo da base 2
- **log10(x)** retorna o logaritmo da base 10.

MATLAB também possui uma função interna **exp(n),** que retorna a constante e^n.

MATLAB tem muitas funções trigonométricas internas para seno, coseno, tangente e assim por diante. Por exemplo, a **sin** é a função seno em radianos. A função inversa ou arco seno em radianos é **asin**, a função seno hiperbólico em radianos é **sinh**, e a função seno hiperbólico inversa é **asinh**. Existem também funções que usam graus em vez de radianos: **sind** e **asind**. Existem variações semelhantes para as outras funções trigonométricas.

1.4.4 Constantes

As variáveis são usadas para armazenar valores que podem mudar, ou para os quais os valores não são conhecidos antes do tempo. A maioria das línguas também tem a capacidade de armazenar constantes, que são valores conhecidos antes do tempo e que não podem ser alterados. Um exemplo de um valor constante seria pi ou p, que é 3.14159.

No MATLAB, existem funções que retornam alguns desses valores constantes, alguns dos quais incluem:

pi 3.14159.

i $\sqrt{-1}$

j $\sqrt{-1}$

inf infinito ∞

NaN significa “not a number” (não é um número), como o resultado de 0/0.

PERGUNTA RÁPIDA!

Não existe uma constante interna para **e** (2.718), então, como esse valor pode ser obtido no MATLAB?

Resposta

Use a função exponencial **exp**; e ou e^1 é equivalente a **exp(1)**.

```
>> exp(1)
ans =
    2.7183
```

Nota: não confunda o valor e com o e usado no MATLAB para especificar um expoente na notação científica.

1.4.5 Números Randômicos

Quando um programa está sendo escrito para trabalhar com dados, e os dados ainda não estão disponíveis, muitas vezes é útil testar o programa primeiro, inicializando as variáveis de dados com ***números randômicos*** *(aleatórios)*. Os números randômicos também são úteis em simulações. Existem várias funções internas no MATLAB que geram números randômicos, alguns dos quais serão ilustrados nesta seção.

Geradores de números randômicos ou funções não são verdadeiramente randômicos. Basicamente, a maneira como ele funciona é que o processo começa com um número, que é chamado de ***semente***. Frequentemente, a semente inicial é um valor predeterminado ou é obtida a partir do relógio incorporado no computador. Então, com base nesta semente, um processo determina o próximo "número randômico". Usando esse número como a semente na próxima vez, outro número randômico é gerado, e assim por diante. Estes são chamados de ***pseudo-randômicos*** e eles não são verdadeiramente randômicos porque existe um processo que determina o próximo valor de cada vez.

A função **rand** pode ser usada para gerar números reais randômicos uniformemente distribuídos; chamando-o gera um número real randômico no intervalo aberto (0,1), o que significa que os pontos finais do intervalo não estão incluídos. Não há argumentos passados para a função **rand** em sua forma mais simples. Aqui estão dois exemplos de chamar a função **rand**:

```
>> rand
ans =
    0.8147
>> rand
ans =
    0.9058
```

A semente para a função **rand** será sempre a mesma sempre que MATLAB for iniciado, a menos que a semente inicial seja alterada. Muitas das funções aleatórias e geradores de

números randômicos foram atualizadas em versões recentes do MATLAB; como resultado, os termos 'semente' e 'estado' anteriormente utilizados em funções aleatórias não devem mais ser usados. A função **rng** define a semente inicial. Existem várias maneiras pelas quais ela pode ser chamado:

```
>> rng('shuffle')
>> rng(intseed)
>> rng('default')
```

Com 'shuffle', a função **rng** usa a data e a hora atuais que são retornadas da função **clock** interna para definir a semente, então a semente será sempre diferente. Um número inteiro também pode ser passado para ser a semente. A opção 'default' definirá a semente para o valor padrão usado quando o MATLAB for iniciado. A função **rng** também pode ser chamada sem argumentos, que retornará o estado atual do gerador de números randômicos:

```
>> state_rng = rng; % obtém estado
>> randone = rand
randone =
    0.1270
>> rng (state_rng); % restaura o estado
>> randtwo = rand   % o mesmo que randone
randtwo =
    0.1270
```

Nota
As palavras após o % são comentários e são ignoradas pelo MATLAB.

O gerador de números randômicos é inicializado quando o MATLAB é iniciado, o que gera o que é chamado de ***fluxo global*** de números randômicos. Todas as funções aleatórias obtêm seus valores desse fluxo.

À medida que **rand** retorna um número real no intervalo aberto (0, 1), multiplicar o resultado por um número inteiro N retornaria um número real randômico no intervalo aberto (0, N). Por exemplo, multiplicar por 10 retorna um número real no intervalo aberto (0, 10), então a expressão

```
rand * 10
```

retornaria um resultado no intervalo aberto (0, 10).

Para gerar um número real randômico no intervalo de baixo para alto, primeiro crie as variáveis *baixo* e *alto*. Então, use a expressão *rand*(alto-baixo)+baixo*. Por exemplo, a seqüência

```
>> baixo = 3;
>> alto = 5;
>> rand * (alto - baixo) + low
```

geraria um número real randômico no intervalo aberto (3, 5).

A função **randn** é usada para gerar números reais randômicos normalmente distribuídos.

1.4.5.1 *Gerando Inteiros Randômicos*

Como a função **rand** retorna um número real, isso pode ser arredondado para produzir um número inteiro randômico. Por exemplo,

```
>> round(rand * 10)
```

geraria um número inteiro randômico no intervalo de 0 a 10 inclusive (rand * 10 geraria um número real randômico no intervalo aberto (0, 10), o arredondamento que retornará um número inteiro). No entanto, esses números inteiros não seriam distribuídos uniformemente no intervalo. Um método melhor é usar a função **randi**, que, na sua forma mais simples, **randi (imax)**, retorna um número inteiro randômico no intervalo de 1 a *imax*, inclusive. Por exemplo, **randi(4)** retorna um número inteiro randômico no intervalo de 1 a 4. Um intervalo também pode ser passado; por exemplo, **randi([imin, imax])** retorna um inteiro randômico no intervalo inclusivo de imin para imax:

```
>> randi ([3, 6])
ans =
     4
```

PRÁTICA 1.4

Gerar um número randômico

- real na faixa (0,1)
- real na faixa (0, 100)
- real na faixa (20, 35)
- inteiro na faixa inclusa de 1 a 100
- inteiro na faixa inclusiva de 20 a 35.

1.5 CARACTERES E CODIFICAÇÃO

Um caractere em MATLAB é representado usando apóstrofos (por exemplo, 'a' ou 'x'). Os apóstrofos são necessárias para denotar um caractere; sem eles, uma letra seria interpretada como um nome de variável. Os caracteres são colocados em uma ordem usando o que é chamado de uma ***codificação de caracteres***. Na codificação de caracteres, todos os caracteres no ***conjunto de caracteres*** do computador são colocados em uma sequência e a eles são dados valores inteiros equivalentes. O conjunto de caracteres inclui todas as letras do alfabeto, dígitos e sinais de pontuação; basicamente, todas as teclas em um teclado são caracteres. Caracteres especiais, como a tecla Enter, também estão incluídos. Então, 'x', '!' e '3' são todos caracteres. Com apóstrofos, '3' é um caractere, não um número.

A codificação de caracteres mais comum é o American Standard Code for Information Interchange, ou ASCII. ASCII padrão tem 128 caracteres, que possuem valores inteiros equivalentes de 0 a 127. Os primeiros 32 (valores inteiros de 0 a 31) são caracteres não imprimíveis. As letras do alfabeto estão em ordem, o que significa que 'a' vem antes de 'b', então 'c', e assim por diante.

As funções numéricas podem ser usadas para converter um caractere em seu valor numérico equivalente (por exemplo, **double** será convertido em um valor **double** e **int32** converterá em um valor inteiro usando 32 bits). Por exemplo, para converter o caractere 'a' em seu equivalente numérico, a seguinte declaração pode ser usada:

```
>> numequiv = double('a')
numequiv =
     97
```

Isso armazena o valor duplo 97 na variável *numequiv*, que mostra que o caractere 'a' é o 98º caractere na codificação de caracteres (como os números equivalentes começam em 0). Não importa qual tipo de número é usado para converter 'a'; por exemplo,

```
>> numequiv = int32('a')
```

também armazenaria o valor inteiro 97 na variável *numequiv*. A única diferença entre estes será o tipo da variável resultante (**double** no primeiro caso, **int32** no segundo).

A função **char** faz o inverso; converte de qualquer número para o caractere equivalente:

```
>> char(97)
ans =
a
```

À medida em que as letras do alfabeto estão em ordem, o caractere 'b' tem o valor equivalente de 98, 'c' é 99, e assim por diante. A matemática pode ser aplicada em caracteres. Por exemplo, para obter o próximo caractere na codificação de caracteres, 1 pode ser adicionado ao inteiro ou ao caracter:

```
>> numequiv = double('a');
>> char(numequiv + 1)
ans =
b
>> 'a' + 2
ans =
    99
```

Observe a diferença na formatação (o recuo) quando um número é exibido em relação a um caractere:

```
>> var = 3
var =
    3
>> var = '3'
var =
3
```

MATLAB também lida com strings, que são sequências de caracteres entre apóstrofos. Por exemplo, usando a função **double** em uma string mostrará o valor numérico equivalente de todos os caracteres na string:

```
>> double('abcd')
ans =
    97  98  99 100
```

Para mudar os caracteres de uma string "up" na codificação de caracteres, um valor inteiro pode ser adicionado a uma string. Por exemplo, a seguinte expressão mudará em um:

```
>> char ('abcd' + 1)
ans =
bcde
```

PRÁTICA 1.5

- Encontre o equivalente numérico do caractere 'x'.
- Encontre o caractere equivalente de 107.

Nota
Os apóstrofos não são exibidos quando o caractere é exibido.

1.6 EXPRESSÕES RELACIONAIS

As expressões que são conceitualmente **true** (verdadeira) ou **false** (falsa) são chamadas de ***expressões relacionais***; eles também são às vezes chamados de ***expressões booleanas*** ou ***expressões lógicas***. Essas expressões podem usar ambos os ***operadores relacionais***, que relacionam duas expressões de tipos compatíveis e ***operadores lógicos***, que operam sobre operandos ***lógicos***.

Os operadores relacionais em MATLAB são:

Operador	Significado
>	maior que
<	menor que
>=	maior ou igual
<=	menor ou igual a
==	igual a
~=	diferente de

Todos esses conceitos devem ser familiares, embora os operadores reais utilizados possam ser diferentes dos usados em outras linguagens de programação ou nas aulas de matemática. Em particular, é importante notar que o operador para a igualdade são dois sinais iguais consecutivos, e não um único sinal igual (como o único sinal de igualdade já é usado como operador de atribuição).

Para os operandos numéricos, o uso desses operadores é direto. Por exemplo, 3 < 5 significa "3 é menor que 5", que é, conceitualmente, uma expressão verdadeira. Em MATLAB, como em muitas linguagens de programação, "true" (verdadeiro) é representado pelo valor lógico 1 e "false" (falso) é representado pelo valor lógico 0. Assim, a expressão 3 < 5 realmente exibe na Janela de Comandos o valor 1 (lógico) em MATLAB. Mostrar o resultado de expressões assim na Janela de Comandos demonstra os valores das expressões.

```
>> 3 < 5
ans =
     1
>> 2 > 9
ans =
     0
>> class(ans)
ans =
lógical
```

O tipo de resultado é **logical**, não **double**. MATLAB também possui **true** e **false** incorporados. Em outras palavras, o verdadeiro é equivalente a **logical(1)** e o falso é equivalente a **logical(0)**. (Em algumas versões do MATLAB, o valor mostrado para o resultado dessas expressões é **true** ou **false** na Janela de Espaço de Trabalho). Embora estes sejam valores lógicos, as operações matemáticas podem ser realizadas com 1 ou 0 resultantes.

```
>> 5 <7
ans =
     1
>> ans + 3
ans =
     4
```

A comparação de caracteres (por exemplo, 'a' < 'c') também é possível. Os caracteres são comparados usando seus valores equivalentes na codificação de caracteres ASCII. Então, 'a' < 'c' é uma expressão verdadeira porque o caractere 'a' vem antes do personagem 'c'.

```
>> 'a' < 'c'
ans =
    1
```

Os operadores lógicos são:

Operador	Significado
||	ou
&&	e
~	não

Todos os operadores lógicos operam sobre operandos **lógicos** (**logical**) ou **booleanos**. O operador **não** é um operador unário; os outros são binários. O operador **não** terá uma expressão lógica (**logical**), que é **true** (verdadeira) ou **false** (falsa), e dará o valor oposto. Por exemplo, ~(3 < 5) é falso porque (3 < 5) é verdadeiro. O operador **ou** tem duas expressões lógicas como operandos. O resultado é verdadeiro se um ou ambos os operandos forem verdadeiros, e falso, somente se ambos os operandos forem falsos. O operador **e** também opera sobre dois operandos lógicos. O resultado de uma e expressão é verdadeiro somente se ambos os operandos forem verdadeiros; e é falso se um ou ambos são falsos. Os operadores ou/e mostrados aqui são usados para **escalares** ou valores únicos. Outros operadores ou/e serão explicados no Capítulo 2.

Os operadores || e && em MATLAB são exemplos de operadores que são conhecidos como operadores de ***curto-circuito***. Isso significa que se o resultado da expressão puder ser determinado com base na primeira parte, a segunda parte também não será avaliada. Por exemplo, na expressão:

```
2 < 4 || 'a' == 'c'
```

A primeira parte, 2 < 4, é verdadeira, assim, toda a expressão é verdadeira; A segunda parte, 'a' == 'c', não seria avaliada.

Além desses operadores lógicos, o MATLAB também possui uma função **xor**, que é a função **ou exclusivo**. Ele retorna **verdadeiro lógico** se um (e apenas um) dos argumentos é **verdadeiro**. Por exemplo, na expressão seguinte, apenas o primeiro argumento é **verdadeiro**, então o resultado é **verdadeiro**:

```
>> xor(3 < 5, 'a' > 'c')
ans =
    1
```

Neste exemplo, ambos os argumentos são **verdadeiros**, assim o resultado é **falso**:

```
>> xor(3 < 5, 'a' < 'c')
ans =
    0
```

Dado os valores **lógicos** de **verdadeiro** e **falso** para as variáveis *x* e *y*, a ***tabela verdade*** (ver Tabela 1.1) mostra como os operadores lógicos funcionam para todas as combinações. Observe que os operadores lógicos são ***comutativos*** (por exemplo, *x* || *y* é o mesmo que *y* || *x*).

Tabela 1.1 Tabela de Verdade para Operadores Lógicos					
x	y	~x	x || y	x && y	xor (x, y)
verdadeiro	verdadeiro	falso	verdadeiro	verdadeiro	falso
verdadeiro	falso	falso	verdadeiro	falso	verdadeiro
falso	falso	verdadeiro	falso	falso	falso

Tal como acontece com os operadores numéricos, é importante conhecer as regras de precedência do operador. A Tabela 1.2 mostra as regras para os operadores que foram vistos até agora em ordem de precedência.

Tabela 1.2 Regras de precedência do operador	
Operadores	Precedência
parênteses () potência ^ negação -, não ~ (unários) multiplicação, divisão *, /, \ adição, subtração +, - relacional <, <=, >, >=, ==, ~ = e && ou || atribuição =	maior menor

PERGUNTA RÁPIDA!

Suponha que exista uma variável *x* que tenha sido inicializada. Qual seria o valor da expressão

```
3 < x < 5
```

se o valor de *x* for 4? E se o valor de *x* for 7?

Resposta

O valor desta expressão sempre será **verdadeiro lógico**, ou 1, independentemente do valor da variável *x*. As expressões são avaliadas da esquerda para a direita. Então, primeiro a expressão 3 < *x* será avaliada. Existem apenas duas possibilidades: ela será verdadeira ou falsa, o que significa que a expressão terá um valor de 1 ou 0. Então, o restante da expressão que será avaliada, será 1 < 5 ou 0 < 5. Essas duas expressões são **verdadeiras**. Assim, o valor de *x* não importa: a expressão 3 < x < 5 seria **verdadeira** independentemente do valor da variável *x*. Este é um erro lógico; não exigiria o alcance desejado. Se queríamos uma expressão com o valor **verdadeiro lógico**, somente se *x* estivesse no intervalo de 3 a 5, poderíamos escrever 3 < x && x < 5 (note que os parênteses não são necessários).

PRÁTICA 1.6

Pense sobre o que seria produzido pelas seguintes expressões e, em seguida, digite-as para verificar suas respostas.

```
3 == 5 + 2
'b' < 'a' + 1
10 > 5 + 2
(10 > 5) + 2
'c' == 'd' - 1 && 2 < 4
'c' == 'd' - 1 || 2 > 4
xor('c' == 'd' - 1, 2 > 4)
xor('c' == 'd' - 1, 2 < 4)
10 > 5 > 2
```

- Explore outras características interessantes

  Esta seção lista alguns recursos e funções no MATLAB, relacionado ao que foi explicado neste capítulo, que você pode desejar explorar por conta própria.

  - Janela de Espaço de Trabalho: há muitos outros aspectos da Janela de Espaço de Trabalho para explorar. Para tentar isso, crie algumas variáveis. Faça a Janela de Espaço de Trabalho a janela ativa clicando com o mouse nela. A partir daí, você pode escolher quais os atributos das variáveis a serem visíveis, escolhendo Escolher Colunas no menu. Além disso, se você clicar duas vezes em uma variável na Janela de Espaço de Trabalho, exibirá uma janela do Editor de Variáveis que permitirá a você modificar a variável.
  - Clique no *fx* ao lado do *prompt* na Janela de Comandos e, sob MATLAB, escolha Matemática, em seguida, Matemática Elementar, em seguida, Expoentes e Logaritmos para ver mais funções nessa categoria.
  - Use **help** para aprender sobre a função **path** e funções de diretório relacionadas.
  - A função **pow2**.
  - Funções relacionadas à moldagem de tipos: **cast**, **typecast**.
  - Encontre a precisão da representação de ponto flutuante para os tipos **single** e **double** usando a função **eps**.

Resumo

Armadilhas Comuns

É comum ao aprender a programar, cometer erros simples de ortografia e confundir a pontuação necessária. Aqui são fornecidos exemplos com erros muito comuns. Alguns destes incluem:

- Colocar um espaço em um nome de variável
- Confundir o formato de uma declaração de atribuição como

  ```
  expressão = nomedavariável
  ```

  ao invés de

  ```
  nomedavariável = expressão
  ```

  O nome da variável deve estar sempre à esquerda

- Usar um nome de função interna como um nome de variável e, em seguida, tentar usar a função
- Confundir os dois operadores de divisão / e \
- Esquecer as regras de precedência do operador
- Confundir a ordem dos argumentos passados para funções; por exemplo, para encontrar o resto da divisão de 3 por 10 usando **rem (3, 10)** em vez de **rem (10, 3)**
- Não usar diferentes tipos de argumentos ao testar funções
- Esquecer de usar parênteses para passar um argumento para uma função (por exemplo, "fix 2.3" em vez de "**fix(2.3)**") – MATLAB retorna o equivalente ASCII para cada caractere quando este erro é cometido (o que acontece é que ele é interpretado como a função de uma string, "**fix ('2.3')**")
- Confundir && e ||
- Confundir || e **xor**

- Colocar um espaço em operadores de dois caracteres (por exemplo, digitando "< =" em vez de "<=")
- Usar = em vez de == para igualdade.

Diretrizes de estilo de programação

Seguir estas diretrizes tornará seu código muito mais fácil de ler e entender e, portanto, mais fácil de trabalhar e modificar.

- Use nomes de variáveis mnemônicas (nomes que fazem sentido, por exemplo, *raio* em vez de *xyz*).
- Embora as variáveis denominadas resultado e RESULTADO sejam diferentes, evite isso, pois seria confuso.
- Não use nomes de funções internas como nomes de variáveis.
- Guarde os resultados em variáveis nomeadas (em vez de usar *ans*) se elas forem ser usadas mais tarde.
- Certifique-se de que os nomes das variáveis tenham menos caracteres que **namelengthmax**.
- Se forem desejados diferentes conjuntos de números randômicos, configure a semente para as funções randômicas usando **rng**.

Funções e Comandos do MATLAB			
demo	int64	fix	asinh
help	uint8	floor	sind
lookfor	uint16	ceil	asind
doc	uint32	round	pi
quit	uint64	mod	i
exit	intmin	rem	j
namelengthmax	intmax	sign	inf
who	char	sqrt	NaN
whos	logical	nthroot	rand
clear	true	log	rng
single	false	log2	clock
double	class	log10	randn
int8	format	exp	randi
int16	sin	asin	xor
int32	abs	sinh	

Operadores do MATLAB			
atribuição =	multiplicação *	maior que >	desigualdade ~=
reticências ou continuação ...	dividido por /	menor que <	ou para scalares ||
adição +	dividido em \	maior ou igual a >=	e para escalares &&
negação -	exponenciação ^	menor ou igual a <=	não ~
subtração -	parênteses ( )	igualdade ==	

Exercícios

1. Crie uma variável para armazenar o peso atômico do cobre (63.55).

2. Crie uma variável *minhaidade* e armazene a sua idade nela. Subtraia dois do valor da variável. Adicione um ao valor da variável. Observe a Janela de Espaço de Trabalho e a Janela do Histórico de Comando enquanto você faz isso.

3. Use a função interna **namelengthmax** para descobrir o número máximo de caracteres que você pode ter em um nome de identificador, sob sua versão do MATLAB.

4. Crie duas variáveis para armazenar um peso em libras e em onças. Use **who** e **whos** para ver as variáveis. Apague uma delas e depois use **who** e **whos** novamente.

5. Use **intmin** e **intmax** para determinar o intervalo dos valores que podem ser armazenados nos tipos **uint32** e **uint64**.

6. Armazene um número com uma casa decimal em uma variável **double** (o padrão). Converta a variável para o tipo **int32** e armazene o resultado em uma nova variável.

7. Crie uma tabela (em um processador de texto ou planilha, não em MATLAB) mostrando o intervalo para todos os tipos inteiros. Calcule os valores mínimo e máximo, e use as funções **intmin** e **intmax** para verificar seus resultados.

8. Explore o comando **format** com mais detalhes. Use o **help format** para encontrar opções. Experimente **format bank** para exibir valores em dólares.

9. Encontre uma opção de **format** que resultaria no seguinte formato de saída:

```
>> 5 / 16 + 2 / 7
ans =
   67/112
```

10. Pense nos resultados para as seguintes expressões e digite-as para verificar suas respostas.

```
25 / 5 * 5
4 + 3 ^ 2
(4 + 3) ^ 2
3 \ 12 + 5
4 - 2 * 3
```

    À medida que o mundo se torna mais "*plano*", é cada vez mais importante que engenheiros e cientistas possam trabalhar com colegas em outras partes do mundo. A conversão correta de dados de um sistema de unidades para outro (por exemplo, do sistema métrico para o sistema dos EUA ou vice-versa) é extremamente importante.

11. Crie uma variável *libra* para armazenar um peso em *libras*. Converta isso em *quilogramas* e atribua o resultado a uma variável *quilograma*. O fator de conversão é 1 quilograma = 2,2 libras.

12. Crie uma variável *tempf* para armazenar uma temperatura em graus Fahrenheit (°F). Converta isso em graus Celsius (°C) e armazene o resultado em uma outra variável. O fator de conversão é °C = (°F - 32) * 5/9.

13. Use outra quantidade para ser convertida de um sistema de unidades para outro.

14. A função **sin** calcula e retorna o seno de um ângulo em radianos, e a função **sind** retorna o seno de um ângulo em graus. Verifique se chamando a função **sind**, passando 90 graus para ela, resulta em 1. Que argumento você passaria para **sin** para obter o resultado 1?

15. A resistência combinada em paralelo (R_T) de três resistores (R_1, R_2 e R_3) é dada por

$$R_T = \frac{1}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3}}$$

Crie variáveis para os três resistores, armazene valores em cada uma e depois calcule a resistência combinada.

16. Use **help elfun** ou faça experiências para responder as seguintes perguntas:

    - **fix(3.5)** é o mesmo que o **floor(3.5)**?
    - **fix(3.4)** é o mesmo que **fix(-3.4)**?
    - **fix(3.2)** é o mesmo que o **floor(3.2)**?
    - **fix(-3.2)** é o mesmo que o **floor(-3.2)**?
    - **fix(-3.2)** é o mesmo que **ceil(-3.2)**?

17. Para que intervalo de valores a função **round** é equivalente à função **floor**?
    Para que intervalo de valores a função **round** é equivalente à função **ceil**?

18. Use **help** para determinar a diferença entre as funções **rem** e **mod**.

19. Escreva expressões MATLAB para o seguinte:

    $\sqrt{19}$
    3^{12}
    `tan(π)`

20. Gere um número randômico:

    - real no intervalo (0, 20)
    - real no intervalo (20, 50)
    - inteiro no intervalo inclusivo de 1 a 10
    - inteiro no intervalo inclusivo de 0 a 10
    - inteiro no intervalo inclusivo de 50 a 100.

21. Entre em uma nova Janela de Comandos e digite **rand** para obter um número real randômico. Anote o número. Então, saia do MATLAB e repita isso, novamente tomando nota do número randômico; deve ser o mesmo que antes. Finalmente, saia do MATLAB e volte a entrar em uma nova Janela de Comandos. Desta vez, mude a semente antes de gerar um número randômico; deve ser diferente.

22. Na codificação de caracteres ASCII, as letras do alfabeto são pela ordem: 'a' vem antes de 'b' e também 'A' vem antes de 'B'. No entanto, o que vem primeiro – letras minúsculas ou maiúsculas?

23. Desloque a string 'xyz' para frente, na codificação de caracteres, em dois caracteres.

24. Quais seriam o resultado das seguintes expressões?

```
'b' >= 'c' - 1
3 == 2 + 1
(3 == 2) + 1
xor(5 < 6, 8 > 4)
```

25. Crie duas variáveis *x* e *y* e armazene números nelas. Escreva uma expressão que seja verdadeira se o valor de *x* for maior que 5 ou se o valor de *y* for inferior a 10, mas não se ambas forem verdadeiras.

26. Use o operador de igualdade para verificar que 3*10^5 é igual a 3e5.

27. Use o operador de igualdade para verificar o valor de log10 (10000).

28. Existem funções equivalentes para **intmin** e **intmax** para tipos de números reais? Use **help** para descobrir.

29. Um vetor pode ser representado pelas suas coordenadas retangulares x e y ou pelas suas coordenadas polares r e Θ. A relação entre eles é dada pelas equações:

    x = r * cos(Θ)
    y = r * sin(Θ)

    Atribua valores para as coordenadas polares, as variáveis r e teta. Em seguida, usando esses valores, atribua para as coordenadas retangulares correspondentes, as variáveis x e y.

30. Na relatividade especial, o fator Lorentz é um número que descreve o efeito da velocidade em várias propriedades físicas quando a velocidade é significativa em relação à velocidade da luz. Matematicamente, o fator de Lorentz é dado como:

$$\gamma = \frac{1}{\sqrt{1 - \frac{v^2}{c^2}}}$$

    Use 3 x 10^8 m/s para a velocidade da luz, *c*. Crie variáveis para *c* e para a velocidade *v* e, a partir delas, uma variável *lorentz* para o fator Lorentz.

31. Uma empresa fabrica uma peça para a qual existe um peso desejado. Existe uma tolerância de N por cento, o que significa que o intervalo entre menos e mais, N% do peso desejado é aceitável. Crie uma variável que armazene um peso e outra variável para N (por exemplo, atribua dois para ela). Crie variáveis que armazenem os valores mínimos e máximos no intervalo de pesos aceitáveis para esta peça.

32. Um engenheiro ambiental determinou que o custo C de um reservatório de contenção será baseado no raio *r* do tanque:

$$\mathrm{C} = \frac{32430}{r} + 428\pi r$$

    Crie uma variável para o raio e, em seguida, uma para o custo.

33. Uma planta química libera uma quantidade A de poluente em uma corrente. A concentração máxima C do poluente num ponto que está uma distância x da planta é:

$$\mathrm{C} = \frac{A}{x}\sqrt{\frac{2}{\pi e}}$$

Crie variáveis para os valores de A e x e, em seguida, para C. Suponha que a distância x esteja em metros. Experimente valores diferentes para x.

34. A média geométrica g de n números x_i é definida como a enésima raiz do produto de x_i:

$$g = \sqrt[n]{X_1 X_2 X_3 \ldots X_n}$$

(Isso é útil, por exemplo, para encontrar a taxa média de retorno para um investimento, que você faria em engenharia econômica). Se um investimento retornar 15% no primeiro ano, 50% no segundo e 30% no terceiro ano, a taxa média de retorno seria $(1{,}15 * 1{,}50 * 1{,}30)^{1/3}$. Calcule isso.

Referência

ATAWAY, S. MATLAB A Pratical Introduction to Programming and Program Solving. Butterworth-Heinemann/Elsevier, Waltham, MA, USA, Third Edition, 2013.