5. CAPÍTULO 5 – INSTRUÇÕES DE REPETIÇÃO E CÓDIGO DE VETORIZAÇÃO

TERMOS CHAVE			CONTEÚDO
instruções em laços	iterador	laço externo	5.1 O Laço for 2
laços contados	impressão de eco	laço interno	5.2 Laços for Aninhados 8
laços condicionais	soma corrente	laço infinito	5.3 Laços while 16
ação	produto em	fatorial	5.4 Laços com Vetores e Matrizes: Vetorização 26
código vetorizado	execução	sentinela	5.5 Cronometragem 36
iterar	pré-alocar	contando	
laço ou variável	laço aninhado	verificação de erros	

Considere o problema de calcular a área de um círculo com um raio de 0.3 cm. Um programa MATLAB® certamente não é necessário para fazer isso; você usaria sua calculadora e digitaria π * 0.32. No entanto, se uma tabela de áreas circulares é desejada, para raios que variam de 0.1 cm a 100 cm em passos de 0.05 (por exemplo, 0.1, 0.15, 0.2, etc.), seria muito tedioso usar uma calculadora e escrever tudo até o fim. Um dos grandes usos de linguagens de programação e pacotes de software, como o MATLAB, é a capacidade de repetir um processo como esse.

Este capítulo cobrirá as instruções (comandos) do MATLAB que permitem que outras instruções sejam repetidas. As instruções que fazem isso são chamadas de ***instruções de laço***, ***laços***, ou ***comandos de repetição***. Existem dois tipos básicos de laços na programação: laços contados e laços condicionais. Um laço contado é um laço que repete instruções em um número especificado de vezes (assim, antes da execução, sabe-se quantas vezes as instruções devem ser repetidas). Em um laço contado, por exemplo, você pode dizer "repita essas instruções 10 vezes". Um laço condicional também repete instruções, mas, não se sabe, com antecedência, quantas vezes as instruções precisarão ser repetidas. Com um laço condicional, por exemplo, você pode dizer "repita essas instruções até que essa condição se torne falsa". As instruções que são repetidas em qualquer laço são chamadas de ***ação(ões)*** do laço.

Existem duas instruções de laço diferentes no MATLAB: a instrução **for** e a instrução **while**. Na prática, a instrução **for** é usada como o laço contado e o **while** normalmente é usado como laço condicional. Para simplificar, é assim que eles serão apresentados aqui. Em muitas linguagens de programação, percorrer os elementos em um vetor ou matriz é um conceito muito fundamental. No MATLAB, no entanto, como é escrito para trabalhar com vetores e matrizes, o laço através de elementos geralmente não é necessário. Em vez disso, é utilizado o "código vetorizado", o que significa substituir os laços com matrizes pelo uso de funções e operadores internos. Ambos os métodos serão descritos neste capítulo. As seções anteriores se concentrarão nos "conceitos de programação", usando laços. Estes serão contrastados com "os métodos eficientes", usando ***código vetorizado***. Os laços ainda são relevantes e necessários no MATLAB em outros contextos, mas não normalmente quando se trabalha com vetores ou matrizes.

5.1 O LAÇO FOR

A instrução **for**, ou o laço **for**, é usada quando é necessário repetir instruções em um *script* ou função, e quando é conhecido antecipadamente quantas vezes as instruções serão repetidas. As instruções a serem repetidas são chamadas de ação do laço. Por exemplo, pode-se saber que a ação do laço será repetida cinco vezes. A terminologia usada é que nós iteramos cinco vezes a ação do laço.

A variável usada para iterar os valores é chamada de ***variável do laço*** ou ***variável iteradora***. Por exemplo, a variável pode iterar pelos números inteiros de 1 a 5 (por exemplo, 1, 2, 3, 4 e, em seguida, 5). Embora, em geral, os nomes de variáveis devam ser mnemônicos, é comum em muitas linguagens que uma variável iteradora receba o nome i (e se mais de uma variável iteradora for necessária, i, j, k, l, etc.). Isso é histórico e é devido à forma como variáveis inteiras foram nomeadas na linguagem Fortran. No entanto, no MATLAB i e j são internas a funções que retornam o valor $\sqrt{-1}$, portanto, usar como uma variável do laço substituirá esse valor. Se isso não for um problema, tudo bem em usar i como uma variável do laço.

A forma geral do laço **for** é:

```
for variável_do_laço = intervalo
    ação
end
```

Onde $variável\_do\_laço$ é a variável do laço, "intervalo" é o intervalo de valores através dos quais a variável do laço é para iterar, e a ação do laço consiste em todas as instruções até o **end**. Assim como com instruções **if**, a ação é recuada para facilitar a visualização. O intervalo pode ser especificado usando qualquer vetor, mas normalmente a maneira mais fácil de especificar o intervalo de valores é usar o operador de dois pontos.

Como exemplo, imprimiremos uma coluna de números de 1 a 5.

O CONCEITO DE PROGRAMAÇÃO

O laço pode ser inserido na Janela de Comandos, embora, como **if** e **switch**, os laços farão mais sentido em *scripts* e funções. Na Janela de Comandos, os resultados apareceriam após o laço **for**:

```
>> for i = 1:5
       fprintf('%d\n', i)
   end
1
2
3
4
5
```

O que a instrução **for** realizada era imprimir o valor de i e depois o caractere nova linha para cada valor de i, de 1 a 5 em passos de 1. A primeira coisa que acontece é que i é inicializado para ter o valor 1. Então, a ação do laço é executada, que é a instrução **fprintf** que imprime o

valor de i (1) e, em seguida, o caractere nova linha para mover o cursor para a linha baixo. Então, i é incrementado para ter o valor de 2. Em seguida, a ação do laço é executada, o que imprime 2 e o nova linha. Então, i é incrementado para 3 e isso é impresso; então, i é incrementado para 4 e isso é impresso; e então, finalmente, i é incrementado para 5 e isso é impresso. O valor final de i é 5; esse valor pode ser usado assim que o laço terminar.

O MÉTODO EFICIENTE

Naturalmente, **disp** também pode ser usado para imprimir um vetor de coluna, para obter o mesmo resultado:

```
>> disp([1:5]')
1
2
3
4
5
```

PERGUNTA RÁPIDA!

Como você pode imprimir esta coluna de inteiros (usando o método de programação):

```
0
50
100
150
200
```

Resposta

Em um laço, você pode imprimir esses valores começando com 0, incrementando de 50 e terminando em 200. Cada um é impresso usando uma largura de campo de 3.

```
>> for i = 0:50:200
       fprintf('%3d\n', i)
   end
```

5.1.1Laços for que não Usam a Variável Iteradora na Ação

No exemplo anterior, o valor da variável do laço foi usado na ação do laço **for**: ela foi impressa. Nem sempre é necessário realmente usar o valor da variável do laço, no entanto. Às vezes, a variável é usada simplesmente para iterar ou repetir uma ação por um determinado número de vezes. Por exemplo,

```
for i = 1:3
    fprintf('Eu não vou mascar chiclete\n')
end
```

produz a saída:

```
Eu não vou mascar chiclete
Eu não vou mascar chiclete
Eu não vou mascar chiclete
```

A variável i é necessária para repetir a ação três vezes, mesmo que o valor de i não seja usado na ação do laço.

PERGUNTA RÁPIDA!

Qual seria o resultado do seguinte laço?

```
for i = 4:2:8
    fprintf('eu não vou mascar chiclete\n')
end
```

Resposta

Exatamente a mesma saída acima! Não importa que a variável do laço itere através dos valores 4, depois 6, depois 8 ao invés de 1, 2, 3. Como a variável do laço não é usada na ação, esta é apenas outra maneira de especificar que a ação deve ser repetida três vezes. Claro, usar 1:3 faz mais sentido!

PRÁTICA 5.1

Escreva um laço **for** que imprima uma coluna de cinco *.

5.1.2 Entrada em um Laço for

O *script* a seguir repete o processo de solicitar ao usuário um número e fazer eco na impressão do número (o que significa simplesmente imprimi-lo de volta). Um laço **for** especifica quantas vezes isso ocorrerá. Este é outro exemplo em que a variável do laço não é usada na ação, mas apenas especifica quantas vezes repetir a ação.

ecofor.m

```
% Este script faz um laço for repetir a ação de solicitar
% ao usuário um número e fazer ecoar sua impressão

for vezes = 1:3
    numentrada = input('Digite um número: ');
    fprintf('Você digitou %.1f\n', numentrada)
end
```

```
>> ecofor
Digite um número: 33
Você digitou 33.0
Digite um número: 1.1
Você digitou 1.1
Digite um número: 55
```

```
Você digitou 55.0
```

Neste exemplo, a variável do laço itera os valores de 1 a 3, portanto, a ação é repetida três vezes. A ação consiste em solicitar ao usuário um número e ecoar com uma casa decimal.

5.1.3 Calculando Somas e Produtos

Uma aplicação muito comum de um laço **for** é calcular somas e produtos. Por exemplo, em vez de apenas imprimir os números que o usuário digita, podemos calcular a soma dos números. Para fazer isso, precisamos adicionar cada valor a uma ***variável de soma corrente***. Uma soma contínua que continua mudando, à medida que continuamos adicionando a ela. Primeiro, a soma deve ser inicializada com 0.

Como exemplo, escreveremos uma função $somannums$ que somará os n números digitados pelo usuário; n é um argumento inteiro que é passado para a função. Em uma função para calcular a soma, precisamos de um laço, de uma variável iteradora i e também uma variável para armazenar a soma corrente. Neste caso, usaremos o argumento de saída $soma$ como variável da soma em execução. Toda vez que através do laço, o próximo valor que o usuário digita é adicionado ao valor de $soma$. Esta função retornará o resultado final, que é a soma de todos os números, armazenados no argumento de saída $soma$.

somannums.m

```
function soma = somannums(n)
% sumnnums retorna a soma dos n números
% digitados pelo usuário
% Formato da chamada: somannums(n)

soma = 0;
for i = 1:n
    numentrada = input('Digite um número: ');
    soma = soma + numentrada;
end
end
```

Aqui está um exemplo em que 3 é passado para ser o valor do argumento de entrada n; a função calcula e retorna a soma dos números que o usuário digita, $4 + 3.2 + 1.1$ ou 8.3:

```
>> soma_de_nums = somannums(3)
Digite um número: 4
Digite um número: 3.2
Digite um número: 1.1
soma_de_nums =
   8.3000
```

Outra aplicação muito comum de um laço **for** é calcular o ***produto corrente***. Com um produto, o produto corrente deve ser inicializado com 1 (em oposição a uma soma corrente, que é inicializada com 0).

PRÁTICA 5.2

Escreva uma função $prodnnums$ de que seja semelhante à função $somannums$, mas calcule o produto dos números digitados pelo usuário.

5.1.4 Vetores Pré-Alocados

Quando números são inseridos pelo usuário, muitas vezes é necessário armazená-los em um vetor. Existem dois métodos básicos que podem ser usados para realizar isso. Um método é começar com um vetor vazio e estender o vetor adicionando cada número a ele conforme os números são inseridos pelo usuário. Estender um vetor, no entanto, é muito ineficiente. O que acontece, é que toda vez que um vetor é estendido, um novo "pedaço" de memória deve ser encontrado, grande o suficiente, para conter o novo vetor, e todos os valores devem ser copiados na memória, do local original para o novo. Isso pode demorar muito tempo.

Um método melhor é pré-alocar o vetor com o tamanho correto e, em seguida, alterar o valor de cada elemento para armazenar os números que o usuário digita. Esse método envolve referenciar cada índice do vetor de saída e colocar cada número no próximo elemento no vetor de saída. Este método é muito superior, se for conhecido antecipadamente quantos elementos o vetor terá. Um método comum é usar a função **zeros** para pré-alocar o vetor com o tamanho correto.

A seguir, uma função que realiza isso e retorna o vetor resultante. A função recebe um argumento de entrada n e repete o processo n vezes. Como é sabido que o vetor resultante terá n elementos, o vetor pode ser pré-alocado.

criavetfor.m

```
function vetnum = criavetfor(n)
% criavetfor retorna um vetor de comprimento n
% Ele solicita n números do usuário e os coloca em um vetor
% Formato da chamada: criavetfor(n)

vetnum = zeros(1, n);
for indvet = 1:n
    numentrada = input('Digite um número: ');
    vetnum(indvet) = numentrada;
end
end
```

A seguir está um exemplo de chamada desta função e o armazenamento do vetor resultante em uma variável chamada $meuvetor$.

```
>> meuvetor = criavetfor(3)
Digite um número: 44
Digite um número: 2.3
Digite um número: 11

meuvetor =
   44.0000   2.3000   11.0000
```

É muito importante notar que a variável do laço $indvet$ é usada como o índice no vetor.

PERGUNTA RÁPIDA!

Se você precisa apenas imprimir a soma ou a média dos números que o usuário digita, você precisaria armazená-los em uma variável vetor?

Resposta

Não. Você poderia simplesmente adicionar cada um número a uma soma corrente enquanto os lê em um laço.

PERGUNTA RÁPIDA!

E se você quisesse calcular quantos números que o usuário inseriu eram maiores que a média?

Resposta

Sim, então você precisaria armazená-los em um vetor porque você teria que voltar através deles para contar quantos eram maiores do que a média (ou, alternativamente, você poderia voltar e pedir ao usuário para inseri-los novamente !!) .

5.1.5 Exemplo de Laço for: Subplot

Uma função que é muito útil em todos os tipos de gráficos é a **subplot**, que cria uma matriz de gráficos na Janela da Figura atual. Três argumentos são passados para ela na forma **subplot (r, c, n)**, onde r e c são as dimensões da matriz e n é o número da parcela particular dentro dessa matriz. Os gráficos são numerados em linhas, começando no canto superior esquerdo. Em muitos casos, é útil criar uma **subplot** em um laço **for** para que a variável do laço possa iterar os números inteiros de 1 a n.

Quando a função **subplot** é chamada em um **laço**, os dois primeiros argumentos sempre serão os mesmos, pois fornecem as dimensões da matriz. O terceiro argumento iterará pelos números atribuídos aos elementos da matriz. Quando a função **subplot** é chamada, ela torna o elemento especificado o gráfico “ativo”; então, qualquer função de plotagem pode ser usada com formatação completa, como rotulagem de eixo e títulos dentro desse elemento.

Por exemplo, a seguinte **subplot** mostra a diferença, em uma janela da figura, entre usar 20 pontos e 40 pontos para plotar **sin(x)** entre 0 e 2 * π. A função **subplot** cria um vetor de linhas de 1 x 2 linhas na Janela de Figura, para que as duas plotagens sejam mostradas lado a lado. A variável do laço i itera os valores 1 e 2.

Na primeira vez através do laço, quando eu tenho o valor 1, 20 * 1 ou 20 pontos são usados, e o valor do terceiro argumento para a função de **subplot** é 1. A segunda vez através do laço, 40 pontos são usados e o terceiro argumento para “subplotar” é 2. A Janela de Figura resultante com ambas as parcelas é mostrada na Figura 5.1.

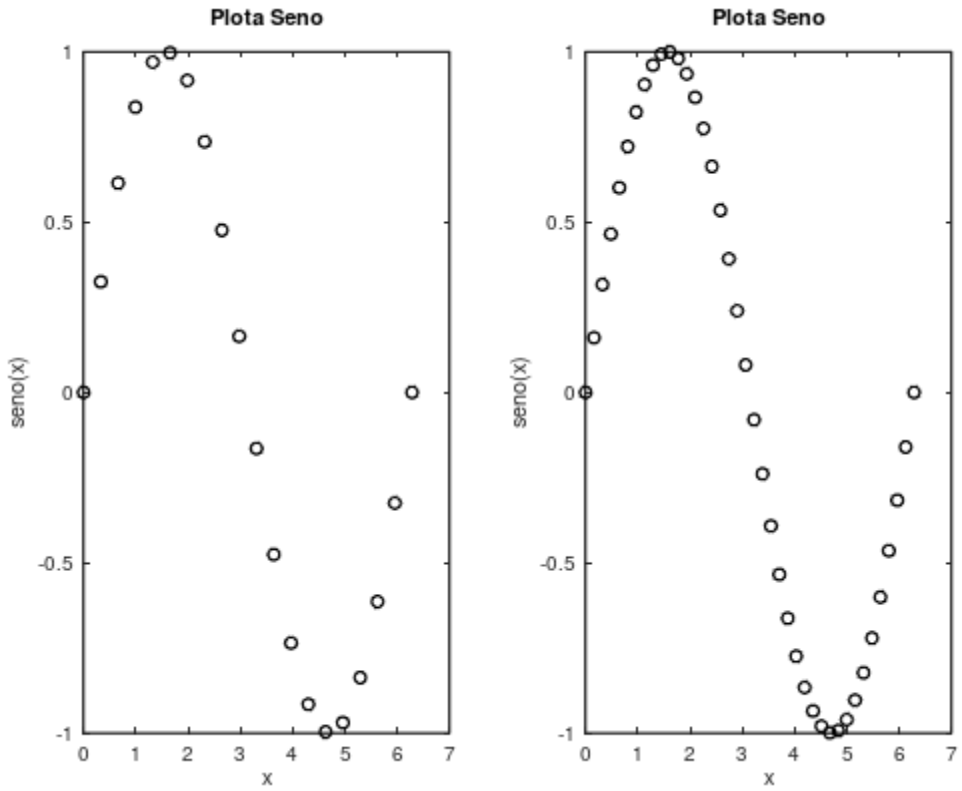

FIGURA 5.1 **subplot** para demonstrar um gráfico usando 20 pontos e 40 pontos

demosubplot.m

```
% Demonstra subplot, usando um laço for
for i = 1:2
    x = linspace(0, 2*pi, 20*i);
    y = sin(x);
    subplot(1, 2, i)
    plot(x, y, 'ko')
    xlabel('x')
    ylabel('seno(x)')
    title('Plota Seno')
end
```

Observe que, uma vez que as funções de manipulação de *strings* serão abordadas no Capítulo 7, será possível ter títulos personalizados (por exemplo, mostrando o número de pontos).

5.2 LAÇOS FOR ANINHADOS

A ação de um laço pode ser qualquer instrução válida. Quando a ação de um laço é outro laço, isso é chamado de laço aninhado.

A forma geral de um laço aninhado é a seguinte:

```
for varlacoum = intervaloum % laço externo
    % acaoum que inclui o laço interno
    for varlacodois = intervalodois % laço interno
        acaodois
    end
end
```

O primeiro laço **for** é chamado de ***laço externo***; o segundo laço **for** é chamado de ***laço interno***. A ação do laço externo consiste (em parte; pode haver outras instruções) de todo o laço interno.

Como exemplo, um laço aninhado será mostrado em um *script* que imprimirá um retângulo de asteriscos (*). Variáveis no *script* especificarão quantas linhas e colunas serão impressas. Por exemplo, se as linhas tiverem o valor 3 e as colunas tiverem o valor 5, um retângulo 3 x 5 será impresso. Como as linhas de saída são controladas pela impressão do caractere nova linha, o algoritmo básico é o seguinte.

Para cada linha de saída:

- imprime o número necessário de asteriscos
- move o cursor para a próxima linha (impressão '\ n').

impasteriscos.m

```
% Imprime um retângulo de asteriscos
% As dimensões serão especificadas por duas variáveis
% para o número de linhas e colunas

linhas = 3;
colunas = 5;
% laço sobre as linhas
for i = 1:linhas
    % para cada repetição de linha, imprimir *'s e,
    % em seguida, um \n
    for j = 1:colunas
        fprintf('*')
    end
    fprintf('\n')
end
```

A execução do *script* exibe a saída:

```
>> impasteriscos
*****
*****
*****
```

A variável $linhas$ especifica o número de linhas a serem impressas e a variável $colunas$ especifica quantos asteriscos devem ser impressas em cada linha. Existem duas variáveis do laço: i é a variável do laço sobre as linhas e j é a variável do laço sobre as colunas. Como o número de linhas é conhecido e o número de colunas é conhecido (dado pelas variáveis de linhas e colunas), são usados laços **for**. Há um para laço para percorrer as linhas e outro para imprimir o número necessário de asteriscos para cada linha.

Os valores das variáveis dos laços não são usados dentro dos laços, mas são usados simplesmente para iterar o número correto de vezes. O primeiro laço **for** especifica que a ação será repetida $linhas$ vezes. A ação desse laço é imprimir os asteriscos e, em seguida, o caractere nova linha. Especificamente, a ação é fazer um laço para imprimir asteriscos das colunas (por exemplo, cinco asteriscos) através de uma linha. Em seguida, o caractere nova

linha é impresso depois de todos os cinco asteriscos, para então mover o cursor para a próxima linha.

Nesse caso, o laço externo está sobre as linhas e o laço interno está sobre as colunas. O laço externo deve estar sobre as linhas porque o *script* está imprimindo um certo número de linhas de saída. Para cada linha, um laço é necessário para imprimir o número necessário de asteriscos; este é o laço **for** interno.

Quando esse *script* é executado, primeiro a variável do laço externo i é inicializada com 1. Então, a ação é executada. A ação consiste no laço interno e, em seguida, na impressão do caractere nova linha. Portanto, enquanto a variável do laço externo possui o valor 1, a variável do laço interno j itera em todos os seus valores. Como o valor das colunas é 5, o laço interno imprime um único asterisco cinco vezes. Em seguida, o caractere nova linha é impresso e, em seguida, a variável do laço externo i é incrementada para 2. A ação do laço externo é executada novamente, o que significa que o laço interno imprimirá cinco asteriscos e o caractere nova linha, então, será impresso. Isso continua, e, ao todo, a ação do laço externo será executada $linhas$ vezes. Observe que a ação do laço externo consiste em duas instruções (o laço **for** e uma instrução **fprintf**). A ação do laço interno, no entanto, é apenas uma única instrução **fprintf**. A instrução **fprintf** para imprimir o caractere nova linha deve ser separada da outra instrução **fprintf** que imprime o caractere asterisco. Se simplesmente tivéssemos

```
fprintf('*\n')
```

como a ação do laço interno, isso imprimiria uma coluna longa de 15 estrelas, não um retângulo 3 x 5.

PERGUNTA RÁPIDA!

Como esse script poderia ser modificado para imprimir um triângulo de asteriscos, como o seguinte, em vez de um retângulo:

```
*
**
***
```

Resposta

Nesse caso, o número de estrelas a serem impressas em cada linha é o mesmo que o número da linha (por exemplo, uma estrela é impressa na linha 1, duas estrelas na linha 2 e assim por diante). O laço for interno não faz um laço para colunas, mas para o valor da variável do laço de linha (portanto, não precisamos das variáveis $colunas$):

trianguloaster.m

```
% Imprime um triângulo de estrelas
% Quantos, serão especificados por uma variável
% para o número de linhas
linhas = 3;
for i = 1:linhas
    % laço interno apenas itera para o valor de i
    for j = 1:i
        fprintf('*')
    end
    fprintf('\n')
end
```

Nos exemplos anteriores, as variáveis dos laços foram usadas apenas para especificar o número de vezes que a ação deveria ser repetida. No próximo exemplo, os valores reais das variáveis dos laços serão impressos.

exibevarlacos.m

```
% Exibe as variáveis do laço
for i = 1:3
    for j = 1:2
        fprintf('i = %d, j = %d\n', i, j)
    end
    fprintf('\n')
end
```

A execução desse *script* imprimiria os valores de i e j em uma linha sempre que a ação do laço interno fosse executada. A ação do laço externo consiste no laço interno e na impressão de um caractere nova linha, portanto, há uma separação entre as ações do laço externo:

```
>> exibevarlacos
i=1, j=1
i=1, j=2
i=2, j=1
i=2, j=2
i=3, j=1
i=3, j=2
```

Agora, em vez de apenas imprimir as variáveis do laço, podemos usá-las para produzir uma tabela de multiplicação, multiplicando os valores das variáveis de laço.

A seguinte função $tabelamult$ calcula e retorna uma matriz que é uma tabela de multiplicação. Dois argumentos são passados para a função, que são o número de linhas e de colunas para essa matriz.

tabelamult.m

```
function matsaida = tabelamult(linhas, colunas)
% tabelamult retorna uma matriz que é uma
% tabela de multiplicação
% Formato da chamada: tabelamult(nLinhas, nColunas)
% Pré-aloca a matriz

matsaida = zeros(linhas, colunas);
for i = 1:linhas
    for j = 1:colunas
        matsaida(i, j) = i * j;
    end
end
end
```

No exemplo a seguir de chamada dessa função, a matriz resultante possui três linhas e cinco colunas:

```
>> tabelamult(3, 5)
ans =
    1    2    3    4    5
    2    4    6    8   10
    3    6    9   12   15
```

Note que esta é uma função que retorna uma matriz. Ele pré-aloca a matriz com a função **zeros** e, em seguida, substitui cada elemento. Como o número de linhas e colunas é conhecido, os laços são usados. O laço externo faz um laço sobre as linhas e o laço interno faz um laço sobre as colunas. A ação do laço aninhado calcula $i * j$ para todos os valores de i e j. Assim como nos vetores, é importante notar novamente que as variáveis dos laços são usadas como índices da matriz.

Quando eu tenho o valor 1, *j* itera os valores de 1 a 5, então, primeiro calculamos $1 * 1$, depois 1 * 2, depois 1 * 3, depois 1 * 4 e, finalmente, 1 * 5. Estes são os valores da primeira linha (primeiro no elemento (1, 1), depois (1, 2), depois (1, 3), depois (1, 4) e finalmente (1, 5)). Então, quando eu tenho o valor 2, os elementos na segunda linha da matriz de saída são calculados, como *j* novamente itera os valores de 1 a 5. Finalmente, quando eu tenho o valor 3, os valores na terceira linha são calculado ($3 * 1$, $3 * 2$, $3 * 3$, $3 * 4$ e $3 * 5$).

Essa função pode ser usada em um *script* que solicita ao usuário o número de linhas e colunas, chama essa função para retornar uma tabela de multiplicação e grava a matriz resultante em um arquivo:

criatabmult.m

```
% Solicitar ao usuário linhas e colunas e
% criar uma tabela de multiplicação para armazenar no
% arquivo "minhatabmult.dat"

num_lins = input ('Digite o número de linhas: ');
num_cols = input ('Digite o número de colunas: ');
matrizprod = tabelamult(num_ lins, num_cols);
save minhatabmult.dat matrizprod -ascii
```

A seguir, um exemplo de execução desse *script* e, em seguida, o carregamento do arquivo em uma matriz para verificar se o arquivo foi criado:

```
>> criatabmult
Digite o número de linhas: 6
Digite o número de colunas: 4

>> load minhatabmult.dat
>> minhatabmult
minhatabmult =
    1    2    3    4
    2    4    6    8
    3    6    9   12
    4    8   12   16
    5   10   15   20
    6   12   18   24
```

PRÁTICA 5.3

Para cada um dos seguintes itens (eles são separados), determine o que seria impresso. Em seguida, verifique suas respostas testando-as no MATLAB.

```
mat = [7 11 3; 3:5];
[l, c] = size(mat);
for i = 1:l
    fprintf('A soma é %d\n', soma(mat(i, :)))
end
```

```
for i = 1:2
    fprintf('%d: ', i)
    for j = 1:4
        fprintf('%d ', j)
    end
    fprintf('\n')
end
```

5.2.1 Combinando Laços Aninhados e Instruções if

As instruções dentro de um laço aninhado podem ser quaisquer instruções válidas, incluindo qualquer instrução de seleção. Por exemplo, pode haver uma instrução **if** ou **if-else** como a ação, ou parte da ação, em um laço. Por exemplo, suponha que exista um arquivo chamado "$valdados.dat$" contendo resultados registrados de uma experiência. No entanto, alguns foram registrados erroneamente. Todos os números devem ser positivos. O *script* a seguir lê esse arquivo em uma matriz. Imprime a soma de cada linha apenas dos números positivos. Assumiremos que o arquivo contém números inteiros, mas não é assumido quantas linhas estão no arquivo ou quantos números por linha (apesar de assumirmos que há o mesmo número de inteiros em cada linha).

somasopos.m

```
% Soma apenas números positivos do arquivo
% Lê do arquivo para uma matriz e, em seguida,
% calcula e imprime a soma apenas
% de números positivos de cada linha

load valdados.dat
[l c] = size(valdados);
for linha = 1:l
somacorrente = 0;
for col = 1:c
    if valdados(lin, col) >= 0
        somacorrente = somacorrente + valdados(lin, col);
    end
end
fprintf('A soma da linha %d é %d\n', linha, somacorrente)
end
```

Por exemplo, se o arquivo contiver:

```
   33  -11    2
    4    5    9
   22    5   -7
    2   11    3
```

a saída do programa ficaria assim:

```
>> somasopos
A soma da linha 1 é 35
A soma da linha 2 é 18
A soma da linha 3 é 27
A soma da linha 4 é 16
```

O arquivo é carregado e os dados são armazenados em uma variável matriz. O *script* localiza as dimensões da matriz e percorre todos os elementos da matriz usando um laço aninhado; o laço externo itera através das linhas e o laço interno itera através das colunas. Isso é importante; como uma ação é desejada para cada linha, o laço externo deve iterar sobre as linhas. Para cada elemento, uma instrução **if-else** determina se o elemento é positivo ou não. Apenas adiciona os valores positivos à soma da linha. Como a soma é calculada para cada

linha, a variável $somacorrente$ é inicializada para 0 para cada linha, ou seja, dentro do laço externo.

PERGUNTA RÁPIDA!

Seria importante se a ordem dos laços fosse invertida neste exemplo, para que o laço externo iterasse sobre as colunas e o laço interno sobre as linhas?

Resposta

Sim, como queremos uma soma para cada linha, o laço externo deve iterar sobre as linhas.

PERGUNTA RÁPIDA!

O que você teria que mudar para calcular e imprimir a soma apenas dos números positivos de cada coluna em vez de cada linha?

Resposta

Você inverteria os dois laços e mudaria a frase para dizer "A soma da coluna". Isso é tudo o que mudaria. Os elementos na matriz ainda seriam referenciados como dados (lin, col). O índice da linha é sempre dado primeiro, depois o índice da coluna e independente da ordem dos laços.

PRÁTICA 5.4

Escreva uma função $meusmincols$ que encontre o valor mínimo em cada coluna de um argumento de matriz e retorne um vetor dos mínimos da coluna. Um exemplo de chamar a função é o seguinte:

```
>> mat = randi(20, 3, 4)
mat =
   15   19   17    5
    6   14   13   13
    9    5    3   13
>> meusmincols(mat)
ans =
   6   5   3   5
```

PERGUNTA RÁPIDA!

A função $meusmincols$ no problema Prática 5.4 também funcionaria para um argumento vetorial?

Resposta

Sim, deve, como um vetor é apenas um subconjunto de uma matriz. Nesse caso, uma das ações do laço seria executada apenas uma vez (para as linhas, se for um vetor de linha ou para as colunas, se for um vetor de coluna).

5.3 LAÇOS WHILE

A instrução **while** é usada como o laço condicional no MATLAB; é usado para repetir uma ação quando antecipadamente não se sabe quantas vezes a ação será repetida. A forma geral da instrução **while** é:

```
while condição
    ação
end
```

A ação, que consiste em qualquer número de instruções, é executada desde que a condição seja ***verdadeira***.

A maneira como funciona é que primeiro a condição é avaliada. Se for logicamente ***verdadeira***, a ação é executada. Então, para começar, a instrução **while** é como uma instrução **if**. No entanto, nesse ponto, a condição é avaliada novamente. Se ainda for ***verdadeira***, a ação é executada novamente. Então, a condição é avaliada novamente. Se ainda for ***verdadeira***, a ação é executada novamente. Então, a condição é avaliada novamente ... Eventualmente, isso tem que parar! Eventualmente, algo na ação tem que mudar alguma coisa na condição, então ela se torna ***falsa***. A condição deve eventualmente tornar-se ***falsa*** para evitar um ***laço infinito***. (Se isso acontecer, Ctrl-C sairá do laço.)

Como exemplo de um laço condicional, escreveremos uma função que encontrará o primeiro fatorial que é maior do que o argumento de entrada $superior$. Para um inteiro n, o fatorial de n, escrito como $n!$, é definido como $n! = 1 * 2 * 3 * 4 * \ldots * n$. Para calcular um fatorial, um laço for seria usado. No entanto, neste caso, não sabemos o valor de n, portanto, temos que continuar calculando o próximo fatorial até que um valor seja atingido, o que significa que deve-se usar um laço **while**.

O algoritmo básico é ter duas variáveis: uma que percorre os valores 1, 2, 3 e assim por diante; e um que armazena o fatorial do iterador em cada etapa. Começamos com 1 e 1 fatorial, que é 1. Então, verificamos o fatorial. Se não for maior que $superior$, a variável iteradora incrementará para 2 e localizará seu fatorial (2). Se isso não for maior que o valor $superior$, o iterador aumentará para 3 e a função encontrará seu fatorial (6). Isso continua até chegarmos ao primeiro fatorial que é maior que $superior$.

Assim, o processo de incrementar uma variável e encontrar seu fatorial é repetido até chegarmos ao primeiro valor maior que alto. Isso é implementado usando um laço **while**:

fatmaiorsup.m

```
function fatmaior = fatmaiorsup(superior)
% fatmaiorsup retorna o primeiro fatorial > superior
% Formato da chamada: fatmaiorsup(entradaInteiro)

fator = 0;
fatorial = 1;
while fatorial <= superior
    fator = fator + 1;
    fatorial = fatorial * fator;
end
fatmaior = fatorial;
end
```

Um exemplo de chamada da função, passando 5000 para o valor do argumento de entrada, segue:

```
>> fatmaiorsup(5000)
ans =
   5040
```

A variável iteradora $fator$ é inicializada em 0 e a variável de produto corrente $fatorial$, que armazenará o fatorial de cada valor de $fator$, é inicializada em 1. Na primeira vez em que o laço **while** é executado, a condição é ***verdadeira***: 1 é menor ou igual a 5000. Assim, a ação do laço é executada, que é incrementar $fator$ para 1 e $fatorial$ se torna 1 $(1 * 1)$.

Após a execução da ação do laço, a condição é avaliada novamente. Como ainda será ***verdadeira***, a ação é executada: $fator$ é incrementado para 2 e $fator$ receberá o valor 2 $(1 * 2)$. O valor 2 ainda é ≤ 5000, então a ação será executada novamente: $fator$ será incrementada para 3 e $fatorial$ receberá o valor 6 $(2 * 3)$. Isso continua até que o primeiro valor de $fatorial$ seja encontrado > 5000. Assim que $fatorial$ chegar a este valor, a condição será ***falsa*** e o laço **while** terminará. Nesse ponto, o fatorial é atribuído ao argumento de saída, que retorna o valor.

A razão pela qual $fator$ é inicializado com 0 ao invés de 1 é que a primeira vez que a ação de laço é executada, $fator$ se torna 1 e $fatorial$ se torna 1, então temos 1 e 1!, que é 1.

5.3.1 Condições Múltiplas em um Laço while

Na função $fatmaiorsup$, a condição no laço **while** consistia em uma expressão, que testava se a variável $fatorial$ era menor ou igual à variável $superior$. Em muitos casos, no entanto, a condição será mais complicada do que isso e poderia usar o operador || (ou) ou o operador && (e). Por exemplo, pode ser que seja desejado ficar em um laço **while**, desde que a variável x esteja em um intervalo específico:

```
while x >= 0 && x <= 100
```

Como outro exemplo, a continuação da ação de um laço pode ser desejada desde que pelo menos uma das duas variáveis esteja em um intervalo especificado:

```
while x < 50 || y < 100
```

5.3.2 Lendo de um Arquivo Usando um Laço while

O exemplo a seguir ilustra a leitura de um arquivo de dados usando um laço **while**. Os dados de uma experiência foram registrados em um arquivo chamado "$dadosexp.dat$". O arquivo tem alguns pesos seguidos de um -99 e depois mais pesos, todos na mesma linha. Os únicos valores de dados nos quais estamos interessados, no entanto, são aqueles anteriores a -99. O -99 é um exemplo de sentinela, que é um marcador entre os conjuntos de dados.

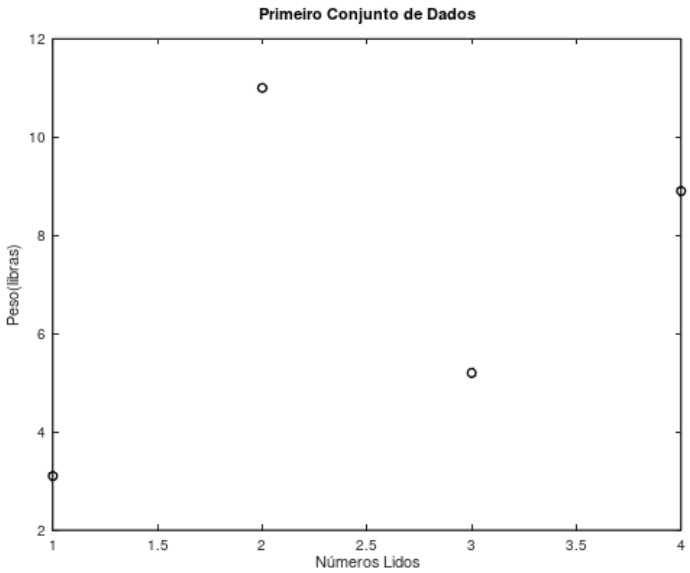

FIGURA 5.2 Gráfico de alguns (mas não todos) dados de um arquivo

O algoritmo para o *script* é:

- ler os dados do arquivo para um vetor
- criar uma nova variável vetor $vetnovo$ que só tem os valores de dados até, mas não incluindo o -99
- plotar os valores do novo vetor, usando círculos pretos.

Por exemplo, se o arquivo contiver o seguinte:

```
3.1   11    5.2    8.9   -99    4.4    62
```

o gráfico produzido se pareceria com a Figura 5.2.

Por simplicidade, assumiremos que o formato do arquivo é conforme especificado. O uso de **load** criará um vetor com o nome $dadosexp$, que contém os valores do arquivo.

O CONCEITO DE PROGRAMAÇÃO

Usando o método de programação, nós percorremos o vetor até que o -99 seja encontrado, criando o novo vetor armazenando cada elemento do $dadosexp$ no vetor $novovet$.

achavalwhile.m

```
% Lê dados de um arquivo, mas apenas plota os números
% até uma sentinela -99. Usa um laço while.

load dadosexp.dat
i = 1;
while dadosexp(i) ~= -99
    vetnovo(i) = dadosexp(i);
    i = i + 1;
end
plot(vetnovo, 'ko')
xlabel ('Números Lidos')
ylabel ('Peso(libras)')
title ('Primeiro Conjunto de Dados')
```

Note que isso estende o vetor $vetnovo$ toda vez que a ação do laço é executada.

O MÉTODO EFICIENTE

Usando a função **find**, podemos localizar o índice do elemento que armazena o -99. Então, o novo vetor compreende todo o vetor original do primeiro elemento até o índice antes do índice do elemento que armazena o -99.

achaval.m

```
% Lê dados de um arquivo, mas apenas plota os números
% até um sentinela -99.Usa find e o operador de dois pontos

load dadosexp.dat
onde = find(dadosexp == -99);
vetnovo = dadosexp(1:onde-1);
plot(vetnovo, 'ko')
xlabel('Números Lidos')
ylabel('Peso (libras)')
title('Primeiro Conjunto de Dados')
```

5.3.3 Entrada em um Laço while

Às vezes, um laço **while** é usado para processar a entrada do usuário, desde que o usuário esteja inserindo dados em um formato correto. O *script* a seguir repete o processo de solicitar do usuário, lendo em um número positivo e fazendo eco da impressão, contanto que o usuário insira corretamente números positivos quando solicitado. Assim que o usuário digita um número negativo, o script imprime “OK” e termina.

numposwhile.m

```
% Solicita do usuário e imprime o eco dos números
% digitados, até que o usuário insira um número negativo

numentrada = input('Digite um número positivo: ');
while numentrada >= 0
    fprintf('Você digitou %d.\n\n', numentrada)
    numentrada = input('Digite um número positivo: ');
end
fprintf('OK!\n')
```

Quando o *script* é executado, a entrada/saída pode se parecer com isto:

```
>> numposwhile
Digite um número positivo: 6
Você digitou um 6.

Digite um número positivo: -2
OK!
```

Observe que o *prompt* é repetido no *script*: uma vez antes do laço e, em seguida, novamente no final da ação. Isso é feito para que, toda vez que a condição for avaliada, haja um novo valor de $numentrada$ a ser verificado. Se o usuário inserir um número negativo pela primeira vez, nenhum valor será impresso em eco:

```
>> numposwhile
Digite um número positivo: -33
OK!
```

Como vimos anteriormente, o MATLAB fornecerá uma mensagem de erro se um caractere for inserido em vez de um número.

```
>> numposwhile
Digite um número positivo: a
Error using input
Undefined function or variable 'a'.

Error in numposwhile (linha 4)
inputnum = input('Digite um número positivo: ');
Digite um número positivo: -4
OK!
```

No entanto, se o caractere for realmente o nome de uma variável, ele usará o valor dessa variável como entrada. Por exemplo:

```
>> a = 5;
>> numposwhile
Digite um número positivo: a
Você digitou um 5.
Digite um número positivo: -4
```

```
OK!
```

Se for desejado armazenar todos os números positivos que o usuário digite, nós os armazenaríamos, um de cada vez, em um vetor. No entanto, como não sabemos antecipadamente quantos elementos precisaremos, não podemos pré-alocar o tamanho correto. Os dois métodos de estender um vetor, de um elemento por vez, são mostrados aqui. Podemos começar com um vetor vazio e concatenar cada valor para o vetor, ou podemos incrementar um índice.

```
vetnum = [];
numentrada = input('Digite um número positivo: ');
while numentrada >= 0
    vetnum = [vetnum numentrada];
    numentrada = input('Digite um número positivo: ');
end
% OU:
i = 0;
numentrada = input('Digite um número positivo: ');
while numentrada >= 0
    i = i + 1;
    vetnum(i) = numentrada;
    numentrada = input('Digite um número positivo: ');
end
```

Tenha em mente que isso é ineficiente e deve ser evitado se o vetor puder ser pré-alocado.

5.3.4 Contando em umLaço while

Embora laços **while** sejam usados quando o número de vezes que a ação será repetida não for conhecido antes do tempo, geralmente é útil saber quantas vezes a ação foi, de fato, repetida. Nesse caso, é necessário contar o número de vezes que a ação é executada. A seguinte variação do *script* anterior conta o número de números positivos que o usuário insere com sucesso.

contanumpos.m

```
% Solicita ao usuário números positivos e imprime o eco
% desde que o usuário insira números positivos
% Conta os números positivos inseridos pelo usuário

contador = 0;
numentrada = input ('Digite um número positivo: ');
while numentrada >= 0
    fprintf('Você digitou% d.\n\n', numentrada)
    contador = contador + 1;
    numentrada = input('Digite um número positivo: ');
end
fprintf('Obrigado, você digitou %d números positivos.\n', ...
contador)
```

O *script* inicializa um contador de variáveis com 0. Em seguida, na ação do laço **while**, toda vez que o usuário insere um número, o *script* incrementa a variável do contador. No final do *script*, imprime o número de números positivos que foram inseridos.

```
>> contanumpos
Digite um número positivo: 4
Você digitou um 4.

Digite um número positivo: 11
Você digitou um 11.

Digite um número positivo: -4
Obrigado, você digitou 2 números positivos.
```

PRÁTICA 5.5

Escreva um *script* medianumneg que repita o processo de solicitar ao usuário números negativos, até que o usuário insira um número zero ou positivo, como acabamos de mostrar. Em vez de imprimi-los, no entanto, o *script* imprimirá a média (apenas dos números negativos). Se nenhum número negativo for inserido, o script imprimirá uma mensagem de erro em vez da média. Use o método de programação. Exemplos de execução deste *script* seguem:

```
>> mediaumneg
Digite um número negativo: 5
Nenhum número negativo para a média.

>> mediaumneg
Digite um número negativo: -8
Digite um número negativo: -3
Digite um número negativo: -4
Digite um número negativo: 6
A média foi de -5.00
```

5.3.5 Verificação de Erro de Entrada do Usuário em um Laço while

Na maioria dos aplicativos, quando o usuário é solicitado a inserir algo, há um intervalo válido de valores. Se o usuário inserir um valor incorreto, em vez de fazer o programa continuar com um valor incorreto, ou apenas imprimir uma mensagem de erro, o programa deverá repetir o prompt. O programa deve continuar solicitando ao usuário, lendo o valor e verificando-o até que o usuário insira um valor que esteja no intervalo correto. Esta é uma aplicação muito comum de um laço condicional: executar o laço até que o usuário insira corretamente um valor num programa. Isso é chamado de verificação de erros.

Por exemplo, o *script* a seguir solicita que o usuário insira um número positivo e faz um laço para imprimir uma mensagem de erro e repete o *prompt* até que o usuário finalmente insira um número positivo.

leumnum.m

```
% Executa o laço até o usuário digitar um número positivo

nument = input('Digite um número positivo: ');
while nument < 0
    nument = input('Inválido! Insira um número positivo: ');
end
fprintf('Obrigado, você digitou %.1f\n', nument)
```

Um exemplo de execução deste *script* é o seguinte:

```
>> leumnum
Digite um número positivo: -5
Inválido! Digite um número positivo: -2.2
Inválido! Digite um número positivo: c
Error using input
Undefined function or varable 'c'.
Error in leumnum (line 5)
nument = input('Digite um número positivo: ');
Inválido! Digite um número positivo: 44
Obrigado, digitou 44.0
```

Nota

O próprio MATLAB captura a entrada de caracteres e imprime uma mensagem de erro e repete o *prompt* quando o 'c' é inserido.

PERGUNTA RÁPIDA!

Como poderíamos modificar o exemplo anterior para que o *script* peça ao usuário para inserir números positivos n vezes, onde n é um número inteiro definido como 3?

Resposta

Toda vez que o usuário inserir um valor, o *script* verifica e, em um laço **while**, diz ao usuário que é inválido até que um número positivo válido seja inserido. Colocando a verificação de erro em um laço **for** que se repete n vezes, o usuário é forçado a inserir três números positivos, conforme mostrado a seguir.

lennums.m

```
% Laço até o usuário digitar n números positivos
n = 3;
fprintf('Por favor, digite %d números positivos\n\n', n)
para i = 1:n
nument = input('Digite um número positivo: ');
while nument < 0
    nument = input('Inválido! Insira um número positivo: ');
end
fprintf('Obrigado, você digitou %.1f\n', nument)
end
```

```
>> lennums

Por favor, insira 3 números positivos

Digite um número positivo: 5.2
Obrigado, você digitou 5.2
Digite um número positivo: 6
Obrigado, você digitou 6.0
Digite um número positivo: -7.7
Inválido! Digite um número positivo: 5
Obrigado, você digitou 5.0
```

5.3.5.1 Verificação de Erros para Inteiros

Como o MATLAB usa o tipo **double** por padrão para todos os valores, para verificar se o usuário inseriu um inteiro, o programa tem que converter o valor de entrada para um tipo inteiro (por exemplo, int32) e então verificar se igual à entrada original. Os exemplos a seguir ilustram o conceito. Se o valor da variável num for um número real, convertê-lo para o tipo int32 irá arredondá-lo, então o resultado não é o mesmo que o valor original.

```
>> num = 3.3;
>> numint = int32(num)
inum =
   3
>> num == numint
ans =
   0
```

Se, no entanto, o valor da variável num for um inteiro, convertê-lo em um tipo inteiro não alterará o valor.

```
>> num = 4;
>> numint = int32(num)
inum =
4
>> num == numint
ans =
1
```

O *script* a seguir usa essa ideia para verificar erros em dados inteiros; ele faz um laço até que o usuário insira corretamente um número inteiro.

leumint.m

```
% Verificação de erro até o usuário inserir um inteiro
nument = input('Digite um inteiro: ');
num2 = int32(nument);
while num2 ~= nument
    nument = input('Inválido! Insira um inteiro: ');
    num2 = int32(nument);
end
fprintf('Obrigado, você digitou %d\n', nument)
```

Exemplos de execução deste script são:

```
>> leumint
Digite um número inteiro: 9.5
Inválido! Digite um inteiro: 3.6
Inválido! Digite um número inteiro: -11
Obrigado, você digitou -11

>> leumint
Digite um inteiro: 5
Obrigado, você digitou 5
```

Colocando essas idéias juntas, o *script* a seguir faz um laço até que o usuário insira corretamente um número inteiro positivo. Existem duas partes na condição, pois o valor deve ser positivo e deve ser um inteiro.

leumintpos.m

```
% Verificaçã de erro até que o usuário insira um número
% inteiro positivo
nument = input ('Digite um inteiro positivo: ');
num2 = int32(nument);
while num2 ~= nument || num2 < 0
    nument = input ('Inválido! 'Digite um inteiro positivo: ');
    num2 = int32 (nument);
end
fprintf('Obrigado, você digitou %d\n', nument)
```

```
>> leumintpos
Digite um número inteiro positivo: 5.5
Inválido! Digite um número inteiro positivo: -4
Inválido! Digite um número inteiro positivo: 11
Obrigado, você digitou 11
```

PRÁTICA 5.6

Modifique o script $leumintpos$ para ler n inteiros positivos em vez de apenas um.

5.4 LAÇOS COM VETORES E MATRIZES: VETORIZAÇÃO

Na maioria das linguagens de programação, ao executar uma operação com um vetor, um laço for é usado para percorrer todo o vetor, usando a variável do laço como o índice no vetor. Em geral, no MATLAB, supondo que exista uma variável vetor vet, os índices variam de 1 ao comprimento do vetor, e a instrução **for** percorre todos os elementos executando a mesma operação em cada um deles:

```
for i = 1:length(vet)
    % faz alguma coisa com vet(i)
end
```

Na verdade, esse é um dos motivos para armazenar valores em um vetor. Normalmente, os valores em um vetor representam "a mesma coisa", portanto, normalmente, em um programa, a mesma operação seria executada para cada elemento.

Da mesma forma, para uma operação em uma matriz, um laço aninhado seria necessário e as variáveis de laço que iteram sobre as linhas e colunas são usadas como os subscritos na matriz. Em geral, assumindo uma variável matriz mat, usamos o tamanho para retornar separadamente o número de linhas e colunas e usamos essas variáveis nos laços **for**. Se uma ação é desejada para cada linha na matriz, o laço aninhado seria assim:

```
[l, c] = tamanho (mat);
for lin = 1:l
    for col = 1:c
        % faça alguma coisa com mat(lin, col)
    end
end
```

Se, em vez disso, uma ação for desejada para cada coluna na matriz, o laço externo seria sobre as colunas. (Note, no entanto, que a referência ao elemento de uma matriz sempre se referencia primeiro o índice da linha e depois o índice da coluna.

```
[l, c] = tamanho (mat);
for col = 1:c
    for lin = 1:l
        % faça alguma coisa com mat(lin, col)
    end
end
```

Normalmente, isso não é necessário no MATLAB! Embora os laços for sejam muito úteis para muitas outras aplicações no MATLAB, eles não são normalmente usados para operações em vetores ou matrizes; em vez disso, o método eficiente é usar funções e/ou operadores internos. Isso é chamado de código vetorizado. O uso de laços e instruções de seleção com vetores e matrizes é um conceito básico de programação em muitas outras linguagens, e "o conceito de programação" e "o método eficiente" são destacados nesta seção e, até certo ponto, no restante deste livro.

5.4.1 Vetorizando Somas e Produtos

Por exemplo, digamos que queremos executar uma multiplicação escalar e, nesse caso, multiplicar cada elemento de um vetor v por 3 e armazenar o resultado novamente em v, em que v é inicializado da seguinte forma:

```
>> v = [3 7 2 1];
```

O CONCEITO DE PROGRAMAÇÃO

Para realizar isso, podemos percorrer todos os elementos do vetor e multiplicar cada elemento por 3. A seguir, a saída é suprimida no laço e, em seguida, o vetor resultante é mostrado:

```
>> for i = 1:length(v)
       v(i) = v(i) * 3;
   end
>> v
v =
    9   21    6    3
```

O MÉTODO EFICIENTE

```
>> v = v * 3
```

Como poderíamos calcular o fatorial de n, $n! = 1 * 2 * 3 * 4 * \ldots n$?

O CONCEITO DE PROGRAMAÇÃO

O algoritmo básico é inicializar um produto corrente com 1 e multiplicar o produto em corrente por cada inteiro de 1 a n. Isso é implementado em uma função:

meufatorial.m

```
function prodcorr = meufatorial(n)
% meufatorial retorna n!
% Formato da chamada: meufatorial(n)

prodcorr = 1;
for i = 1:n
    prodcorr = prodcorr * i;
end
end
```

Qualquer argumento inteiro positivo pode ser passado para esta função, e ela calculará o fatorial desse número. Por exemplo, se 5 for passado, a função calculará e retornará $1 * 2 * 3 * 4 * 5$ ou 120:

```
>> meufatorial(5)
```

```
ans =
    120
```

O MÉTODO EFICIENTE

O MATLAB possui uma função inerna, fatorial, que calculará o fatorial de um inteiro n. A função **prod** também pode ser usada para calcular o produto do vetor 1:5.

```
>> fatorial(5)
ans =
   120
>> prod(1:5)
ans =
   120
```

PERGUNTA RÁPIDA!

O MATLAB possui uma função **cumsum** que retorna um vetor de todas as somas correntes de um vetor de entrada. No entanto, muitas outras linguagens, não. Então, como poderíamos escrever a nossa própria **cumsum**?

Resposta

Essencialmente, existem dois métodos de programação que podem ser usados para simular a função **cumsum**. Um método é começar com um vetor vazio e estender o vetor adicionando cada soma a ele enquanto as somas correntes são calculadas. Um método melhor é pré-alocar o vetor para o tamanho correto e, em seguida, alterar o valor de cada elemento para serem somas correntes consecutivas.

minhacumsum.m

```
function vetsaida = minhacumsum(vet)
% minhacumsum imita cumsum para um vetor
% Pré-aloca o vetor de saída
% Formato da chamada: minhacumsum(vetor)

vetsaida = zeros(length(vet));
somacorr = 0;
for i = 1:length(vec)
    somacorr = somacorr + vet(i);
    vetsaida(i) = somacorr;
end
end
```

Um exemplo de chamar a função é o seguinte:

```
>> minhacumsum([5 9 4])
ans =
    5   14   18
```

PRÁTICA 5.7

Escreva uma função que imite a função **cumprod**. Use o método de pré-alocação do vetor de saída.

PERGUNTA RÁPIDA!

Como somaríamos cada coluna individual de uma matriz?

Resposta

O método de programação exigiria um laço aninhado no qual o laço externo itera sobre as colunas. A função irá somar cada coluna e retornar um vetor de linha contendo os resultados.

somacolmat.m

```
function somasaida = somacolmat(mat)
% somacolmat encontra a soma de cada coluna de uma matriz
% Retorna um vetor das somas das colunas
% Formato da chamada: somacolmat(matriz)

[lin, col] = size(mat);
% Pré-aloca o vetor com o número de colunas
somasaida = zeros(1, col);
% Cada coluna está sendo somada de forma que o laço externo
% tem que iterar sobre as colunas
for i = 1:col
    % Inicialize a soma corrente com 0 para cada coluna
    somacorr = 0;
    for j = 1:lin
        somacorr = somacorr + mat(j, i);
    end
    somasaida(i) = runsum;
end
end
```

Observe que o argumento de saída será um vetor de linha contendo o mesmo número de colunas que a matriz de argumento de entrada. Além disso, como a função está calculando uma soma para cada coluna, a variável $somacorr$ deve ser inicializada com 0 para cada coluna, portanto, ela é inicializada dentro do laço externo.

```
>> mat = [3:5; 2 5 7]
mat =
   3   4   5
   2   5   7
>> somacolmat(mat)
ans =
    5    9   12
```

É claro que a função interna **sum** do MATLAB realizaria a mesma coisa, como já vimos.

PRÁTICA 5.8

Modifique a função *somacolmat*. Crie uma função *somalinmat* para calcular e retornar um vetor com todas as somas de linhas em vez de somas de coluna. Por exemplo, chamá-la e passar a variável *mat* da Pergunta Rápida anterior resultaria no seguinte:

```
>> somalinmat(mat)
ans =
   12   14
```

5.4.2 Vetorização de Laços com Instruções de Seleção

Em muitas aplicações, é útil determinar se os números em uma matriz são positivos, zero ou negativos.

O CONCEITO DE PROGRAMAÇÃO

Uma função *sinaldonum* que segue irá realizar isso:

sinaldonum.m

```
function matsaida = sinaldonum(mat)
% sinaldonum imita a função sign
% Formato da chamada: sinaldonum(matriz)

[l, c] = size(mat);
for i = 1:l
    for j = 1:c
        if mat(i, j) > 0
            matsaida(i, j) = 1;
        elseif mat(i, j) == 0
            matsaida(i, j) = 0;
        else
            matsaida(i, j) = -1;
        end
    end
end
end
```

Aqui está um exemplo de uso desta função:

```
>> mat = [0 4 -3; -1 0 2]
mat =
   0   4  -3
  -1   0   2

>> sinaldonum(mat)
ans =
   0   1  -1
```

```
  -1    0    1
```

O MÉTODO EFICIENTE

Uma inspeção de perto revela que a função realiza a mesma tarefa que a função interna **sign**!

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

Outro exemplo de uma aplicação comum em um vetor é encontrar o valor mínimo e / ou máximo no vetor.

O CONCEITO DE PROGRAMAÇÃO

Por exemplo, o algoritmo para encontrar o valor mínimo em um vetor é o seguinte.

- O mínimo de trabalho (o mínimo que foi encontrado até agora) é o primeiro elemento no vetor para começar.
- Executa um laço através restante do vetor (do segundo elemento até o final).
  - Se qualquer elemento for menor que o mínimo de trabalho, esse elemento é o novo mínimo de trabalho. A seguinte função implementa esse algoritmo e retorna o valor mínimo encontrado no vetor.

mindovet.m

```
função minsaida = myminvec (vet)
% mindovet retorna o valor mínimo em um vetor
% Formato da chamada: mindovet(vetor)
minsaida = vet(1);
for i = 2:length(vet)
    if vet(i) < minsaida
        minsaida = vet(i);
    end
end
end
```

```
>> vet = [3 8 99-1];
>> mindovet(vet)
ans =
  -1

>> vet = [3 8 99 11];
>> mindovet(vet)
ans =
   3
```

Nota

Uma instrução **if** é usada no laço em vez de uma instrução **if-else**. Se o valor do próximo elemento no vetor for menor que $minsaida$, o valor de $minsaida$ será alterado; caso contrário, nenhuma ação é necessária.

O MÉTODO EFICIENTE

Use a função **min**:

```
>> vet = [5 9 4];
>> min(vet)
ans =
   4
```

PERGUNTA RÁPIDA!

Determine o que a seguinte função realiza:

xxx.m

```
functiion reslogico = xxx(vet)
% PR para você - o que isso faz?

reslogico = false;
i = 1;
while i <= length(vet) && reslogico == false
    if vet(i) ~= 0
        reslogico = true;
    end
    i = i + 1;
end
end
```

Resposta

A saída produzida por esta função é a mesma que qualquer função para um vetor. Ela inicializa o argumento de saída para ***falso***. Em seguida, ele percorre o vetor e, se algum elemento for diferente de zero, altera o argumento de saída para ***verdadeiro***. Ela executa um laço até que um valor diferente de zero seja encontrado ou tenha passado por todos os elementos.

PERGUNTA RÁPIDA!

Determine o que a seguinte função realiza.

yyy.m

```
function reslogico = yyy(mat)
% PR para você - o que isso faz?

contador = 0;
[l, c] = size(mat);
for i = 1:l
    for j = 1:c
        if mat(i, j) ~= 0
        contador = contador + 1;
    end
end
end
logresult = contador == numel(mat);
end
```

Resposta

A saída produzida por esta função é a mesma que a função **all**.

Como outro exemplo, escreveremos uma função que receberá um vetor e um inteiro como argumentos de entrada e retornará um vetor lógico que armazena ***verdadeiro*** lógico apenas para elementos do vetor que são maiores que o inteiro e ***falso*** para os outros elementos.

O CONCEITO DE PROGRAMAÇÃO

A função recebe dois argumentos de entrada: o vetor e um inteiro n com o qual comparar. Ele percorre todos os elementos no vetor de entrada e armazena ***verdadeiro*** ou ***falso*** no vetor de resultado, dependendo se $vet(i) > n$ é ***verdadeiro*** ou ***falso***.

testavetmaiorn.m

```
funtion vetsaida = testavetmaiorn(vet, n)
% testavetmaiorn testa se elementos de um vetor
% são maiores que n ou não
% Formato da chamada: testavetmaiorn(vetor, n)
% Pré-aloca o vetor para falso lógico

vetsaida = false(size(vet));
for i = 1:length(vet)
    % Se um elemento for > n, mude para verdadeiro
    if vet(i) > n
        vetsaida(i) = true;
    end
end
end
```

Observe que, como o vetor foi pré-alocado para ***falso***, a cláusula **else** não é necessária.

O MÉTODO EFICIENTE

Como vimos, o operador relacional > criará automaticamente um vetor lógico.

testavetmaiorn2.m

```
função outvec = testvecgtnii (vet, n)
% testavetmaiorn2 testa se elementos de um vetor
% são maiores que n ou não - sem laço
% Formato da chamada: testavetmaiorn2(vetor, n)

vetsaida = vet > n;
end
```

PRÁTICA 5.9

Chame a função $testavetmaiorn2$, passando um vetor e um valor para n. Conte quantos valores no vetor foram maiores que n.

5.4.3 Dicas para Escrever Código Eficiente

Para ser capaz de escrever código eficiente no MATLAB, incluindo a vetorização, há vários recursos importantes a serem lembrados:

- operações escalares e de matriz
- vetores lógicos
- funções internas
- pré-alocação de vetores.

Existem muitas funções no MATLAB que podem ser utilizadas em vez de códigos que usam laços e instruções de seleção. Essas funções já foram monstradas, mas vale a pena repeti-las para enfatizar sua utilidade:

- **sum** e **prod** – calcula a soma ou produto de cada elemento em um vetor ou coluna em uma matriz
- **cumsum** e **cumprod** – retorna um vetor ou matriz cumulativa (somas ou produtos correntes)
- **min** e **max** – encontra o valor mínimo ou máximo de um vetor ou em cada coluna de uma matriz
- **any**, **all** e **find** – trabalhar com expressões lógicas
- funções “is”, como **isletter** e **isequal** – retorna valores lógicos.

Em quase todos os casos, o código que é mais rápido de escrever pelo programador também é mais rápido para o MATLAB executar. Portanto, “código eficiente” significa que é eficiente tanto para o programador quanto para o MATLAB.

PRÁTICA 5.10

Vetorize o seguinte código (reescreva o código de forma eficiente):

```
i = 0;
for inc = 0:0.5:3
    i = i + 1;
    meuvet(i) = sqrt(inc);
end
-----------------------------------
[l c] = size(mat);
matnova = zeros(l, c);
for i = 1:l
    for j = 1:c
        matnova(i, j) = sign(mat(i, j));
    end
end
```

O MATLAB possui um código de verificação de funções integrado que pode detectar possíveis problemas em *scripts* e funções. Considere, por exemplo, o seguinte *script* que estende um vetor dentro de um laço:

codigoruim.m

```
for j = 1:4
    vet(j) = j
end
```

O código de verificação da função indicará isso, bem como a boa prática de programação de suprimir a saída nos *scripts*:

```
>> checkcode('codigoruim')
L 2 (C 5-7): The variable 'vet' appears to change size on every
  loop iteration (within a script). Consider preallocating for
  speed.
L 2 (C 12): Terminate statement with semicolon to suppress
  Output (within a script).
```

As mesmas informações são mostradas no Code Analyser Report (Relatório do Analisador de Código), que podem ser produzidos dentro do MATLAB para um arquivo (*script* ou função) ou para todos os arquivos de código dentro de uma pasta. Clicar na seta para baixo do Current Folder (Pasta Atual) e, em seguida, escolher Relatórios e, em seguida, o Code Analyser Report (Relatório do Analisador de Código) verificará o código de todos os arquivos na pasta atual. Ao visualizar um arquivo no Editor, clique na seta para baixo e, em seguida, em Show Code Analyser Report (Exibir Relatório do Analisador de Código) para obter um relatório apenas sobre esse arquivo.

5.5 CRONOMETRAGEM

O MATLAB possui funções internas que determinam quanto tempo leva para executar o código. Um conjunto de funções relacionadas é **tic**/**toc**. Essas funções são colocadas em torno do código e imprimem o tempo necessário para o código executar. Essencialmente, a função **tic** ativa um timer (cronômetro) e depois **toc** avalia o timer e imprime o resultado. Aqui está um *script* que ilustra essas funções.

fortictoc.m

```
tic
minhasoma = 0;
for i = 1:20000000
    minhasoma = minhasoma + i;
end
toc
```

```
>> fortictoc
Elapsed time is 0.087294 seconds.
```

Nota

Ao usar funções de temporização, como **tic**/**toc**, esteja ciente de que outros processos em execução em segundo plano (por exemplo, qualquer navegador da Web) afeta a velocidade do seu código.

Aqui está um exemplo de um *script* que demonstra quanto a pré-alocação de um vetor acelera o código.

tictocprealoc.m

```
% Isso mostra a diferença de tempo entre
% pré-alocando um vetor versus não

clear
disp('Sem pré-alocação')
tic
for i = 1:10000
    x(i) = sqrt(i);
end
toc
disp('Com pré-alocação')
tic
y = zeros(1, 10000);
for i = 1:10000
    y(i) = sqrt(i);
end
toc
```

```
>> tictocprealoc
Sem pré-alocação
```

```
Elapsed time is 0.005070 seconds.
Com pré-alocação
Elapsed time is 0.000273 seconds.
```

PERGUNTA RÁPIDA!

A pré-alocação pode acelerar o código, mas para pré-alocar é necessário conhecer o tamanho desejado. E se você não souber o tamanho final de um vetor (ou matriz)? Isso significa que você tem que estendê-lo em vez de pré-alocar?

Resposta

Se você souber o tamanho máximo possível, poderá pré-alocar para um tamanho maior que o necessário e excluir os elementos "não usados". Para fazer isso, você teria que contar o número de elementos que são realmente usados. Por exemplo, se você tiver um vetor vet que tenha sido pré-alocado e uma variável $contador$ que armazene o número de elementos que foram realmente usados, isso reduzirá os elementos desnecessários:

```
vet = vet(1:contador)
```

O MATLAB também possui um Profiler que irá gerar relatórios detalhados sobre o tempo de execução dos códigos. Em versões mais recentes do MATLAB, no Editor, clique em Run and Time; Isso trará um relatório no Profile Viewer (Visualizador de Perfil). Escolha o nome da função para ver um relatório muito detalhado, incluindo um Code Analyzer Report (Relatório do Analisador de Código). Na Janela de Comandos, isso pode ser acessado usando **profile on**, **profile off** e **profile viewer**.

```
>> plofile on
>> tictocprealoc
Sem pré-alocação
Elapsed time is 0.047721 seconds.
Pré-alocação
Elapsed time is 0.040621 seconds.
>> profile viewer
>> profile off
```

Explorar Outros Recursos Interessantes

- Explore o que acontece quando você usa uma matriz em vez de um vetor para especificar o intervalo em um laço **for**. Por exemplo,

  ```
  for i = mat
      disp(i)
  end
  ```

  Tente adivinhar antes de investigar!

- Experimente a função **pause** em laços.

- Investigue a função **vectorize**.
- As funções **tic** e **toc** estão no tópico de ajuda **timefun**. Digite **help timefun** para investigar algumas das outras funções de temporização. ■

Resumo

Armadilhas Comuns

- Esquecer de inicializar uma soma corrente ou variável contador com 0.
- Esquecer de inicializar uma variável de produto corrente com 1.
- Nos casos em que os laços são necessários, não perceber que, se uma ação é necessária para cada linha de uma matriz, o laço externo deve itera sobre as linhas (e se uma ação for necessária para cada coluna, o laço externo deve iterar sobre as colunas).
- Não perceber que é possível que a ação de um laço **while** nunca seja executada.
- Não verificar erros de entrada de um programa.
- Vetorizar o código sempre que possível. Se não é necessário usar laços no MATLAB, não faça!
- Esquecer que a **subplot** numera os gráficos em linha em vez de em coluna.
- Não perceber que a função de subplot cria apenas uma matriz dentro da Janela de Figura. Cada parte dessa matriz deve ser preenchida com um gráfico, usando qualquer tipo de função de plotagem.

Diretrizes de Estilo de Programação

- Usar laços para repetição apenas quando necessário:
  instruções **for** como laços contados
  instruções **while** como laços condicionais.
- Não usar i ou j para nomes de variáveis iteradoras se o uso das constantes internas i e j for desejado.
- Recuar a ação de laços.
- Se a variável do laço está sendo usada para especificar quantas vezes a ação do laço deve ser executada, use o operador de dois pontos $1{:}\,n$ onde n é o número de vezes que a ação deve ser executada.
- Pré-alocar vetores e matrizes sempre que possível (quando o tamanho é conhecido antecipadamente).
- Quando os dados são lidos em um laço, armazene-os apenas em um array, se for necessário acessar os valores de dados individuais novamente. ■

Palavras Reservadas do MATLAB
for
while
end

Funções e Comandos do MATLAB	
subplot profile	checkcode
factorial	tic / toc

Exercícios

1. Escreva um laço **for** que imprima uma coluna de números reais de 1.5 a 3.1 em passos de 0.2.

2. Na Janela de Comandos, escreva um laço **for** que itere os inteiros de 32 a 255. Para cada um, mostre o caractere correspondente da codificação de caracteres.

3. Crie um vetor x que tenha números inteiros de 1 a 10 e defina um vetor y igual a x. Plote esta linha reta. Agora, adicione ruído aos pontos de dados criando um novo vetor $y2$ que armazene os valores de y mais ou menos 0.25. Plote a linha reta e também estes pontos com ruído.

4. Escreva um *script* $belezamatematica$ que produza a seguinte saída. O *script* deve iterar de 1 a 9 para produzir as expressões à esquerda, executar a operação especificada para obter os resultados mostrados à direita e imprimir exatamente no formato mostrado aqui.

```
>> belezamatematica
1 x 8 + 1 = 9
12 x 8 + 2 = 98
123 x 8 + 3 = 987
1234 x 8 + 4 = 9876
12345 x 8 + 5 = 98765
123456 x 8 + 6 = 987654
1234567 x 8 + 7 = 9876543
12345678 x 8 + 8 = 98765432
123456789 x 8 + 9 = 987654321
```

5. Solicite ao usuário um número inteiro n e imprima "Amo essas coisas!" n vezes.

6. Quando faria diferença um laço **for** definido como $for\ i = 1{:}4$ versus $for\ i = [3\ 5\ 2\ 6]$, e quando isso não faria diferença?

7. Escreva uma função $somapassos2$ que calcule e retorne a soma de 1 para n em passos de 2, onde n é um argumento passado para a função. Por exemplo, se 11 for passado, ele retornará 1 + 3 + 5 + 7 + 9 + 11. Faça isso usando um laço for. Chamar a função ficará assim:

```
>> somapassos2(11)
ans =
   36
```

8. Escreva uma função $produtopor2$ que receberá um valor de um inteiro positivo n e calculará e retornará o produto dos inteiros ímpares de 1 a n (ou de 1 a $n-1$, se n for par). Use um laço **for**.

9. Escreva um *script* que irá:

   - gerar um inteiro aleatório na faixa inclusiva de 2 a 5
   - executar um laço muitas vezes para
     - solicitar ao usuário um número
     - imprimir a soma dos números digitados até o momento, com uma casa decimal.

10. As vendas (em milhões) de duas divisões diferentes de uma empresa para os quatro trimestres de 2017 são armazenadas em variáveis vetores, como as seguintes:

```
div1 = [4.2 3.8 3.7 3.8];
div2 = [2.5 2.7 3.1 3.3];
```

Usando a **subplot**, mostre, lado a lado, os valores de vendas das duas divisões. Em um gráfico, compare as duas divisões.

11. Escreva um *script* que carregará dados de um arquivo para uma matriz. Primeiro, crie o arquivo de dados e verifique se há o mesmo número de valores em todas as linhas do arquivo, para que ele possa ser carregado em uma matriz. Usando um laço for, ele usará a **subplot** para cada linha da matriz e plotará os números de cada elemento da linha na Janela de Fgura.

12. Com uma matriz, quando poderia:
    - seu laço externo iterar sobre as linhas
    - seu laço externo iterar sobre as colunas
    - não importar qual é o laço externo e qual é o laço interno?

13. Escreva um *script* que imprima a seguinte tabela de multiplicação:

```
1
2    4
3    6    9
4    8   12   16
5   10   15   20   25
```

14. Execute este *script* e surpreenda-se com os resultados! Você pode tentar mais pontos para obter uma imagem mais clara, mas pode demorar um pouco para ser executado.

```
clear
clf
x = rand;
y = rand;
plot(x, y)
hold on
for isso = 1:10000
    escolha = round(rand*2);
    if escolha == 0
        x = x/2;
        y = y/2;
    elseif escolha == 1
        x = (x+1)/2;
        y = y/2;
    else
        x = (x+0.5)/2;
        y = (y+1)/2;
    end
    plot(x, y)
    hold on
end
```

15. Uma máquina corta N pedaços de um tubo. Após cada corte, cada pedaço de tubo é pesado e seu comprimento é medido; esses dois valores são então armazenados em um arquivo chamado tubo.dat (primeiro o peso e depois o comprimento em cada linha do arquivo). Ignorando as unidades, o peso deve estar entre 2.1 e 2.3, inclusive, e o comprimento deve estar entre 10.3 e 10.4, inclusive. O seguinte é apenas o começo do que será um longo script para trabalhar com esses dados. Por enquanto, o *script* contará quantas rejeições existem. Uma rejeição é qualquer pedaço de cano que tenha um peso e/ou comprimento inválido. Como um exemplo simples, se N é 3 (significando três linhas no arquivo) e o arquivo é armazenado:

```
2.14   10.30
2.32   10.36
2.20   10.35
```

há apenas um rejeitado – o segundo, porque pesa demais. O *script* imprimiria:

```
Houve 1 rejeição.
```

16. Existem muitas aplicações de processamento de sinal. Tensões, correntes e sons são exemplos de sinais que são estudados em diversas disciplinas, como engenharia biomédica, acústica e telecomunicações. A amostragem de pontos de dados discretos de um sinal contínuo é um conceito importante.

    Um engenheiro de som gravou um sinal sonoro de um microfone. O sinal sonoro foi "amostrado", significando que valores em intervalos discretos foram registrados (ao invés de um sinal sonoro contínuo). As unidades de cada amostra de dados são volts. O microfone não estava ligado o tempo todo, portanto, as amostras de dados que estão abaixo de um determinado limite são consideradas como valores de dados que eram amostras quando o microfone não estava ligado e, portanto, não eram amostras de dados válidas. O engenheiro de som gostaria de saber a voltagem média do sinal sonoro. Escreva um script que solicitará ao usuário o limite e o número de amostras de dados e, em seguida, as amostras de dados individuais. O programa imprimirá a média e uma contagem das amostras de dados VALID ou uma mensagem de erro se não houver amostras de dados válidas. Um exemplo de como a entrada e a saída ficariam na Janela de Comandos é mostrado a seguir.

```
Por favor, insira o limite abaixo do qual as amostras serão
consideradas inválidas: 3.0
Por favor, digite o número de amostras de dados a inserir: 6
Por favor, insira uma amostra de dados: 0.4
Por favor, insira uma amostra de dados: 5.5
Por favor, insira uma amostra de dados: 5.0
Por favor, insira uma amostra de dados: 6.2
Por favor, insira uma amostra de dados: 0.3
Por favor, insira uma amostra de dados: 5.4

A média das 4 amostras de dados válidas é 5.53 volts.
```

Nota
Na ausência de amostras de dados válidas, o programa imprimiria uma mensagem de erro em vez da última linha mostrada.

17. Acompanhe o *script* a seguir para descobrir qual será o resultado e, em seguida, digite-o no MATLAB para verificar os resultados.

```
contador = 0;
numero = 8;
while numero > 3
    numero = numero - 2;
    fprintf('numero é %d\n', numero)
    contador = contador + 1;
end
fprintf('contador é %d\n', contador)
```

18. Trace isso para descobrir qual será o resultado e, em seguida, digite-o no MATLAB para verificar os resultados.

```
contador = 0;
numero = 8;
while numero > 3
    fprintf('numero é %d\n', numero)
    numero = numero - 2;
    contador = contador + 1;
end
fprintf(contador é %d\n', conta)
```

19. O inverso da constante matemática *e* pode ser aproximado como segue:

$$\frac{1}{e} \approx \left(1 - \frac{1}{n}\right)^n$$

Escreva um *script* que itere os valores de n até que a diferença entre a aproximação e o valor real seja menor que 0.0001. O *script* deve imprimir o valor interno de $e - 1$ e com aproximação de quatro casas decimais, além de imprimir o valor de n necessário para tal precisão.

20. Escreva um *script* (por exemplo, chamado $encontraomeu$) que solicitará ao usuário números inteiros mínimos e máximos e, em seguida, outro número inteiro que é a escolha do usuário no intervalo do mínimo ao máximo. O *script* gerará números inteiros aleatórios no intervalo do mínimo ao máximo até que seja gerada uma correspondência para a escolha do usuário. O *script* imprimirá quantos números inteiros aleatórios tiveram que ser gerados até que uma correspondência para a escolha do usuário seja encontrada. Por exemplo, a execução deste *script* pode resultar nesta saída:

```
>> encontreomeu
Por favor, insira seu valor mínimo: -2
Por favor, insira seu valor máximo: 3
Agora digite sua escolha neste intervalo: 0
Demorou 3 tentativas para gerar seu número
```

21. Escreva um *script* $ecoaletras$ que solicitará ao usuário letras do alfabeto e as ecoará até que o usuário digite um caractere que não seja uma letra do alfabeto. Nesse ponto, o *script* imprimirá a não-letra e uma contagem de quantas letras foram inseridas. Aqui estão alguns exemplos de execução deste *script*:

```
>> ecoaletras
Digite uma letra: T
Obrigado, você digitou um T
Digite uma letra: a
Obrigado, você digitou um a
Digite uma letra: 8
8 não é uma letra
Você digitou 2 letras

>> ecoaletras
Digite uma letra:!
! não é uma letra
Você digitou 0 letras
```

22. Uma nevasca é uma enorme tempestade de neve. As definições variam, mas, para os nossos propósitos, vamos supor que uma nevasca é caracterizada por ventos de 30 mph, ou mais, e soprando neve que leva a visibilidade de 0.5 milhas ou menos, sustentada por pelo menos 4 horas. Os dados de uma tempestade um dia foram armazenados em um arquivo $seguetempest.dat$. Existem 24 linhas no arquivo e uma para cada hora do dia. Cada linha no arquivo tem a velocidade do vento e a visibilidade em um local. Crie um arquivo de amostras de dados. Leia esses dados do arquivo e determine se as condições para nevasca ocorreram durante esse dia ou não.

23. Dado o seguinte laço:

```
while x < 10
    ação
end
```

- Para quais valores da variável x a ação do laço seria ignorada inteiramente?
- Se a variável x for inicializada para ter o valor de 5 antes do laço, o que a ação teria que incluir para que isso não fosse um laço infinito?

24. Na termodinâmica, a eficiência de Carnot é a máxima eficiência possível de um motor térmico operando entre dois reservatórios de diferentes temperaturas. A eficiência de Carnot é dada como

$$\eta = 1 - \frac{T_C}{T_H}$$

onde T_C e T_H são as temperaturas absolutas nos reservatórios frio e quente, respectivamente. Escreva um *script* que indique ao usuário as temperaturas dos dois reservatórios em Kelvin e imprima a eficiência Carnot correspondente com três casas decimais. O *script* deve verificar a entrada do usuário, pois a temperatura absoluta não pode ser ≤ 0. O *script* também deve trocar os valores de temperatura se T_H for menor que T_C.

25. Escreva um *script* que usará a função **menu** para apresentar ao usuário as opções para as funções “fix”, “floor” e “ceil”. Verificação de erro por repetição do laço para exibir o menu até que o usuário aperte um dos botões (um erro pode ocorrer se o usuário clicar no "X" na caixa de menu, em vez de pressionar um dos botões). Em seguida, gere um número

aleatório e imprima o resultado da função escolhida pelo usuário, recebendo esse número (por exemplo, **fix (5)**).

26. Escreva um *script* chamado prtemps que solicitará do usuário um valor máximo em Celsius, no intervalo de e16 a 20; faça verificação de erros para se certificar de que está nesse intervalo. Em seguida, imprima uma tabela mostrando graus Fahrenheit e graus Celsius até que este máximo seja atingido. O primeiro valor que exceder o máximo não deve ser impresso. A tabela deve começar em 0 graus Fahrenheit e incrementar em 5 graus Fahrenheit até o máximo (em Celsius) ser atingido. Ambas as temperaturas devem ser impressas com uma largura de campo de 6 e uma casa decimal. A fórmula é $C = 5/9\ (F - 32)$.

27. Escreva um laço **for** que imprima os elementos de uma variável vetor em formato de sentença, independentemente do comprimento do vetor. Por exemplo, se este for o vetor:

```
>> vet = [5.5 11 3.45];
```

este seria o resultado:

```
O elemento 1 é 5.50.
O elemento 2 é 11.00.
O elemento 3 é 3.45.
```

28. Escreva uma função que receberá uma matriz como um argumento de entrada e calculará e retornará a média geral de todos os números na matriz. Use laços, não funções internas, para calcular a média.

29. Crie uma matriz 3 x 5. Execute cada um dos seguintes procedimentos usando laços (se for necessário, se necessário):
    - encontra o valor máximo em cada coluna
    - encontra o valor máximo em cada linha
    - encontre o valor máximo na matriz inteira.

30. Crie um vetor de 5 inteiros aleatórios, cada um na faixa inclusiva de e10 a 10. Execute cada um dos seguintes procedimentos usando laços (com instruções if, se necessário):
    - subtrair 3 de cada elemento
    - contar quantos são positivos
    - obter o valor absoluto de cada elemento
    - encontrar o máximo.

31. O código a seguir foi escrito por alguém que não sabe como usar o MATLAB de maneira eficiente. Reescreva isso como uma única instrução que realizará exatamente a mesma coisa para uma variável matriz (por exemplo, vetorize esse código):

```
[l c] = size(mat);
for i = 1:l
    for j = 1:c
        mat(i, j) = mat(i, j)*2;
    end
end
```

32. Vetorize este código! Escreva uma instrução de atribuição que realizará exatamente a mesma coisa que o código fornecido (suponha que a variável vet tenha sido inicializada):

```
resultado = 0;
for i = 1:length(vet)
    resultado = resultado + vet(i);
end
```

33. Vetorize este código! Escreva uma instrução de atribuição que realizará exatamente a mesma coisa que o código fornecido (suponha que a variável vet tenha sido inicializada):

```
vnovo = zeros(size(vet));
meuprod = 1;
for i = 1:length(vet)
    meuprod = meuprod * vet(i);
    vnovo(i) = meuprod;
end
vnovo % Nota: isto é apenas para mostrar o valor
```

34. Vetorize este código; escreva uma instrução de atribuição que irá realizar a mesma coisa:

```
minhavar = 0;
[l c] = size(mat);
for i = 1:l
    for j = 1:c
        minhavar = minhavar + mat(i, j);
    end
end
minhavar % Nota apenas para exibir o conteúdo do minhavar
```

35. Vetorize este código:

```
n = 3;
x = zeros(n);
y = x;
for i = 1:n
    x(:, i) = i;
    y(i, :) = i;
end
```

36. O seguinte código MATLAB cria um vetor v, que consiste nos índices de todos os elementos em um vetor x que são maiores que 0:

```
v = [];
for i = 1:length(x)
    if x(i) > 0
        v = [v i];
    end
end
```

Escreva uma instrução de atribuição que irá realizar exatamente a mesma coisa usando **find**.

37. Escreva um *script* que solicitará ao usuário uma nota de teste e faça verificação de erros até que o usuário insira uma nota de teste válida. O *script* irá então imprimir a nota. Para

este caso, as notas válidas estão na faixa de 0 a 10 em passos de 0.5. Faça isso criando um vetor de notas válidas e, em seguida, use **any** ou **all** na condição do laço **while**.

38. Qual é mais rápido: usando **false** ou usando **logical(0)** para pré-alocar uma matriz com todos os elementos zeros **logical**? Escreva um *script* para testar isso.

39. Qual é mais rápido: usando uma instrução **switch** ou usando um **if-else** aninhado? Escreva um *script* para testar isso.

40. O fator de resfriamento do vento (WCF) mede a sensação de frio com uma determinada temperatura do ar T (em graus Fahrenheit) e velocidade do vento V (em milhas por hora). Uma fórmula para o WCF é

$$\mathrm{WCF} = 35.7 + 0.6\,\mathrm{T} - 35.7\left(\mathrm{V}^{0.16}\right) + 0.43\,\mathrm{T}\left(\mathrm{V}^{0.16}\right)$$

    Escreva uma função para receber a temperatura e a velocidade do vento como argumentos de entrada e retorne o WCF. Usando laços, imprima uma tabela mostrando os fatores de resfriamento para temperaturas variando de –20 a 55 em passos de 5, e velocidades de vento variando de 0 a 55 em passos de 5. Chame a função para calcular cada fator de resfriamento do vento.

41. Em vez de imprimir os WCFs no problema anterior, crie uma matriz de WCFs e grave-os em um arquivo. Use o método de programação usando laços aninhados.

42. Vetorize a solução para o Exercício 41 usando **meshgrid**.

43. A função **pascal(n)** retorna uma matriz $n\,x\,n$ feita a partir do triângulo de Pascal. Investigue essa função interna e, em seguida, escreva sua própria.

44. Escreva um *script* que solicitará ao usuário N inteiros e, em seguida, escreva os números positivos (≥ 0) em um arquivo ASCII chamado $pos.dat$ e os números negativos em um arquivo ASCII chamado $neg.dat$. Faça verificação de erros de para se certificar de que o usuário insere N inteiros.

45. Escreva um *script* para somar dois números de 30 dígitos e imprima o resultado. Isso não é tão fácil quanto parece no início porque os tipos inteiros podem não ser capazes de armazenar um valor tão grande. Uma maneira de manipular inteiros grandes é armazená-los em vetores, onde cada elemento do vetor armazena um dígito do inteiro. Seu script deve inicializar dois inteiros de 30 dígitos, armazenando cada um em um vetor e, em seguida, adicionar esses inteiros, também armazenando o resultado em um vetor. Crie os números originais usando a função **randi**. Dica: adicione dois números no papel primeiro e preste atenção ao que você faz!

46. Escreva um programa “Jogo Adivinhe Meu Número”. O programa gera um inteiro aleatório em um intervalo especificado e o usuário (o jogador) tem que adivinhar o número. O programa permite ao usuário jogar quantas vezes quiser; na conclusão de cada jogo, o programa pergunta se o jogador quer jogar novamente.

    O algoritmo básico é o seguinte.

    1. O programa inicia imprimindo instruções na tela.
    2. Para cada jogo:
        - o programa gera um novo número inteiro aleatório no intervalo de MIN a MAX. Tratar MIN e MAX como constantes; Comece por inicializá-los para 1 e 100

- executa um laço para solicitar ao jogador um palpite até que o jogador adivinhe corretamente o número inteiro
  - para cada palpite, o programa imprime se o palpite do jogador era muito baixo, muito alto ou correto.

Na conclusão (quando o inteiro foi adivinhado):

- imprimir o número total de palpites para esse jogo.
- imprimir uma mensagem sobre o desempenho do jogador nesse jogo (por exemplo, o jogador demorou demais para adivinhar o número, o jogador estava incrível etc.); Para fazer isso, você terá que decidir sobre intervalos para suas mensagens e fornecer uma justificativa para sua decisão em um comentário no programa.

3. Depois de todos os jogos terem sido jogados, imprima um resumo mostrando a média do número de palpites.

47. Um trocador de CD permite que você carregue mais de um CD. Muitos deles têm botões aleatórios, que permitem reproduzir faixas aleatórias de um CD especificado ou reproduzir faixas aleatórias de CDs aleatórios. Você deve simular uma lista de reprodução de um trocador de CD, usando a função **randi**. O trocador de CD que vamos simular pode carregar três CDs diferentes. Você deve assumir que três CDs foram carregados. Para começar, o programa deve "decidir" quantas faixas existem em cada um dos três CDs, gerando inteiros aleatórios no intervalo de MIN a MAX. Você decide os valores de MIN e MAX. (Veja algum CD. Quantas faixas eles têm? O que é um intervalo razoável?). O programa imprimirá o número de faixas em cada CD. Em seguida, o programa perguntará ao usuário por sua faixa favorita; o usuário deve especificar em qual faixa e em qual CD está. Em seguida, o programa gerará uma "lista de reprodução" das N faixas aleatórias que serão reproduzidas, onde N é um número inteiro. Para cada uma das N músicas, o programa primeiro escolhe aleatoriamente um dos três CDs e escolhe aleatoriamente uma das faixas desse CD. Por fim, o programa imprimirá se a faixa favorita do usuário foi reproduzida ou não. A saída do programa será algo como isso, dependendo dos números inteiros aleatórios gerados e da entrada do usuário:

```
Existem 15 faixas no CD 1.
Existem 22 faixas no CD 2.
Existem 13 faixas no CD 3.
Qual sua faixa favorita?
Por favor, digite o número do CD: 4
Desculpe, não é um CD válido.
Por favor, digite o número do CD: 1
Por favor, insira o número da faixa: 17
Desculpe, essa não é uma faixa válida no CD 1.
Por favor, insira o número da faixa: xyz
Desculpe, essa não é uma faixa válida no CD 1.
Por favor, insira o número da faixa: 11

Lista de reprodução:
CD 2 faixa 20
CD 3 faixa 11
CD 3 faixa 8
CD 2 faixa 1
CD 1 faixa 7
CD 3 faixa 8
```

```
CD 1 faixa 3
CD 1 faixa 15
CD 3 faixa 12
CD 1 faixa 6

Desculpe, sua faixa favorita não foi reproduzida.
```

48. Escreva seu próprio código para realizar a multiplicação de matrizes. Lembre-se de que, para multiplicar duas matrizes, as dimensões internas devem ser as mesmas.

$$[A]_{m\,x\,n}\,[B]_{n\,x\,p} = C_{m\,x\,p}$$

Cada elemento na matriz resultante C é obtido por:

$$C_{ij} = \sum_{k=1}^{n} a_{ik}\, b_{kj}$$

Então, são necessários três laços aninhados.

Referência

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