CAPÍTULO 7

Programando no MATLAB

Programas de computador são uma seqüência de comandos devidamente ordenados que visam à resolução de um problema. Em um programa elementar, os comandos são executados um após o outro, na ordem em que são digitados. Todos os programas apresentados nos capítulos sobre rotinas e funções são exemplos de programas elementares. Entretanto, em muitas situações, é freqüentemente necessário programas mais sofisticados, onde os comandos não são executados necessariamente na ordem em que aparecem na seqüência lógica quando o programa é inicializado com valores diferentes para as variáveis de entrada. Por exemplo, um programa de computador que determine o custo de postagem de pacotes nos Correios e utilize expressões matemáticas diferentes para calcular o custo final, baseadas no peso e no tamanho do pacote, nos itens a serem enviados (o envio de CDs é mais barato do que o de livros, etc.) e o tipo de serviço de transporte disponibilizado (terrestre, marítimo ou aéreo). Em outras situações, pode ser necessário repetir uma seqüência de comandos várias vezes dentro de um programa. Por exemplo, um programa que resolve equações numericamente repete uma seqüência de cálculos até que o erro na resposta seja menor que alguma medida ou parâmetro especificado.

O MATLAB dispõe de ferramentas que podem ser utilizadas para controlar o fluxo de um programa. As sentenças condicionais (Seção 7.2) e a estrutura `switch... case` (Seção 7.3) tornam possível selecionar comandos ou executar grupos específicos de comandos em diferentes situações. Laços `for` ou `while` (Seção 7.4) proporcionam a repetição sistemática de comandos.

Evidentemente, o redirecionamento do fluxo de um programa requer alguma espécie de estrutura de decisão dentro do programa. O programa deve decidir se executa o próximo comando ou se seleciona um ou mais comandos para continuar em uma linha de programa diferente. O programa toma essas decisões comparando os valores assumidos pelas variáveis. Um modo de fazê-lo é usar operadores lógicos e relacionais, que serão explicados na Seção 7.1.

Para finalizar, deve ser mencionado que funções (Capítulo 6) também podem ser utilizadas na programação. Uma função em si é um subprograma. Quando um programa principal encontra uma linha de comando contendo uma função, essa função é chamada, enquanto o programa principal aguarda o retorno do(s) resultado(s) dessa função. Terminada a execução da função, os cálculos são encerrados, os resultados são retornados para o programa que chamou a função e o programa principal segue para a próxima linha de comando.

7.1 OPERADORES LÓGICOS E RELACIONAIS

O operador relacional compara dois números determinando se o resultado da sentença de comparação é verdadeiro (V) ou falso (F) (p. ex.: $5 < 8 \Rightarrow$ V). Se a sentença for verdadeira o valor retornado é 1. Caso contrário, o valor retornado é 0. Um operador lógico examina sentenças verdadeiras/falsas e produz resultados verdadeiro (1) ou falso (0), de acordo com a funcionalidade do operador. Por exemplo, o operador lógico AND resulta 1 (verdadeiro) se, e somente se, todas as sentenças envolvidas na operação forem verdadeiras (1s). Tanto operadores lógicos quanto relacionais podem ser utilizados em expressões matemáticas ou, como será visto adiante, serem combinados a outros comandos para controlar ou tomar decisão sobre o fluxo do programa.

Operadores relacionais:

Os operadores relacionais no MATLAB são:

Operador relacional	Descrição
<	Menor que
>	Maior que
<=	Menor que ou igual a
>=	Maior que ou igual a
= =	Igual a
~ =	Diferente de

Perceba que o operador relacional que testa a igualdade entre dois objetos é representado por dois sinais de igualdade (= =), sem espaço entre eles. Isso porque um único sinal de igualdade representa o operador de atribuição. Os demais operadores duplos (representados por dois caracteres) também não possuem espaços entre os caracteres (<=, >=, ~ =).

- Operadores relacionais são utilizados juntamente com operadores aritméticos dentro de expressões matemáticas. O resultado pode ser utilizado em outras operações matemáticas, no endereçamento de arranjos ou para controlar o fluxo do programa no MATLAB (juntamente com estruturas de tomada de decisão, p. ex.: `if`).
- Quando dois números são comparados, o resultado será 1 (verdadeiro) se, de acordo com o operador relacional, a comparação for verdadeira, e 0 (falso) se a comparação for falsa.

- Se dois escalares estão sendo comparados, o resultado é o escalar 1 ou 0. Caso estejamos comparando arranjos (vetores, matrizes 2D, 3D, etc.), a comparação é feita *elemento por elemento* do arranjo e o resultado é um arranjo lógico de 1s e 0s, cuja dimensão é a mesma dos arranjos originais.
- Se um escalar estiver sendo comparado com um arranjo, o escalar será comparado com cada elemento do arranjo e o resultado é um arranjo lógico de 1s e 0s, de acordo com a posição de cada elemento no arranjo.

Alguns exemplos:

```
>> 5 > 8
```
Verifica se 5 é maior que 8.

```
ans =

0
```
Como a comparação é falsa (5 é menor que 8), a resposta é 0.

```
>> a=5 < 10
```
Verifica se 5 é menor que 10 e atribui a resposta à variável a.

```
a =

   1
```
Como a comparação é verdadeira (5 é menor que 10), a resposta 1 é atribuída à variável a.

```
>> y=(6 < 10) + (7 > 8) + (5*3 == 60/4)
```
Igual a 1 porque 6 é menor que 10. Igual a 0 porque 7 é menor que 8. Igual a 1 porque 5*3 é igual a 60/4.

```
y =

   2

>> b = [15 6 9 4 11 7 14]; c = [8 20 9 2 19 7 10];
```
Declara os vetores b e c.

```
>> d=c>=b
```
Verifica quais elementos de c são maiores do que ou iguais aos elementos de b.

```
d =

   0   1   1   0   1   1   0
```
Atribui 1 onde um elemento de c é maior do que ou igual a um elemento de b.

```
>> b == c
```
Verifica quais dos elementos de c são iguais aos elementos de b.

```
ans =

   0   0   1   0   0   1   0

>> b~=c
```
Verifica quais dos elementos de c são diferentes dos elementos de b.

```
ans =

   1   1   0   1   1   0   1

>> f = b-c>0
```
Subtrai c de b e então verifica quais elementos são maiores que zero.

```
f =
   1   0   0   1   0   0   1
>> A = [2 9 4; -3 5 2; 6 7 -1]
A =
    2   9   4
   –3   5   2
    6   7  –1
>> B = A <=2
B =
    1   0   0
    1   0   1
    0   0   1
```

Declara uma matriz A de dimensão 3 × 3.

Verifica quais elementos em A são menores ou iguais a 2. Atribui os resultados à matriz B.

- Os resultados de uma operação relacional entre vetores formam um vetor de 0s e 1s, denominados vetor lógico, que podem ser utilizados no endereçamento de outros vetores. Quando um vetor lógico é utilizado para tais finalidades, extrai do vetor endereçado os elementos nas posições onde o vetor lógico tem 1s. Por exemplo:

```
>> r =[8 12 9 4 23 19 10]
r =
    8   12  9   4   23  19  10
>> s = r<= 10
s =
    1   0   1   1   0   0   1
>> t = r(s)
t =
    8   9   4   10
>> w = r(r <= 10)
w =
    8   9   4   10
```

Declara um vetor r.

Verifica quais elementos de r são menores ou iguais a 10.

O resultado é um vetor lógico s com 1s nas posições onde os elementos de r são menores ou iguais a 10.

Usa s para endereçar um vetor r e criar o vetor t.

O vetor t é formado pelos elementos de r nas posições onde s tem 1s.

O mesmo pode ser feito em uma única etapa.

- Os vetores e arranjos numéricos de 0s e 1s não funcionam como vetores e arranjos lógicos de 0s e 1s. Arranjos numéricos não podem ser utilizados para endereçar outros arranjos. Por outro lado, os arranjos lógicos podem ser utilizados em operações aritméticas. Uma vez utilizado em operações aritméticas, o arranjo lógico torna-se numérico.

- Ordem de precedência: em uma expressão matemática que inclua operadores lógicos e aritméticos, as operações aritméticas (+, −, *, /, \) precedem todas as operações relacionais. Os operadores relacionais têm, entre eles, igual precedência e são avaliados da esquerda para a direita. Muitas vezes, parênteses são utilizados para modificar a ordem de precedência. Exemplos:

```
>> 3 + 4 < 16/2                    As operações + e / são executadas primeiramente.
ans =                              A resposta é 1 porque 7 < 8 é verdadeiro.
     1
>> 3 + (4 < 16)/2                  4 < 16 é executado primeiro, o que resulta em 1, porque é verdadeiro.
ans =                              3.5 é obtido de 3 + 1/2.
    3.5000
```

Operadores lógicos:

Os operadores lógicos do MATLAB são:

Operador lógico	Nome	Descrição
& Exemplo: A&B	AND	Age em dois operandos (A, B). Se ambos forem verdadeiros, o resultado será verdadeiro (1). De outro modo, o resultado será falso 0.
| Exemplo: A|B	OR	Age em dois operandos (A, B). Se um dos operandos for verdadeiro (em particular os dois), o resultado será verdadeiro (1). De outro modo (ou seja, ambos falsos), o resultado será falso (0).
~ Exemplo: ~A	NOT	Age em um operando (A). Resulta na negação do operando: verdadeiro (1), se o operando for falso, e falso (0), se o operando for verdadeiro.

- Operadores lógicos recebem números como operandos. Qualquer número diferente de zero é verdadeiro e apenas o número zero é falso.
- Operadores lógicos (como os relacionais) podem ser usados juntamente com operadores aritméticos dentro de expressões matemáticas. O resultado pode ser utilizado em outras operações matemáticas, no endereçamento de arranjos ou para controlar o fluxo do programa no MATLAB (juntamente com estruturas de tomada de decisão, p. ex.: `if`).
- Operadores lógicos (como os relacionais) agem tanto em escalares quanto em arranjos genéricos.
- As operações lógicas AND e OR podem agir em escalares puros, arranjos puros ou numa miscelânea, i.e., entre um escalar e um arranjo. Se ambos os objetos são es-

calares, o resultado será um escalar 0 ou 1. Se ambos são arranjos, eles devem possuir a mesma dimensão e a operação lógica será executada *elemento a elemento*. O resultado será um arranjo lógico de mesma dimensão com 0s e 1s de acordo com a posição que os elementos originais ocupam no *array*. Caso um operando seja um escalar e o outro seja um vetor, a operação lógica é feita entre o escalar e cada um dos elementos no arranjo. O resultado é um arranjo lógico de 0s e 1s, de mesma dimensão do arranjo que entra na operação.

- A operação lógica NOT possui apenas um operando. Quando utilizada em um escalar, resulta no escalar 0 ou 1. Se utilizada em arranjos, cada elemento do arranjo será negado, i.e., os elementos falsos serão tornados verdadeiros e os elementos verdadeiros (diferentes de zero) serão tornados falsos.

Eis alguns exemplos:

```
>> 3&7                                          3 AND 7.

ans =                                          Tanto 3 quanto 7 são verdadeiros (diferentes de zero), então o resultado
                                               é verdadeiro (1).
   1

>> a = 5|0                                     5 OR 0 (atribuído à variável a).

a =                                            O valor 1 é atribuído a a, visto que um dos operandos é verdadeiro (di-
                                               ferente de zero).
   1

>> – 25                                        NOT 25.

ans =                                          O resultado é 0, visto que 25 é verdadeiro (diferente de
                                               zero). Negando, resulta em falso.
   0

>> t = 25*((12&0) + (~0) + (0|5))              Usando operadores lógicos em expressões ma-
                                               temáticas.
t =

   50

>> x = [9  3  0  11  0  15]; y = [2  0  13 –11  0  4];   Declarando os vetores x e y.

>> x&y                                         O resultado é um vetor com 1s nas posições, onde x
                                               e y são verdadeiros simultaneamente (diferentes de
ans =                                          zero), e 0s de outro modo.

1 0 0 1 0 1

>> z = x|y                                     O resultado é um vetor com 1s, nas posições, onde x
                                               ou y são verdadeiros (diferentes de zero), e 0s de ou-
z =                                            tro modo.

1 1 1 1 0 1

>> ~(x + y)                                    O resultado é um vetor com 1s, nas posições onde x
                                               ou y são verdadeiros (diferentes de zero), e 0s de ou-
ans =                                          tro modo.

0 0 0 1 1 0
```

Ordem de precedência:

Operadores aritméticos, lógicos e relacionais podem ser utilizados simultaneamente em expressões matemáticas. O resultado da expressão depende do modo como eles estão organizados. O MATLAB utiliza a ordem de precedência que segue:

Precedência	Operação
1 (Mais alta)	Parênteses (se existirem parênteses aninhados, os internos têm precedência mais alta)
2	Exponenciação
3	Lógica NOT (~)
4	Multiplicação e divisão
5	Adição e subtração
6	Operadores relacionais (>,<,>=,<=,= =,~ =)
7	Lógica AND (&)
8 (Mais baixa)	Lógica OR (|)

Se duas ou mais operações possuem a mesma ordem de precedência, a operação mais à esquerda é executada primeiro, depois as demais vão sendo resolvidas uma a uma, até encontrar a operação mais à direita.

Deve ser mencionado aqui que essa ordem foi adotada no MATLAB a partir da versão 6. Versões anteriores do MATLAB têm uma ordem de precedência ligeiramente diferente. Nelas, o operador & não precede |. O usuário deve estar atento a isso. Os problemas de compatibilidade entre versões do MATLAB podem ser resolvidos facilmente com o uso sistemático de parênteses (até mesmo quando eles não são requeridos).

Os exemplos de expressões a seguir ilustram o uso de operadores aritméticos, lógicos e relacionais:

```
>> x = -2; y = 5;
```

Declarando os escalares x e y.

```
>> -5 < x < -1
ans =
0
```

Esta desigualdade está matematicamente correta. Porém, o resultado é falso porque o MATLAB executa-a da esquerda para a direita. −5 < x é verdadeiro, logo 1 < −1 é falso (=0).

```
>> -5 < x & x < -1
ans =
1
```

O resultado matematicamente correto é obtido usando-se o operador lógico &. As desigualdades são executadas em primeiro lugar. Sendo ambas verdadeiras (1), a resposta é 1.

```
>> ~(y < 7)
ans =
0
```

y < 7 é executado primeiro (resultando em verdadeiro). Logo ~1 é falso (=0).

```
>> ~y < 7
ans =
1
```

~y é executado primeiro, resultando em verdadeiro (1) (visto que y é diferente de zero). Logo ~1 é falso (=0), então 0 < 7 é verdadeiro (=1).

```
>> ~((y >= 8)|(x < -1))
ans =
0
```

y >= 8 (falso) e x < –1 (verdadeiro) são executados primeiro. Em seguida, a operação OR é executada e resulta em verdadeiro. Por fim, ~ é executado e resulta em falso (0).

```
>> ~(y >= 8)| (x < -1)
ans =
1
```

y >= 8 (falso) e x < –1 (verdadeiro) são executados primeiro. Em seguida, a operação NOT de (y > =8) é executada e resulta em verdadeiro. Por fim, OR é executado e resulta em verdadeiro (1).

Funções lógicas nativas do MATLAB:

O MATLAB possui um conjunto de funções nativas que se eqüivalem aos operadores lógicos. A seguir, temos uma descrição de algumas dessas funções:

`and(A,B)`	equivalente a `A&B`
`or(A,B)`	equivalente a `A|B`
`not(A)`	equivalente a `~A`

O MATLAB ainda possui as seguintes funções:

Função	Descrição	Exemplo
`xor (a,b)`	Ou Exclusivo. Retorna verdadeiro (1) se houver desigualdade entre os operandos, i.e., se um for verdadeiro e o outro for falso.	>> xor(7,0) ans = 1 >> xor(7, –5) ans = 0
`all (A)`	Retorna verdadeiro (1) se todos os elementos de um vetor `A` forem verdadeiros (diferentes de zero). Retorna falso (0) se um ou mais elementos forem falsos (0). Se `A` for uma matriz, trata as colunas de `A` como vetores e retorna um vetor de 1s e 0s.	>> A = [6 2 15 9 7 11]; >> all(A) ans = 1 >> B=[6 2 15 9 0 11]; >> all(B) ans = 0 >> C = [6 2 15 9 7 11; 0 2 15 9 0 11] C = 6 2 15 9 7 11 0 2 15 9 0 11 >> all(C) ans = 0 1 1 1 0 1

Função	Descrição	Exemplo
`any (A)`	Retorna verdadeiro (1) se qualquer elemento de `A` for verdadeiro (diferente de zero). Retorna falso (0) se todos os elementos de A forem falsos (zero). Se `A` for uma matriz, trata as colunas de `A` como vetores e retorna um vetor 1s e 0s.	>> A=[6 0 15 0 0 11]; >> any(A) ans = 1 >> B = [0 0 0 0 0 0]; >> any(B) ans = 0
`find(A)`	Se `A` for um vetor, retorna os índices dos elementos diferentes de zero.	>> A=[0 9 4 3 7 0 0 1 8]; >> find(A) ans = 2 3 4 5 8 9
`find(A>d)`	Se `A` for um vetor, retorna o endereço dos elementos que são maiores que `d` (qualquer operador relacional pode ser utilizado).	>> find(A>4) ans = 2 5 9

As quatro operações lógicas `and`, `or`, `xor` e `not` podem ser resumidas por meio da tabela verdade:

ENTRADA		SAÍDA				
A	B	AND A&B	OR A|B	XOR (A,B)	NOT ~A	NOT ~B
falso	falso	falso	falso	falso	verdadeiro	verdadeiro
falso	verdadeiro	falso	verdadeiro	verdadeiro	verdadeiro	falso
verdadeiro	falso	falso	verdadeiro	verdadeiro	falso	verdadeiro
verdadeiro	verdadeiro	verdadeiro	verdadeiro	falso	falso	falso

Problema-exemplo 7-1: Analisando dados meteorológicos

Os dados a seguir foram coletados na capital americana (Washington DC) durante o mês de abril de 2002 e correspondem às temperaturas máximas diárias (em ^{o}F): 58 73 73 53 50 48 56 73 73 66 69 63 74 82 84 91 93 89 91 80 59 69 56 64 63 66 64 74 63 69 (dados da U.S. National Oceanic and Atmospheric Administration). Use operações lógicas e relacionais para determinar:

a) O número de dias em que a temperatura esteve acima de 75^{o}F.

b) O número de dias em que a temperatura esteve entre 65^{o} e 80^{o}F.

c) Os dias do mês em que a temperatura esteve entre 50^{o} e 60^{o}F.

Solução

Na rotina abaixo, as temperaturas foram colocadas em um vetor. Em seguida, operadores lógicos e relacionais foram usados para analisar os dados.

```
T = [58  73  73  53  50  48  56  73  73  66  69  63  74  82  84 ...
     91  93  89  91  80  59  69  56  64  63  66  64  74  63  69];

Tmaior_igual75 = T >= 75;
```

Vetor contendo 1s nas posições onde `T >= 75`.

```
Ndias_Tmaior_igual75 = sum(Tmaior_igual75)
```

Adiciona todos os 1s no vetor `Tmaior_igual75`.

```
Tentre65_80 = (T >=65)&(T <=80);
```

Vetor contendo 1s nas posições onde `T >= 65` e `T <=80`.

```
Ndias_Tentre65_80=sum(Tentre65_80)
```

Adiciona todos os 1s no vetor `Tentre65_80`.

```
datasTentre50_60 = find((T >=50)&(T <=60))
```

A função `find` retorna os endereços dos elementos em `T` cujos valores estão entre 50 e 60.

A rotina (salva como Capitulo7_Exemplo1) executada na linha do *prompt* resulta em:

```
>> Capitulo7_Exemplo1
Ndias_Tmaior_igual75 =
     7

Ndias_Tentre65_80 =
     12

datasTentre50_60 =
     1    4    5    7    21          23
```

A temperatura esteve acima de 75°F em 7 dias do mês.

A temperatura esteve entre 65° e 80°F em 12 dias

Os dias em que a temperatura esteve entre 50° e 60°F.

7.2 SENTENÇAS CONDICIONAIS

As sentenças condicionais são estruturas que permitem ao MATLAB tomar decisões e optar por executar um grupo de comandos que seguem dessa sentença (caso seja verdadeira) ou selecionar um desvio (salto no grupo de comandos), seguindo para a próxima linha de instrução. Em uma sentença condicional é imperativo uma expressão de teste. Se a expressão de teste for verdadeira, o MATLAB executa um grupo de comandos que segue a sentença. Se a expressão for falsa, o programa salta para outro comando ou grupo de comandos. A sintaxe básica de uma sentença condicional é:

`if` expressão condicional consistindo de operadores lógicos e/ou relacionais

Exemplos:

```
if    a < b
if    c >= 5
if    a = = b
if    a ~ = 0
if    (d<h)&(x>7)
if    (x~ =13)|(y<0)
```

Todas as variáveis devem ter sido inicializadas.

- As sentenças condicionais podem fazer parte de um programa, rotina ou função.
- Conforme mostrado a seguir, cada sentença `if` deve ser seguida de um comando `end`.

A sentença `if` é usada geralmente em três estruturas: `if-end`, `if-else-end` e `if-elseif-else-end`, que são os objetos de estudo das próximas seções.

7.2.1 A estrutura `if-end`

A estrutura condicional `if-end` tem a estrutura esquemática apresentada na Figura 7-1. A figura mostra como os comandos devem ser digitados no programa, bem como o fluxograma que, simbolicamente, dita o fluxo ou a seqüência segundo o qual os comandos serão executados. Quando um programa é executado, encontra uma sentença `if`. Se a expressão condicional de teste for verdadeira (1), o programa continuará a executar os comandos listados abaixo da sentença `if`, até encontrar o comando `end`. Se a expressão de teste for falsa (0), o programa saltará o grupo de comandos entre as sentenças `if` e `end`, continuando a execução dos comandos listados abaixo de `end`. Portanto, o programa executará a sentença somente quando o resultado do teste for verdadeiro.

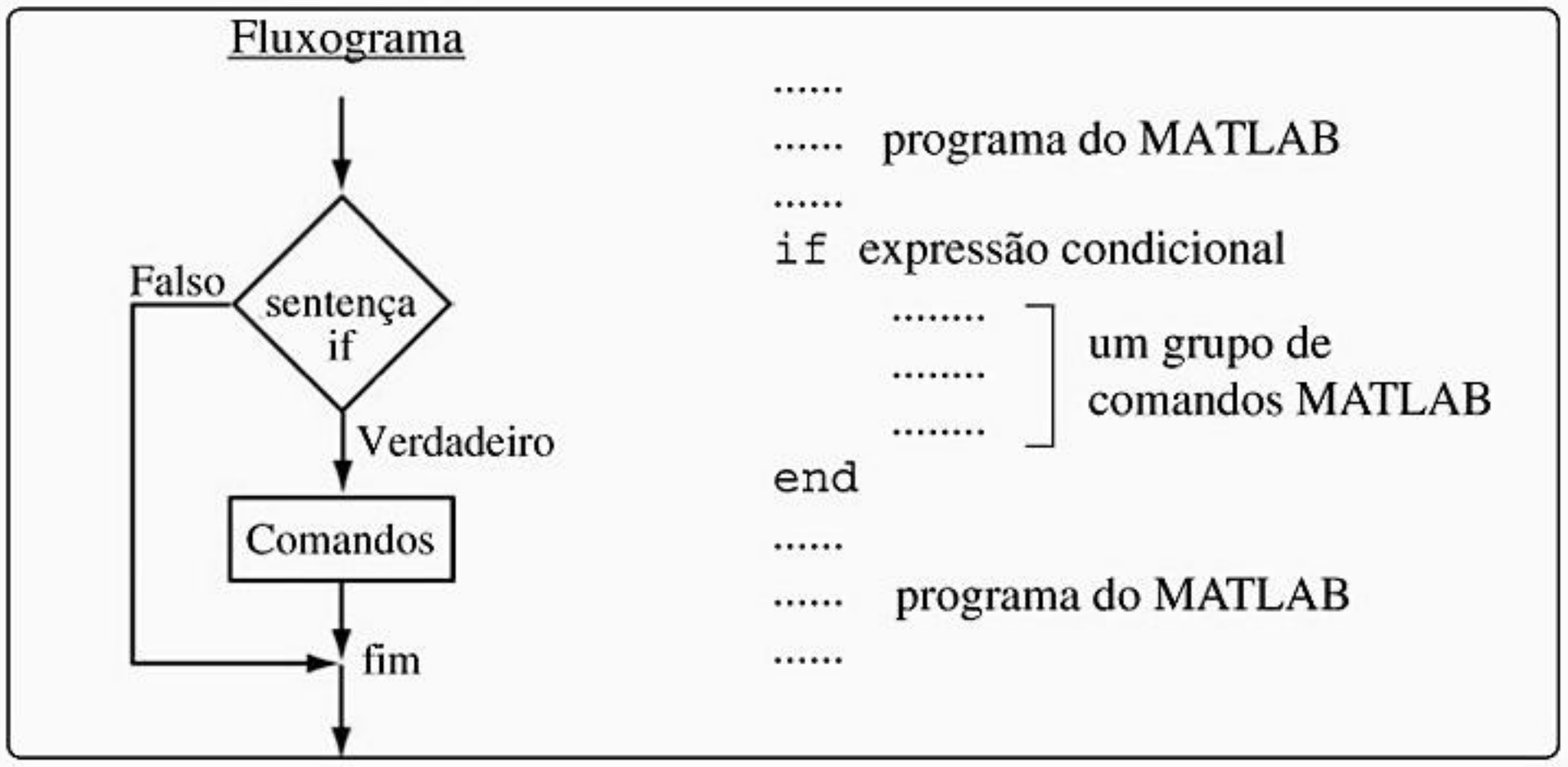


FIGURA 7-1 A estrutura da sentença condicional `if-end`.

As instruções `if` e `end` aparecem na tela em azul e os comandos entre as sentenças `if` e `end` são automaticamente indentados ou alinhados, o que facilita a leitura e a organização do programa. Um exemplo onde a sentença `if-end` é utilizada é mostrado a seguir.

Problema-exemplo 7-2: Calculando o salário de um trabalhador

Um trabalhador é pago de acordo com a jornada semanal de 40 horas, mais 50% sobre as horas extras trabalhadas. Escreva uma rotina que calcule o salário desse trabalhador. O programa deve solicitar ao usuário a quantidade de horas trabalhadas e o valor pago por hora. Por último, o programa deve retornar o salário.

Solução

O programa é mostrado abaixo. Inicialmente, a rotina determina o salário multiplicando o número de horas trabalhadas pela valor da hora. A seguir, uma sentença `if` verifica se o número de horas trabalhadas excede 40 horas semanais. Caso exceda, os comandos internos à sentença calculam o pagamento extra e adicionam o valor ao salário-base. Se não exceder, o programa salta para a sentença `end`.

```
 = input('Por favor, digite o numero de horas trabalhadas: ');
h = input('Por favor, digite o valor da hora trabalhada: $');
Salario = t*h;
if t > 40
   Salario = Salario + (t-40)*0.5*h;
end
fprintf('O salario do trabalhador e R$%5.2f',Salario)
```

A aplicação do programa em dois casos pode ser vista a seguir (o arquivo foi salvo como Capitulo7_Exemplo2).

```
>> Capitulo7_Exemplo2
Por favor, digite o numero de horas trabalhadas: 35
Por favor, digite o valor da hora trabalhada: $5.00
O salario do trabalhador e $175.00
>> Capitulo7_Exemplo2
Por favor, digite o numero de horas trabalhadas: 50
Por favor, digite o valor da hora trabalhada: $10.00
O salario do trabalhador e $550.00
```

7.2.2 A estrutura `if-else-end`

Esta estrutura executa um grupo de comandos, se o resultado do teste for verdadeiro, ou outro grupo, se o resultado for falso. Observe a Figura 7-2.

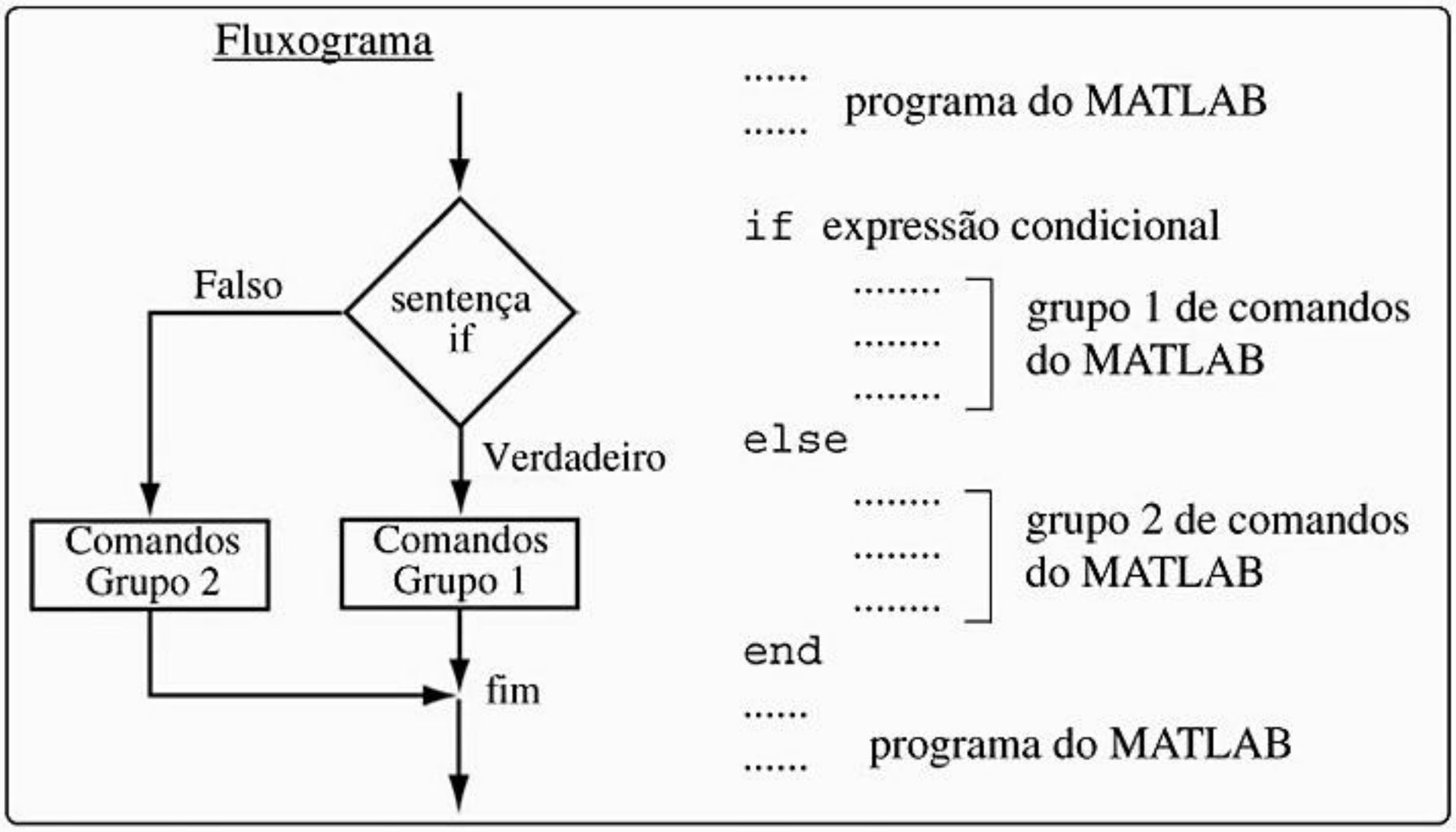


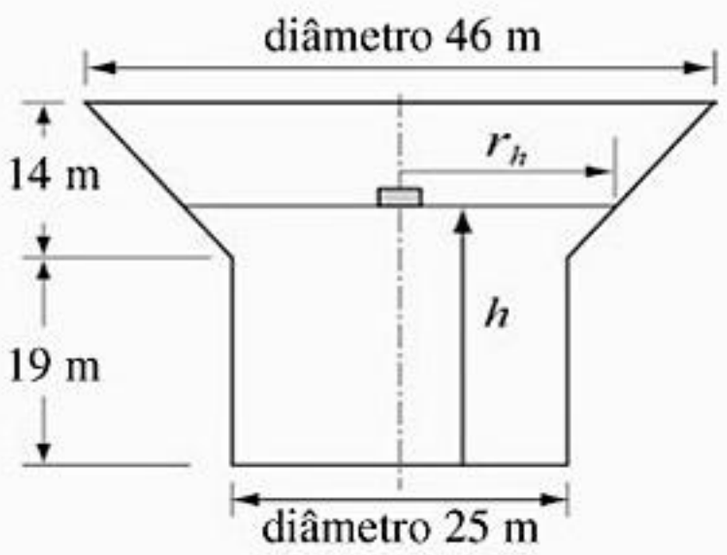
FIGURA 7-2 A estrutura da sentença condicional `if-else-end`.

A figura mostra como os comandos devem ser digitados e o fluxograma ilustra o fluxo ou a seqüência do programa segundo o qual os comandos serão executados. A primeira linha contém uma sentença `if` com a respectiva expressão de teste. Se o resultado da expressão de teste for verdadeiro, o programa executará os comandos do grupo 1 (entre as sentenças `if` e `else`) e saltará para o comando `end`. Se o resultado do teste for falso, o programa saltará para a sentença `else` e executará os comandos do grupo 2 (entre as sentenças `else` e `end`).

O exemplo a seguir ilustra a estrutura condicional `if-else-end`.

Problema-exemplo 7-3: Nível de uma caixa d'água

O tanque de uma caixa d'água possui a geometria mostrada na figura (a base é um cilindro e a parte superior é um cone invertido cortado). Dentro do tanque há uma bóia que indica o nível da água. Escreva uma função que determine o volume da água armazenada no tanque, na posição indicada pela bóia (altura h). A função deve receber a variável h (em m) e retornar o volume da água (em m^3).



Solução

Para $0 \leq h \leq 19$m, o volume do tanque é dado pelo volume de um cilindro de altura $h =$ 19m: $V = (12{,}5)^2 \pi h$.

Para $19 \le h \le 33$m, o volume do tanque é a adição dos volumes do cilindro com $h = 19$m e o volume da água na porção do cone invertido:

$$V = \pi 12.5^2 \cdot 19 + \frac{1}{3}\pi(h-19)(12.5^2 + 12.5 \cdot r_h + r_h^2)$$

onde $r_h = 12.5 + \frac{10.5}{14}(h-19)$.

A função, escrita na janela Editor, foi salva como `v = voldagua (h)`.

```
function v = voldagua(h)
% Voldagua determina o volume de agua numa caixa dagua.
% A entrada e o nivel de agua (em m).
% A saida e o volume de agua numa caixa dagua (em m3).
if h <= 19
   v = pi*12.5^2*h;
else
   rh = 12.5 + 10.5*(h-19)/14;
   v = pi*12.5^2*19 + pi*(h-19)*(12.5^2 + 12.5*rh + rh^2)/3;
end
```

Chamando a função na linha do *prompt* (salva como `voldagua`) para resolver dois casos:

```
>> voldagua(8)
ans =
   3.9270e+003
>> voldagua(25.7)
ans =
   1.4115e+004
```

7.2.3 A estrutura `if-elseif-else-end`

A estrutura de `if-elseif-else-end` está indicada na Figura 7-3. A figura mostra como os comandos devem ser digitados no programa e ilustra o fluxo e/ou a seqüência por meio de um fluxograma segundo o qual os comandos serão executados. Essa estrutura possui duas sentenças de teste (`if` e `elseif`) que tornam possível selecionar um dos três grupos de comandos para a execução. A primeira linha traz a sentença `if` com a expressão de teste. Se o resultado do teste é verdadeiro, o programa executa os comandos do grupo 1 (entre as sentenças `if` e `elseif`) e salta para `end`. Se o resultado do teste da sentença `if` é falso, o programa salta para a sentença `elseif`. Caso a expressão de teste de `elseif` seja verdadeira, o programa executa os comandos do grupo 2 (entre `elseif` e `else`) e salta para `end`. Se a expressão de teste de `elseif` é falsa, o programa salta para `else` e executa os comandos do grupo 3 (entre `else` e `end`).

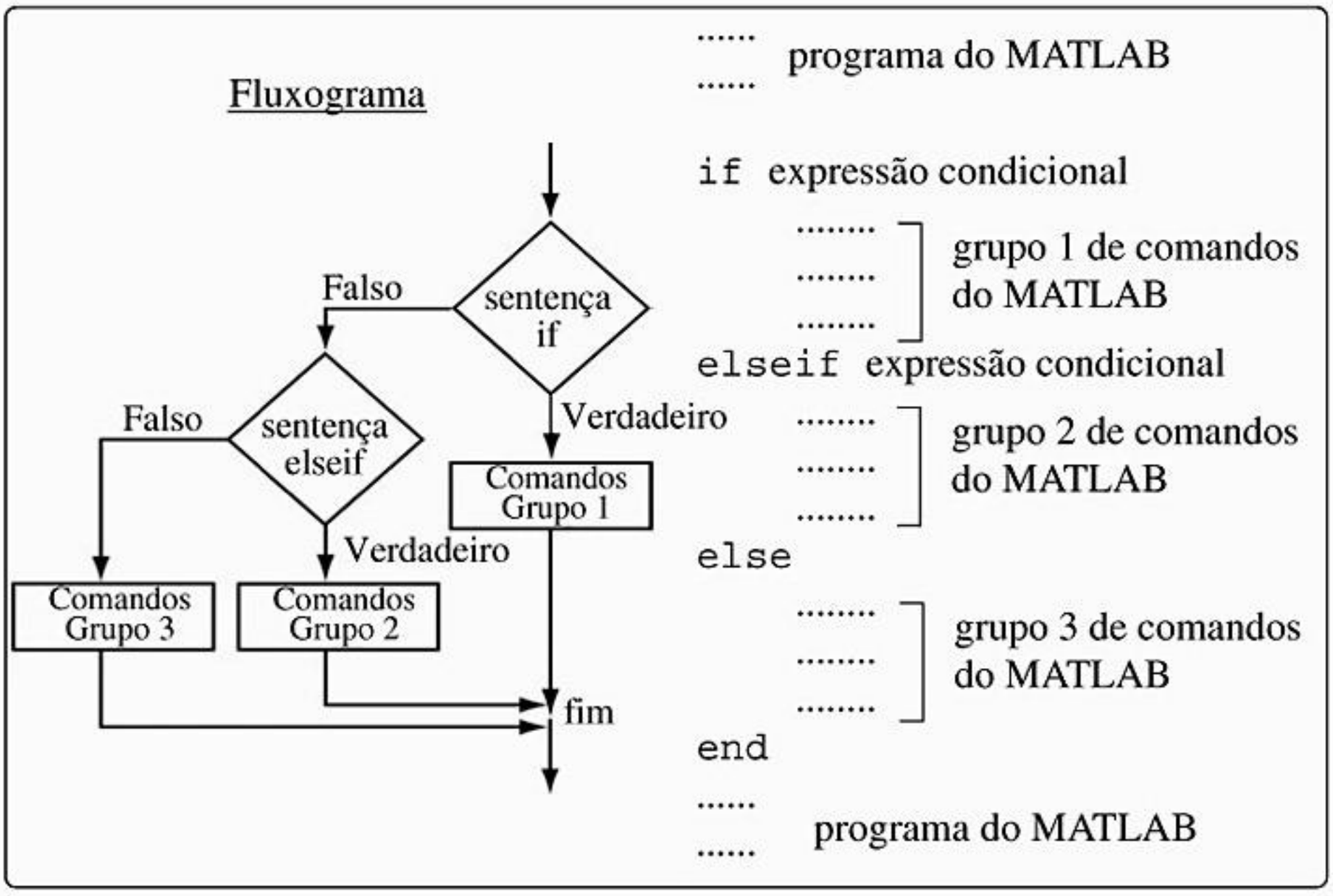


FIGURA 7-3 A estrutura da sentença condicional `if-elseif-else`.

Deve ser enfatizado que é possível aninhar muitas sentenças `elseif` na estrutura e vários grupos de comandos podem ser adicionados dentro desse ninho. Isso possibilita testar muitas condições num único programa. Além disso, o comando `else` é opcional. Isso significa que, caso sejam aninhadas várias sentenças `elseif` qualquer sentença condicional verdadeira terá os respectivos comandos executados. Caso contrário, não haverá execução de comandos, a menos que haja um `else` na estrutura.

7.3 A SENTENÇA `switch-case`

A sentença `switch-case` é um modo mais elegante de selecionar uma dentre várias opções de um programa. A estrutura da sentença é mostrada na Figura 7-4.

- A primeira linha deve conter o comando `switch`, escrito da seguinte forma:

`switch` valor (opcao) do *switch*

O valor ou opção do *switch* pode ser um número ou uma *string*. É, tipicamente, uma variável inicializada com um número ou uma *string*. Entretanto, é possível introduzir uma expressão matemática, cujas variáveis encontram-se declaradas na opção do *switch*.

- Seguindo na estrutura do comando `switch`, vem um ou muitos comandos `case`. Cada `case` possui um valor característico (numérico ou *string*) no lado direito do comando (valor1, valor2, etc.) e um grupo de comandos associados abaixo dele.

```
.....
.....      programa do MATLAB

switch expressão de switch
     case valor 1
     ........
     ........      ] grupo 1 de comandos
     case valor 2
     ........
     ........      ] grupo 2 de comandos
     case valor 3
     ........
     ........      ] grupo 3 de comandos
     otherwise
     ........
     ........      ] grupo 4 de comandos
end
.....
.....    programa do MATLAB
```

FIGURA 7-4 A estrutura da sentença `switch-case`.

- Após o último comando `case`, vem o comando opcional `otherwise`, seguido de um grupo de comandos.
- A última linha sempre deve conter uma sentença `end`.

Como funciona a sentença `switch-case`*?*

O valor da expressão *switch* no comando `swicth` é comparado com os valores relacionados a cada `case`. Caso seja encontrada uma coincidência, o grupo de comandos relacionados ao `case` é executado. (Somente um grupo de comandos; aquele situado entre dois `cases`, ou entre um `case` e um `otherwise` ou entre `otherwise` e `end`.)

- Se houver opção repetida, apenas a primeira será executada.
- Se não for encontrado o valor do *switch* e a sentença `otherwise` estiver presente, o grupo de comandos entre `otherwise` e `end` será executado.
- Se não for encontrado o valor do *switch* e a sentença `otherwise` não estiver presente, nenhum grupo de comandos será executado.
- Uma sentença `case` pode possuir mais de um valor. Isso pode ser feito digitando-se os valores entre chaves, na forma: {`valor1`, `valor2`, `valor3`,...}. Essa forma, não descrita neste livro, é denominada célula de arranjo. O `case` será executado se pelo menos um dos valores entre chaves for encontrado.

NOTA: no MATLAB somente o primeiro `case` *encontrado é executado. Após o grupo de comandos associados ao* `case`*, o programa salta para a sentença* `end`*. Isso distingue o MATLAB de outras linguagens de programação (por exemplo, o C), nas quais um comando* `break` *é necessário na parada de cada* `case`*.*

Problema-exemplo 7-4: Convertendo unidades de energia

Escreva uma rotina que converta uma quantidade de energia ou trabalho escrita em Joule, ft-lb, cal ou eV nas quantidades de energia equivalentes nas demais unidades especificadas pelo usuário. O programa deve solicitar ao usuário que entre com a quantidade de energia, a unidade atual e a nova unidade requerida. A saída é a quantidade de energia escrita na nova unidade.

Fatores de conversão: 1J = 0.738 ft-lb = 0.239 cal = 6.24×10^{18}eV.

Teste o programa para converter:

a) 150J em ft-lb.

b) 2.800 cal em Joules.

c) 2.7eV em cal.

Solução

O programa solução inclui dois conjuntos de sentenças `switch-case` e um conjunto `if-else-end`. O primeiro conjunto `switch-case` é utilizado para converter a quantidade de energia inicial em Joules. O segundo é utilizado para converter a quantidade em Joules para as novas unidades. A estrutura `if-else-end` é utilizada para gerar uma mensagem de erro, caso as unidades sejam digitadas incorretamente.

```
Eent = input('Entre com o valor da energia(trabalho) a ser convertido: ');
Eent_unid = input('Entre com a unidade atual (J, ft-lb,cal ou eV): ','s');
Esai_unid = input('Selecione a nova unidade de energia (J,ft-lb,cal ou eV): ','s');
erro = 0;
switch Eent_unid
case 'J'
   EJ = Eent;
case 'ft-lb'
   EJ = Eent/0.738;
case 'cal'
   EJ = Eent/0.239;
case 'eV'
   EJ = Eent/6.24e18;
otherwise
   erro = 1;
end
switch Esai_unid
```

Atribui 0 à variável `erro`.

Primeira sentença *switch*. O valor do *switch* é uma *string* com as unidades iniciais.

Cada uma das quatro sentenças `case` possui um valor (string) correspondendo a uma unidade inicial e um comando que converte a energia `Eent`, escrita numa unidade qualquer, para Joule (atribui o valor à `EJ`).

Atribui 1 à variável `erro`, caso não seja encontrado o valor do `case`. A unidade inicial foi digitada incorretamente.

Segunda sentença *switch*. O valor do *switch* é uma *string* com as unidades novas.

```
case 'J'
   Esai = EJ;
case 'ft-lb'
   Esai = EJ*0.738;
case 'cal'
   Esai = EJ*0.239;
case 'eV'
   Esai = EJ*6.24e18;
otherwise
   erro = 1;
end
if erro
   disp ('ERRO!!! A unidade atual ou a requerida foi digitada incorretamente.')
else
   fprintf('E = %g %s',Esai,Esai_unid)
end
```

Cada uma das quatro sentenças `case` possui um valor (*string*) correspondendo a uma unidade nova e um comando que converte a energia `EJ` na unidade nova (atribui o valor à `Esai`.).

Atribui 1 à variável `erro` caso não seja encontrado o valor. A unidade nova foi digitada incorretamente.

Estrutura `If-else-end`.

Se a variável `erro` for verdadeira (não-nula), será exibida uma mensagem de erro.

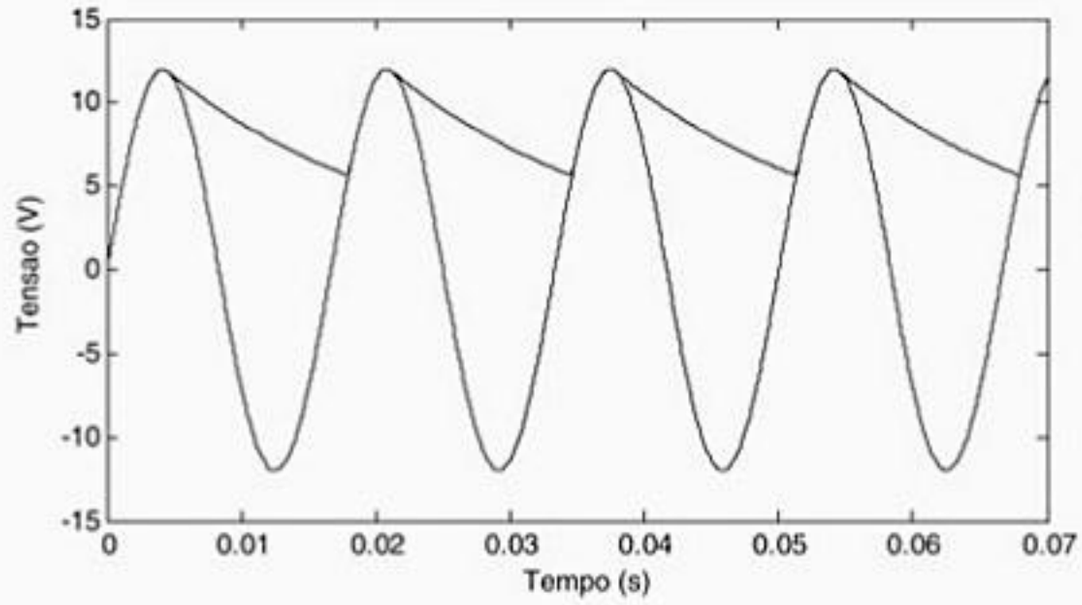
Se `erro` for falsa, a saída exibirá a energia convertida.

A título de exemplo, a rotina (salva como Capitulo7_Exemplo4) é testada na janela Command Window para resolver a letra *b* do problema.

```
>> Capitulo7_Exemplo4
Entre com o valor da energia(trabalho) a ser convertido: 2800
Entre com a unidade atual (J, ft-lb,cal ou eV): cal
Selecione a nova unidade de energia (J,ft-lb,cal ou eV): J
E = 11715.5 J
```

7.4 LAÇOS (*LOOPS*)

O laço é outro método de alterar o fluxo de um programa. Em um laço, a execução de um ou um grupo de comandos é repetida diversas vezes, sucessivamente. Cada seqüência de execução (repetição) do laço é denominada passo. A cada passo, ao menos uma variável é modificada dentro do laço. O MATLAB traz dois tipos de laço: `for-end` e `while-end`. No laço `for-end` (Seção 7.4.1), o número de repetições é conhecido desde o início do laço. No laço `while-end` (Seção 7.4.2), o número de repetições não é conhecido de antemão e o laço é repetido até que uma condição específica de parada seja satisfeita. Ambos os tipos de laço podem ser finalizados a qualquer momento, por meio do comando `break` (veja a Seção 7.6).



$C = 10\ \mu F$

7.8 PROBLEMAS

1. Verifique as seguintes expressões sem usar o MATLAB. Em seguida, compare suas respostas utilizando o MATLAB.

a) $5 <= 8 - 3$

b) $y = 7 < 3 - 1 + 6 > 2$

c) $y = (7 < 3) - 1 + (6 > 2)$

d) $y = 2 \times 4 + 5 == 7 + \frac{20}{4}$

2. Considere $a = 10$ e $b = 6$. Verifique as seguintes expressões sem a ajuda do MATLAB. Em seguida, use o MATLAB e compare com os seus resultados.

a) $y = a >= b$

b) $y = a - b <= \frac{b}{2}$

c) $y = a - (b <= \frac{b}{2})$

3. Considere v = [4 –2 –1 5 0 1 –3 8 2] e w = [0 2 1 –1 0 –2 4 3 2]. Verifique as seguintes expressões sem ajuda do MATLAB. Em seguida, use o MATLAB e compare com os seus resultados.

a) $v >= w$

b) $w \sim= v$

4. Utilize os vetores v e w do problema anterior, mais operadores relacionais, para criar um vetor y formado pelos elementos de w maiores que os elementos de v.

5. Verifique as seguintes expressões sem a ajuda do MATLAB. Confira suas respostas utilizando o MATLAB.

a) 5&–2

b) 8–2|6+5&~2

c) ~(4&0)+8*~(4|0)

6. As temperaturas máximas diárias (em °F) de Nova York (NY) e Anchorage (Alasca), durante o mês de janeiro de 2001, são mostradas nos vetores abaixo (dados da U.S. National Oceanic and Atmospheric Administration).
TNY = [31 26 30 33 33 39 41 41 34 33 45 42 36 39 37 45 43 36 41 37 32 32 35 42 38 33 40 37 36 51 50]
TANC = [37 24 28 25 21 28 46 37 36 20 24 31 34 40 43 36 34 41 42 35 38 36 35 33 42 42 37 26 20 25 31]
Escreva uma rotina para responder às seguintes questões:

a) Calcule a temperatura média mensal em cada cidade.

b) Em quantos dias do mês de janeiro as temperaturas estiveram abaixo da média em cada cidade?

c) Em quantos e em quais dias do mês a temperatura em Anchorage esteve maior que a temperatura em Nova York?

d) Em quantos e em quais dias do mês as temperaturas foram as mesmas em ambas as cidades?

e) Em quantos e em quais dias do mês as temperaturas em ambas as cidades estiveram acima do ponto de congelamento da água (acima de 32°F)?

7. Use o MATLAB de duas formas diferentes, descritas abaixo, para plotar a função:

$$f(x) = \begin{Bmatrix} 4^{ex+2} & \text{para} & -6 \le x \le -2 \\ x^2 & \text{para} & -2 \le x \le -2.5 \\ (x+6.5)^{1/3} & \text{para} & -2.5 \le x \le -6 \end{Bmatrix}$$

a) Escrevendo uma rotina composta de sentenças condicionais e laços.

b) Declarando uma função para $f(x)$ e,então, usando-a em uma rotina para construir o gráfico.

8. Escreva uma rotina que determine as raízes reais da equação quadrática $ax^2 + bx + c = 0$. Salve a função como `raizquad`. Quando o arquivo for executado, o usuário deverá entrar com os valores das constantes a, b e c. Para calcular as raízes da equação, o programa deve primeiro calcular o discriminante D dado por:

$$D = b^2 - 4ac$$

Se $D > 0$, o programa deve exibir a mensagem: "A equação possui duas raízes reais.", e as raízes devem ser mostradas na próxima linha.
Se o $D = 0$, o programa deve exibir a mensagem: "A equação possui duas raízes iguais.", e a raiz deve ser mostrada na próxima linha.
Se $D < 0$, o programa deve exibir a mensagem: "A equação não possui raízes reais."