2. CAPÍTULO 2 – VETORES E MATRIZES

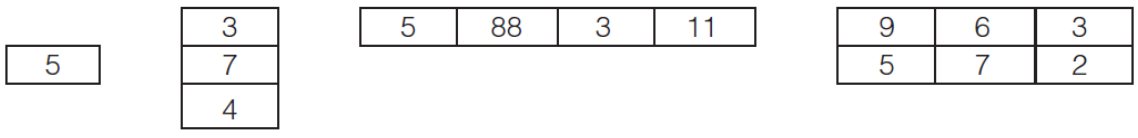
TERMOS CHAVE			CONTEÚDO
vetores	desenrolar uma	multiplicação de	
matrizes	matriz	array	
vetor de linha	indexação linear	divisão de array	
vetor de coluna	coluna de maior	multiplicação de	
escalar	ordem	matriz	
elementos	ordem de coluna	dimensões internas	
array	vetor de variáveis	dimensões externas	
operadores de array	vetor vazio	produto escalar ou	
operador dois-pontos	excluir elementos	produto interno	
iterar	matrizes	produto	
valor do passo	tridimensionais	produto cruzado ou	
concatenar	soma cumulativa	produto externo	
índice	produto cumulativo	vetor lógico	
subscrito	soma corrente	indexação lógica	
vetor de índice	chamadas aninhadas	cruzamento de zero	
transpor	multiplicação escalar		
indexação subscrita	operações de array		

MATLAB® é a abreviação de laboratório de matrizes. Tudo no MATLAB é escrito para trabalhar com vetores e matrizes. Este capítulo apresentará vetores e matrizes. Também serão explicadas operações com vetores, matrizes e funções internas que podem ser usadas para simplificar o código. As operações com matrizes e funções descritas neste capítulo formarão a base para a codificação vetorizada,que será explicada no Capítulo 5.

2.1 VETORES E MATRIZES

Vetores e matrizes são usados para armazenar conjuntos de valores, os quais todos são do mesmo tipo. Uma matriz pode ser visualizada como uma tabela de valores. As dimensões de uma matriz é $l \; x \; c$, onde l é o número de linhas e c é o número decolunas. Isso é lido como "l por c". Um vetor pode ser um ***vetor de linha*** ou um ***vetor de coluna***. Se um vetor tem n elementos, um vetor de linha teria a dimensão $1 \; x \; n$ e um vetor de coluna teria as dimensões $n \; x \; 1$. Um escalar (um valor) tem as dimensões $1 \; x \; 1$. Portanto, vetores e escalares são na verdade, apenas casos especiais de matrizes.

Aqui estão alguns diagramas mostrando, da esquerda para a direita, um escalar, um vetor de coluna,um vetor de linha e uma matriz:



O escalar é $1 \; x \; 1$, o vetor da coluna é $3 \; x \; 1$ (três linhas por coluna), ovetor de linha é $1 \; x \; 4$ (uma linha por quatro colunas), e a matriz é $2 \; x \; 3$ (duas linhas por três colunas). Todos os valores armazenados nessas matrizes são armazenados no que são chamados de ***elementos***.

O MATLAB é escrito para trabalhar com matrizes; o nome MATLAB é a abreviação de laboratório de matrizes. Como o MATLAB é escrito para trabalhar com matrizes, é muito fácil

criar variáveis vetor e matriz, e há muitas operações e funções que podem ser usadas com vetores e matrizes.

Um vetor no MATLAB é equivalente ao que é chamado de ***array unidimensional*** em outras linguagens. Uma matriz é equivalente a um ***array bidimensional***. Usualmente, mesmo em MATLAB, algumas operações que podem ser executadas com vetores ou matrizes são referidas como ***operações de arrays***. O termo array é também frequentemente usado para significar genericamente um vetor ou uma matriz.

Em matemática, a forma geral de uma matriz $m\ xn$ é escrita como:

$$\mathrm{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix} = \mathrm{a}_{ij} \quad \mathrm{i} = 1, \ldots, \mathrm{m}; \quad \mathrm{j} = 1, \ldots, \mathrm{n}$$

2.1.1 Criando Vetores de Linha

Existem várias maneiras de criar variáveis vetores de linha. A maneira mais direta é colocar os valores que você deseja no vetor entre colchetes, separados por espaços ou vírgulas. Por exemplo, ambas as instruções de atribuição cria o mesmo vetor v:

```
>> v = [1 2 3 4]
v =
   1   2   3   4
>> v = [1, 2, 3, 4]
v =
   1   2   3   4
```

Ambos criam uma variável vetor de linha que possui quatro elementos; cada valor é armazenado em um elemento separado no vetor.

2.1.1.1 O Operador Dois-pontos e a Função Linspace

Se, como nos exemplos anteriores, os valores no vetor são espaçados regularmente, o operador ***dois-pontos*** pode ser usado para ***iterar*** esses valores. Por exemplo, $1{:}\,5$ resulta em todos os inteiros de 1 a 5, inclusive:

```
>> vet = 1:5
vet =
   1   2   3   4   5
```

Note que, neste caso, os colchetes [] não são necessários para definir o vetor.

Com o operador dois-pontos, um ***valor de passo*** (***valor de incremento***) também pode ser especificado usando outro dois-pontos, na forma (primeiro:passo:último). Por exemplo, para criar um vetor com todos os elementos inteiros de 1 a 9 em passos de 2:

```
>> vn = 1:2:9
vn =
   1   3   5   7   9
```

PERGUNTA RÁPIDA!

O que acontece se o valor do passo ultrapassar o intervalo definido por $último$, por exemplo:

```
1:2:6
```

Resposta

Isso criaria um vetor contendo 1, 3 e 5. Adicionando 2 ao 5 iria além de 6, então o vetor para em 5; o resultado seria:

```
   1    3    5
```

PERGUNTA RÁPIDA!

Como você pode usar o operador dois-pontos para gerar o vetor mostrado abaixo?

```
   9    7    5    3    1
```

Resposta

```
9:-2:1
```

O valor do passo pode ser um número negativo, portanto, a sequência do resultado está em ordem decrescente (do maior para o menor).

A função **linspace** cria um vetor linearmente espaçado; **linspace(x, y, n)** cria um vetor com n valores no intervalo inclusivo de x a y. Se n éomitido, o padrão é gerar 100 valores. Por exemplo, a instrução seguinte cria um vetor com cinco valores linearmente espaçados entre 3 e 15, incluindo o 3 e o 15:

```
>> ls = linspace(3, 15, 5)
ls =
    3     6     9    12    15
```

Da mesma forma, a função **logspace** cria um vetor logaritmicamente espaçado; **logspace(x, y, n)** cria um vetor com n valores no intervalo inclusivo de $10\hat{}x\ a10\hat{}y$. Se n for omitido, o padrão é gerar 50 valores. Por exemplo:

```
>> logspace(1, 5, 5)
ans =
       10       100      1000     10000    100000
```

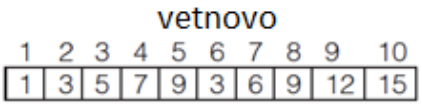
Variáveis vetores também podem ser criadas usando variáveis existentes. Por exemplo,um novo vetor é criado aqui, consistindo, em primeiro lugar, dos valores de vn seguidospor todos os valores de ls:

```
>> vetnovo = [vn ls]
vetnovo =
    1     3     5     7     9     3     6     9    12    15
```

Colocar dois vetores juntos assim para criar um novo é chamado de ***concatenar*** os vetores.

2.1.1.2 Referenciando e Modificando Elementos

Os elementos em um vetor são numerados sequencialmente; cada número de elemento é chamado o ***índice***, ou ***subscrito***. No MATLAB, os índices começam em 1. Normalmente, diagramas de vetores e matrizes mostram os índices. Por exemplo, para a variável $vetnovo$, criada anteriormente, os índices de 1 a 10, dos elementos, são mostrados acima do vetor:



Um elemento específico em um vetor é acessado usando o nome da variável vetor e o índice ou subscrito entre parênteses. Por exemplo, o quinto elemento no vetor $vetnovo$ é o 9.

```
>> vetnovo(5)
ans =
   9
```

A expressão **vetnovo(5)** seria lida como "vetnovo índice 5". Um subconjunto de um vetor, que seria também um vetor, pode ser obtido usando o operador de dois-pontos. Por exemplo, a seguinte instrução obteria do quarto até o sexto elemento do vetor $vetnovo$ e armazenaria o resultado numa variável vetor b:

```
>> b = vetnovo(4:6)
b =
   7   9   3
```

Qualquer vetor pode ser usado para os índices em outro vetor, não apenas um criado usando o operador dois-pontos. Os índices não precisam ser sequenciais. Por exemplo, os seguintes elementos obteriam o primeiro, o décimo e o quinto elementos do vetor $vetnovo$:

```
>> vetnovo([1 10 5])
ans =
    1   15    9
```

O vetor $[1\ 10\ 5]$ é chamado de ***vetor índice***; ele especifica os índices dos elementos, no vetor original, que estão sendo referenciados.

O valor armazenado em um elemento de um vetor pode ser alterado especificando seu índice ou subscrito. Por exemplo, para alterar o segundo elemento do vetor anterior b, para agora armazenar o valor 11, em vez de 9:

```
>> b(2) = 11
b =
    7   11    3
```

Referenciando, numa atribuição, um índice que ainda não existe, um vetor também pode ser estendido. Por exemplo, o comando seguinte cria um vetor vl que possui três elementos. Ao atribuir, a seguir, um valor ao quarto elemento, o vetor é estendido para ter quatro elementos.

```
>> vl = [3 55 11]
vl =
    3   55   11
>> vl(4) = 2
vl =
    3   55   11    2
```

Se houver um intervalo entre o final do vetor e o elemento especificado, os elementos serão preenchidos com 0s. Por exemplo, o comando seguinte estende novamente a variável vl:

```
>> vl(6) = 13
```

```
vl =
     3    55    11     2     0    13
```

Como veremos mais adiante, isso não é muito eficiente, pois pode levar um tempo extra.

PRÁTICA 2.1

Pense no que seria produzido pela seguinte sequência de comandos e expressões e, em seguida, digite-as para verificar suas respostas:

```
vetp = 3:2:10
vetp(2) = 15
vetp(7) = 33
vetp([2:4 7])
linspace(5, 11, 3)
logspace(2, 4, 3)
```

2.1.2 Criando Vetores de Coluna

Uma maneira de criar um vetor de coluna é colocar explicitamente os valores entre colchetes, separados por ponto e vírgula (em vez de vírgulas ou espaços):

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

Não existe uma maneira direta de usar o operador dois-pontos para obter um vetor de coluna. No entanto, qualquer vetor de linha criado usando qualquer método pode ser ***transposto*** para resultar em um vetor de coluna. Em geral, a transposição de uma matriz é uma nova matriz na qual as linhas e colunas são permutadas. Para vetores, a transposição de um vetor de linha resulta em um vetor de coluna e a transposição de um vetor de coluna resulta em um vetor de linha. No MATLAB, o apóstrofo é definido como o ***operador de transposição***.

```
>> vl = 1:3;
>> vc = vl'
vc =
   1
   2
   3
```

2.1.3 Criando Variáveis Matriz

Criar uma variável matriz é simplesmente uma generalização da criação de variáveis vetor de linha e coluna. Ou seja, os valores em uma linha são separados por espaços ou vírgulas e as diferentes linhas são separadas por ponto e vírgula. Por exemplo, a variável matriz mat é criada inserindo-se explicitamentevalores:

```
>> mat = [4 3 1; 2 5 6]
mat =
```

```
   4   3   1
   2   5   6
```

Deve sempre haver o mesmo número de elementos em cada linha. Se você tentar criar uma matriz na qual existam diferentes números de elementos nas linhas, o resultado será uma mensagem de erro, como a seguinte:

```
>> mat = [3 5 7; 1 2]
Error using vertcat
Dimensions of matrices being concatenated are not consistent.
```

Iteradores podem ser usados para gerar os valores nas linhas usando o operador dois-pontos. Por exemplo:

```
>> mat = [2:4; 3:5]
mat =
   2   3   4
   3   4   5
```

As linhas separadas em uma matriz também podem ser especificadas pressionando a tecla Enter após cada linha, em vez de digitar um ponto e vírgula ao inserir os valores da matriz, como em:

```
>> matnova= [2 6 88
33 5 2]
matnova =
    2    6   88
   33    5    2
```

Matrizes de números randômicos (aleatórios) podem ser criadas usando a função **rand**. Se um único valor n for passado para **rand**, uma matriz $n\,x\,n$ será criada; a passagem de dois argumentos especificará o número de linhas e colunas:

```
>> rand(2)
ans =
   0.84210   0.15729
   0.94868   0.63499
>> rand(1,3)
ans =
   0.70607   0.94743   0.19145
```

Matrizes de inteiros aleatórios podem ser geradas usando **randi**; depois que o intervalo é passado, as dimensões da matriz são passadas (novamente, usando um valor n para uma matriz $n\,x\,n$ ou dois valores para as dimensões):

```
>> randi([5, 10], 2)
ans =
   10    6
    5    7
>> randi([10, 30], 2, 3)
ans =
   18   28   11
   10   15   26
```

Observe que o intervalo pode ser especificado para **randi**, mas não para **rand** (os formatos para chamar essas funções são diferentes).

O MATLAB também possui várias funções que criam matrizes especiais. Por exemplo, a função **zeros** cria uma matriz com todos os elementos iguais a zero e a função **ones** cria uma matriz com todos os elementos iguais a um. Como **rand**, um argumento pode ser passado (que será o número de linhas e colunas) ou dois argumentos (primeiro o número de linhas e depois o número de colunas).

```
>> zeros(3)
ans =
   0   0   0
   0   0   0
   0   0   0
>> ones(2,4)
ans =
   1   1   1   1
   1   1   1   1
```

Note que não há função de dois, ou dezenas, ou cinquenta e três – apenas **zeros** e **ones**!

2.1.3.1 Referenciando e Modificando Elementos da Matriz

Para referenciar elementos de matriz, são fornecidos, entre parênteses, os subscritos da linha e, em seguida, da coluna (sempre primeiro a linha e depois a coluna). Por exemplo, isso cria uma variável matriz mat e, em seguida, referencia-se ao valor na segunda linha e terceira coluna de mat:

```
>> mat = [2:4; 3:5]
mat =
   2   3   4
   3   4   5
>> mat(2,3)
ans =
   5
```

Isso é chamado ***indexação subscrita***; ela usa os subscritos de linha e coluna. Também é possível se referir a um subconjunto de uma matriz. Por exemplo, isso se refere à primeira e segunda linhas, segunda e terceira colunas:

```
>> mat(1:2, 2:3)
ans =
   3   4
   4   5
```

Usar apenas um dois-pontos para o subscrito de linha significa referenciar todas as linhas, independentemente de quantas, e usar dois-pontos para o subscrito da coluna, significa referenciar todas as colunas. Por exemplo, o comando a seguir referencia todas as colunas da primeira linha ou, em outras palavras, a primeira linha inteira:

```
>> mat(1, :)
ans =
   2   3   4
```

Este referencia toda a segunda coluna:

```
>> mat(:, 2)
ans =
   3
   4
```

Se um índice único é usado com uma matriz, o MATLAB referencia os elementos da matriz na ordem coluna por coluna. Por exemplo, para a matriz $matint$ criada aqui, os dois primeiros elementos são da primeira coluna e os dois últimos são da segunda coluna:

```
>> matint = [100 77; 28 14]
intmat =
   100    77
    28    14
>> matint(1)
ans =
   100
>> matint(2)
ans =
   28
>> matint(3)
ans =
   77
>> matint(4)
ans =
   14
```

Isso é chamado de ***indexação linear***. Geralmente, este estilo é muito melhor quando se trabalha com matrizes, usando indexação subscrita.

O MATLAB armazena matrizes na memória na ***ordem prioritária de colunas***, ou ***ordem de colunas***, e é por isso que a indexação linear referencia os elementos em ordem de colunas.

Um elemento individual em uma matriz pode ser modificado pela atribuição de um novo valor para isso.

```
>> mat = [2:4; 3:5];
>> mat(1,2) = 11
mat =
    2   11    4
    3    4    5
```

Uma linha ou coluna inteira também pode ser alterada. Por exemplo, a instrução seguinte substitui a segunda linha inteira por valores de um vetor obtido usando o operador dois-pontos.

```
>> mat(2, :) = 5:7
mat =
    2   11    4
    5    6    7
```

Observe que, como toda a linha está sendo modificada, um vetor de linha com o comprimento correto deve ser atribuído. Qualquer subconjunto de uma matriz pode ser modificado desde que o que está sendo atribuído tenha o mesmo número de linhas e colunas que o subconjunto que está sendo modificado.

Para estender uma matriz, um elemento individual não poderia ser adicionado, pois significaria que não haveria mais o mesmo número de elementos em cada linha. No entanto, uma linha ou coluna inteira pode ser adicionada. Por exemplo, o comando seguinte adicionaria uma quarta coluna à matriz:

```
>> mat(:, 4) = [9 2]'
```

```
mat =
    2   11    4    9
    5    6    7    2
```

Assim como vimos com vetores, se existe uma lacuna entre a matriz atual e alinha ou coluna que está sendo adicionada, o MATLAB preencherá com zeros.

```
>> mat(4, :) = 2:2:8
mat =
    2   11    4    9
    5    6    7    2
    0    0    0    0
    2    4    6    8
```

2.1.4 Dimensões

As funções **length** e **size** do MATLAB são usadas para encontrar as dimensões de vetores e matrizes. A função **length** retorna o número de elementos em um vetor. A função **size** retorna o número de linhas e colunas de um vetor ou uma matriz. Por exemplo, o vetor a seguir vet tem quatro elementos, então seu comprimentoé 4. É um vetor de linha, então o tamanho é $1\ x\ 4$.

```
>> vet = -2:1
vet =
  -2  -1   0   1
>> length(vet)
ans =
   4
>> size(vet)
ans =
   1   4
```

Para criar a seguinte variável matriz mat, os iteradores são usados para gerar as duas linhas e, em seguida, a matriz é transposta de modo que tenha três linhas e duas colunas ou, em outras palavras, o tamanho seja $3\ x\ 2$.

```
>> mat = [1:3; 5:7]'
mat =
   1   5
   2   6
   3   7
```

A função **size** retorna o número de linhas e, em seguida, o número de colunas, portanto, para capturar esses valores em variáveis separadas, colocamos um ***vetor de duas variáveis*** à esquerda da atribuição. A variável nl armazena o primeiro valor retornado, que é o número de linhas, e a nc armazena o número de colunas.

```
>> [nl, nc] = size(mat)
nl =
   3
nc =
   2
```

Observe que este exemplo demonstra conceitos muito importantes e únicos no MATLAB: a capacidade de uma função retornar vários valores e a capacidade de ter um vetor de variáveis no lado esquerdo de uma atribuição para armazenar os valores.

Se for chamada apenas como uma expressão, a função **size** retornará os dois valores em um vetor:

```
>> size(mat)
ans =
   3   2
```

Para uma matriz, a função **length** retornará o número de linhas ou o número de colunas, o que for maior (nesse caso, o número de linhas, 3).

```
>> length(mat)
ans =
   3
```

PERGUNTA RÁPIDA!

Como você poderia criar uma matriz de zeros com o mesmo tamanho de outra matriz?

Resposta

Para uma variável matriz mat, a seguinte expressão faria isso:

```
zeros(size(mat))
```

A função **size** retorna o tamanho da matriz, que é então passada para a função **zeros**, que retorna uma matriz de zeros com o mesmo tamanho da matriz. Não é necessário, neste caso, armazenar os valores retornados da função **size** em variáveis.

O MATLAB também possui uma função **numel**, que retorna o número total de elementos em qualquer array (vetor ou matriz):

```
>> vet = 9:-2:1
vet =
   9   7   5   3   1
>> numel(vet)
ans =
   5
>> mat = [3:2:7; 9 33 11]
mat =
    3    5    7
    9   33   11
>> numel(mat)
ans =
   6
```

Para vetores, isso é equivalente ao comprimento do vetor. Para matrizes, é equivalente ao produto do número de linhas pelo número de colunas.

É importante notar que em aplicações de programação, é melhor não assumir que as dimensões de um vetor ou matriz são conhecidas. Em vez disso, para ser geral, use a função **length** ou **numel** para determinar o número de elementos em um vetor e use **length** (e armazene o resultado em duas variáveis) para uma matriz.

O MATLAB também possui uma expressão interna, **end**, que pode ser usada para referenciar o último elemento em um vetor; por exemplo, $v(end)$ é equivalente a $v(length(v))$. Para matrizes, pode-se referenciar a última linha ou coluna. Então, por exemplo, usar **end** para o índice de linha, referenciaria a última linha.

Nesse caso, o elemento referenciado está na primeira coluna da última linha:

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

A utilização de **end** para o índice da coluna referencia o valor da última coluna (por exemplo, a última coluna da segunda linha):

```
>> mat(2, end)
ans =
   5
```

A expressão **end** só pode ser usada como um índice.

2.1.4.1 Alteração de Dimensões

Além do operador de transposição (**'**), o MATLAB possui várias funções internas que alteram as dimensões ou configuração das matrizes, incluindo **reshape**, **fliplr**, **flipud** e **rot90**.

A função **reshape** altera as dimensões de uma matriz. A seguinte variável matriz mat é $3\ x\ 4$ ou, em outras palavras, tem 12 elementos (cada um nointervalo de 1 a 100).

```
>> mat = randi(100, 3, 4)
mat =
   52   67   76   32
   47   88   58   39
    4   39   84   34
```

Estes 12 valores podem ser organizados como uma matriz $2\ x\ 6$, $6\ x\ 2$, $3\ x\ 4$, $4\ x\ 3$, $1\ x\ 12$, ou $12\ x\ 1$. A função **reshape** itera através da matriz, por ordem de colunas. Por exemplo, ao remodelar (usando **reshape**) mat em uma matriz $2\ x\ 6$, os valores da primeira coluna na matriz original $(14, 21\ e\ 20)$ são usadas primeiro, depois os valores da segunda coluna $(61, 28, 20)$ e assim por diante.

```
>> reshape(mat, 2, 6)
ans =
   52    4   88   76   84   39
   47   67   39   58   32   34
```

Note que nestes exemplos mat é inalterada; em vez disso, os resultados são armazenados na variável padrão ans, cada vez.

A função **fliplr** inverte as colunas da matriz da esquerda para a direita (em outras palavras, a coluna mais à esquerda, a primeira coluna, torna-se a última coluna a segunda torna-se a penúltima e assim por diante), e a função **flipud** inverte as linhas da matriz de cima para baixo.

```
>> mat
mat =
    52    67    76    32
    47    88    58    39
     4    39    84    34
>> fliplr(mat)
ans =
    32    76    67    52
    39    58    88    47
    34    84    39     4
>> mat
mat =
    52    67    76    32
    47    88    58    39
     4    39    84    34
>> flipud(mat)
ans =
     4    39    84    34
    47    88    58    39
    52    67    76    32
```

A função **rot90** gira a matriz no sentido anti-horário em 90 graus, portanto, por exemplo, o valor no canto superior direito torna-se o canto superior esquerdo e a última coluna se torna a primeira linha.

```
>> mat
mat =
    52    67    76    32
    47    88    58    39
     4    39    84    34
>> rot90(mat)
ans =
    32    39    34
    76    58    84
    67    88    39
    52    47     4
```

PERGUNTA RÁPIDA!

Existe uma função **rot180**? Existe uma função, como a**rot90**, para girar no sentido horário?

Resposta

Não exatamente, mas um segundo argumento pode ser passado para a função **rot90**, que é um inteiro n; a função irá girar $90 * n$ graus. O inteiro $n$pode ser positivo ou negativo. Por exemplo, se 2 for passado, a função girará a matriz em 180 graus (assim, seria o mesmo que girar o resultado de **rot90** em outros 90 graus).

```
>> mat
mat =
    52    67    76    32
    47    88    58    39
     4    39    84    34
>> rot90(mat, 2)
ans =
    34    84    39     4
```

```
    39    58    88    47
    32    76    67    52
```

Se um número negativo é passado para n, a rotação seria na direção oposta, ou seja, no sentido horário.

```
>> mat
mat =
    52    67    76    32
    47    88    58    39
     4    39    84    34
>> rot90(mat, -1)
ans =
     4    47    52
    39    88    67
    84    58    76
    34    39    32
```

A função **repmat** pode ser usada para criar uma matriz; **repmat(mat, m, n)** cria uma matriz maior, que consiste em uma matriz $m\ x\ n$ de cópias da matrizmat. Por exemplo, aqui está uma matriz randômica $2\ x\ 2$:

```
>> matint = randi(100,2)
matint=
    39    93
     1    38
```

Replicar essa matriz seis vezes como uma matriz $3\ x\ 2$ produziria cópias de $intmat$ desta forma:

matint	matint
matint	matint
matint	matint

```
>> repmat(matint, 3, 2)
ans =
    39    93    39    93
     1    38     1    38
    39    93    39    93
     1    38     1    38
    39    93    39    93
     1    38     1    38
```

2.1.5 Vetores Vazios

Um vetor vazio (um vetor que não armazena valores) pode ser criado usando colchetes vazios:

```
>> vetv = []
vetv =
    []
>> length(vetv)
ans =
    0
```

Nota

Existe uma diferença entre ter uma variável vetor vazia e não existir a variável ainda.

Valores podem ser adicionados a um vetor vazio, concatenando ou adicionando valores ao vetor existente. A instrução a seguir pega o que está atualmente em $vetv$, que não é nada, e adiciona um 4 a ela.

```
>> vetv = [vetv 4]
vetv =
   4
```

A instrução a seguir pega o que está atualmente em $vetv$, que é 4, e adiciona um 11 a ele.

```
>> vetv = [vetv 11]
vetv =
    4   11
```

Isso pode ser repetido continuamente quantas vezes desejar para construir um vetor a partir do nada. Às vezes isso é necessário, embora, geralmente, não seja uma boa ideia se pode ser evitado, pois pode ser um processo demorado.

Vetores vazios também podem ser usados para excluir elementos de vetores. Por exemplo, para remover o terceiro elemento de um vetor, o vetor vazio é atribuído a ele:

```
>> vet = 4:8
vet =
   4   5   6   7   8
>> vet(3) = []
vet =
   4   5   7   8
```

Os elementos neste vetor agora são numerados de 1 a 4.

Subconjuntos de um vetor também podem ser excluídos. Por exemplo:

```
>> vet = 3:10
vet =
    3    4    5    6    7    8    9   10
>> vet(2:4) = []
vet =
    3    7    8    9   10
```

Elementos individuais não podem ser excluídos das matrizes, pois as matrizes sempre precisam ter o mesmo número de elementos em cada linha.

```
>> mat = [7 9 8; 4 6 5]
mat =
   7   9   8
   4   6   5
>> mat(1, 2) = [];
Subscripted assignment dimension mismatch.
```

No entanto, linhas ou colunas inteiras podem ser excluídas de uma matriz. Por exemplo, para excluir a segunda coluna:

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

Além disso, se a indexação linear é usada com uma matriz para excluir um elemento, a matriz será remodelada em um vetor de linha.

```
>> mat = [7 9 8; 4 6 5]
mat =
    7    9    8
    4    6    5
>> mat(3) = []
mat =
    7    4    6    8    5
```

PRÁTICA 2.2

Pense no que seria produzido pela seguinte sequência de declarações e expressões, e depois digite-as para verificar suas respostas.

```
mat = [1:3; 44 9 2; 5:-1:3]
mat(3, 2)
mat(2, :)
size(mat)
mat(:, 4) = [8; 11; 33]
numel(mat)
v = mat(3, :)
v(v(2))
v(1) = []
reshape(mat, 2, 6)
```

2.1.6 Matrizes Tridimensionais

As matrizes que foram mostradas até agora foram bidimensionais; estas matrizes têm linhas e colunas. Matrizes no MATLAB, no entanto, não estão limitadas a duas dimensões. Na verdade, no Capítulo 13, veremos aplicativos de imagem nos quais são usadas ***matrizes tridimensionais***. Para uma matriz tridimensional, imagine uma matriz bidimensional como sendo uma página e, em seguida, a terceira dimensão consiste em mais páginas em cima daquela (então elas são empilhadas no topo uma das outras).

Aqui está um exemplo de criação de uma matriz tridimensional. Primeiro, duas matrizes bidimensionais $camadaum$ e $camadadois$ são criadas; é importante que elas tenham as mesmas dimensões (neste caso, $3\ x\ 5$). Em seguida, estas são postas em "camadas" na matriz tridimensional mat. Note que terminamos com uma matriz que tem duas camadas, cada uma das quais é $3\ x\ 5$. A matriz tridimensional resultante tem dimensões $3\ x\ 5\ x\ 2$.

Nota

Para atribuir as camadas, a matriz mat deve ser criada antes, vazia ([ ]) ou com as dimensões $3\ x\ 5\ x\ 2$ (Por exemplo, com as funções **zeros** ou **ones**).

```
>> camadaum = reshape(1:15, 3, 5)
camadaum =
     1     4     7    10    13
     2     5     8    11    14
     3     6     9    12    15
>> camadadois = fliplr(flipud(camadaum))
camadadois =
    15    12     9     6     3
    14    11     8     5     2
    13    10     7     4     1
>> clear mat
>> mat(:, :, 1) = camadaum
mat =
     1     4     7    10    13
     2     5     8    11    14
     3     6     9    12    15
>> mat(:, :, 2) = camadadois
mat(:,:,1) =
     1     4     7    10    13
     2     5     8    11    14
     3     6     9    12    15
mat(:,:,2) =
    15    12     9     6     3
    14    11     8     5     2
    13    10     7     4     1
>> size(mat)
ans =
    3    5    2
```

Matrizes tridimensionais também podem ser criadas usando as funções **zeros**, **ones** e **rand**, especificando, para começar, três dimensões. Por exemplo, **zeros(2,4,3)** criará uma matriz $2\ x\ 4\ x\ 3$ com todos os elementos zeros.

A menos que especificado de outra forma, no restante deste livro "matrizes" serão assumidas como sendo bidimensionais.

2.2 VETORES E MATRIZES COMO ARGUMENTOS DE FUNÇÃO

No MATLAB, um vetor inteiro ou matriz pode ser passado como um argumento para uma função; a função será avaliada em cada elemento. Isso significa que o resultado será do mesmo tamanho que o argumento de entrada.

Por exemplo, vamos encontrar o seno em radianos de cada elemento de um vetor vet. A função **sin** retornará automaticamente o seno de cada elemento individual e o resultado será um vetor com o mesmo tamanho que o vetor de entrada.

```
>> vet = -2:1
vet =
```

```
  -2  -1   0   1
>> vetseno = sin(vet)
vetseno=
  -0.90930  -0.84147   0.00000   0.84147
```

Para uma matriz, a matriz resultante terá o mesmo tamanho que a matriz do argumento de entrada. Por exemplo, a função **sign** encontrará o sinal de cada elemento da matriz:

```
>> mat = [0 4 -3; -1 0 2]
mat =
   0   4  -3
  -1   0   2
>> sign(mat)
ans =
   0   1  -1
  -1   0   1
```

Funções como **sin** e **sign** podem ter escalares ou arrays (vetores ou matrizes) passados para elas. Há um número de funções que são escritas especificamente para operar com vetores ou com colunas de matrizes; estas incluem as funções **min**, **max**, **sum**, **prod**, **cumsum** e **cumprod**. Os usos dessas funções serão demonstrados primeiro com vetores e depois com matrizes.

Por exemplo, suponha que temos as seguintes variáveis vetor:

```
>> vet1 = 1:5;
>> vet2 = [3 5 8 2];
```

A função **min** retornará o valor mínimo de um vetor e a função **max** retornará o valor máximo.

```
>> min(vet1)
ans =
   1
>> max(vet2)
ans =
   8
```

A função **sum** somará todos os elementos de um vetor. Por exemplo, para $vet1$, ele retornará $1 + 2 + 3 + 4 + 5$ ou 15:

```
>> sum(vet1)
ans =
   15
```

A função **prod** retornará o produto de todos os elementos de um vetor; para, por exemplo, para $vet2$ ele retornará $3 * 5 * 8 * 2$ ou 240:

```
>> prod(vet2)
ans =
   240
```

As funções **cumsum** e **cumprod** retornam a ***soma cumulativa*** ou ***produto cumulativo***, respectivamente. Uma soma cumulativa, ou soma contínua, armazena a soma acumulada até cada elemento, enquanto adiciona os elementos do vetor. Por exemplo, para $vet1$, armazenaria o primeiro elemento, 1, depois3 $(1 + 2)$, depois 6 $(1 + 2 + 3)$,depois10 $(1 + 2 + 3 + 4)$, então, finalmente, 15 $(1 + 2 + 3 + 4 + 5)$. O resultado é um vetor, que tem tantos elementos quanto o vetor argumento de entrada que é passado para ela:

```
>> cumsum(vet1)
ans =
    1    3    6   10   15
>> cumsum(vet2)
ans =
    3    8   16   18
```

A função **cumprod** armazena os produtos cumulativos à medida que multiplica elementos juntos no vetor; novamente, o vetor resultante terá o mesmo tamanho que o vetor de entrada:

```
>> cumprod(vet1)
ans =
     1     2     6    24   120
```

Para matrizes, todas essas funções operam individualmente em cada coluna. Se uma matriz tem dimensões $r \; x \; c$, o resultado para as funções **min**, **max**, **sum** e **prod** será um vetor de linha $1 \text{ x } c$, conforme eles retornam o mínimo, o máximo, a soma ou o produto, respectivamente, para cada coluna. Por exemplo, suponha a seguinte matriz:

```
>> mat = randi([1 20], 3, 5)
mat =
   14   16    1   14   13
   11   15   11   10   20
   11    5    6    2   13
```

A seguir, os resultados das funções **max** e **sum**:

```
>> max(mat)
ans =
   14   16   11   14   20
>> sum(mat)
ans =
   36   36   18   26   46
```

Para encontrar o resultado da função para cada linha, em vez de cada coluna, um método seria transpor a matriz.

```
>> max(mat')
ans =
   16   20   13
>> sum(mat')
ans =
   58   67   37
```

Como colunas é o padrão, elas são consideradas a primeira dimensão. Especificar a segunda dimensão como um argumento para uma dessas funções resultará na operação da função por linha. A sintaxe é ligeiramente diferente; para as funções **sum** e **prod**, este é o segundo argumento, enquanto que para as funções **min** e **max** ele deve ser o terceiro argumento e o segundo argumento deve ser um vetor vazio:

```
>> max(mat, [], 2)
ans =
   16
   20
   13
>> sum(mat, 2)
```

```
ans =
   58
   67
   37
```

Observe a diferença no formato da saída com esses dois métodos (transpondo resultados em vetores de linha, ao passo que especificar a segunda dimensão resulta em vetores de coluna).

PERGUNTA RÁPIDA!

Como essas funções operam em colunas, como podemos obter um resultado geral para a matriz? Por exemplo, como podemos determinar o máximo geral na matriz?

Resposta

Teríamos que obter o máximo do vetor de linha do máximo das colunas, em outras palavras ***aninhar as chamadas*** para a função **max**:

```
>> max(max(mat))
ans =
   20
```

Para as funções **cumsum** e **cumprod**, novamente elas retornam a soma cumulativa ou produto cumulativo de cada coluna. A matriz resultante terá as mesmas dimensões que a matriz de entrada:

```
>> mat
mat =
   14   16    1   14   13
   11   15   11   10   20
   11    5    6    2   13
>> cumsum(mat)
ans =
   14   16    1   14   13
   25   31   12   24   33
   36   36   18   26   46
```

2.3 OPERAÇÕES ESCALARES E DE ARRAY EM VETORES E MATRIZES

Operações numéricas podem ser feitas em vetores inteiros ou matrizes. Por exemplo, digamos que queremos multiplicar todos os elementos de um vetor v por 3.

No MATLAB, podemos simplesmente multiplicar v por 3 e armazenar o resultado de volta em v, em uma declaração de atribuição:

```
>> v = [3 7 2 1];
>> v = v*3
v =
    9   21    6    3
```

Como outro exemplo, podemos dividir cada elemento por 2:

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

```
   1.50000   3.50000   1.00000   0.50000
```

Para multiplicar cada elemento em uma matriz por 2:

```
>> mat = [4:6; 3:-1:1]
mat =
   4   5   6
   3   2   1
>> mat*2
ans =
    8   10   12
    6    4    2
```

Essa operação é chamada de ***multiplicação escalar***. Estamos multiplicando cada elemento em um vetor ou matriz por um escalar (ou dividindo cada elemento em um vetor ou uma matriz por um escalar).

PERGUNTA RÁPIDA!

Não há função de dezenas para criar uma matriz com todos os elementos dezenas, então como poderíamos conseguir isso?

Resposta

Podemos usar a função **ones** e multiplicar por dez, ou a função **zeros** e adicionar dez:

```
>> ones(1,5)*10
ans =
   10   10   10   10   10
>> zeros(2)+10
ans =
   10   10
   10   10
```

As ***operações de array*** são operações que são executadas em vetores ou matrizes, termo por termo ou elemento a elemento. Isso significa que dois arrays (vetores ou matrizes) devem ter o mesmo tamanho, para começar. Os exemplos a seguir demonstram o uso dos operadores de adição e subtração de matriz.

```
>> v1 = 2:5
v1 =
   2   3   4   5
>> v2 = [33 11 5 1]
v2 =
   33   11    5    1
>> v1+v2
ans =
   35   14    9    6
>> mata = [5:8; 9:-2:3]
mata =
   5   6   7   8
   9   7   5   3
>> matb = reshape(1:8, 2, 4)
matb =
```

```
   1   3   5   7
   2   4   6   8
>> mata-matb
ans =
   4   3   2   1
   7   3  -1  -5
```

No entanto, para qualquer operação baseada em multiplicação (que significa multiplicação, divisão e exponenciação), um ponto deve ser colocado na frente do operador para operações de arrays. Por exemplo, para o operador de exponenciação, deve ser usado .^ ao trabalhar com vetores e matrizes, em vez de apenas o operador ^. Calcular o quadrado um vetor, por exemplo, significa multiplicar cada elemento por si mesmo, portanto o operador .^deve ser usado.

```
>> v = [3 7 2 1];
>> v^2
Error using ^
Inputs must be a scalar and a square matrix.
To compute elementwise POWER, use POWER (.^) instead.
>> v.^2
ans =
    9   49    4    1
```

Da mesma forma, o operador .∗ Deve ser usado para multiplicação de arrays e ./ ou .\ para divisão de arrays. Os exemplos a seguir demonstram a multiplicação de arrays e a divisão de arrays.

```
>> v1 = 2:5
v1 =
   2   3   4   5
>> v2 = [33 11 5 1]
v2 =
   33   11    5    1
>> v1.*v2
ans =
   66   33   20    5
>> mata = [5:8; 9:-2:3]
mata =
   5   6   7   8
   9   7   5   3
>> matb = reshape(1:8, 2, 4)
matb =
   1   3   5   7
   2   4   6   8
>> mata./matb
ans =
   5.00000   2.00000   1.40000   1.14286
   4.50000   1.75000   0.83333   0.37500
```

Os operadores .^, .∗, ./ e .\ são chamados de operadores de arrays e são usados para multiplicar ou dividir vetores ou matrizes de mesmo tamanho, elemento a elemento. Observe que a multiplicação de matrizes é uma operação muito diferente e será abordada na próxima seção.

PRÁTICA 2.3

Crie uma variável vetor e subtraia 3 de cada elemento dela.
Crie uma variável matriz e divida cada elemento por 3.
Crie uma variável matriz e calcule o quadrado de cada elemento.

2.4 MULTIPLICAÇÃO DE MATRIZES

Multiplicação de matrizes não significa multiplicar termo por termo; não é uma operação de array. A multiplicação de matrizes tem um significado muito específico. Em primeiro lugar, para multiplicar uma matriz A por uma matriz B resultando numa matriz C, o número de colunas de A devem ser iguais ao número de linhas de B. Se a matriz A tiver dimensões $m\ x\ n$, isso significa que a matriz B deve ter dimensões $n\ x\ alguma\ coisa$; que nós vamos chamar de p.

Dizemos que as ***dimensões internas*** (os ns) devem ser as mesmas. O resultado, a matriz C, tem o mesmo número de linhas que A e o mesmo número de colunas que B (isto é, as ***dimensões externas*** $m\ x\ p$). Em notação matemática,

$$[\mathrm{A}]_{m\ x\ n}[\mathrm{B}]_{m\ x\ p} = [\mathrm{C}]_{m\ x\ p}$$

Isso só define o tamanho de C, não como encontrar os elementos de C.

Os elementos da matriz C são definidos como a soma dos produtos elementos correspondentes nas linhas de A e colunas de B, ou, em outras palavras:

$$c_{ij} \sum_{k=1}^{n} a_{ik} b_{kj} \ .$$

No exemplo a seguir A, é $2\ x\ 3$ e B é $3\ x\ 4$; as dimensões internas sãoambos são3, então a multiplicação da matriz A*B é possível (note que B*A não seria possível). C terá como tamanho as dimensões externas $2\ x\ 4$. Os elementos de C são obtidos usando o somatório que acabamos de descrever. A primeira linha de C é obtida usando a primeira linha de A e em sucessão as colunas de B. Por exemplo, $C(1,1)$ é $3 * 1 + 8 * 4 + 0 * 0$ ou 35. $C(1,2)$ é $3 * 2 + 8 * 5 + 0 * 2$ ou 46.

$$\underset{\mathrm{A}}{\begin{bmatrix} 3 & 8 & 0 \\ 1 & 2 & 5 \end{bmatrix}} * \underset{\mathrm{B}}{\begin{bmatrix} 1 & 2 & 3 & 1 \\ 4 & 5 & 1 & 2 \\ 0 & 2 & 3 & 0 \end{bmatrix}} = \underset{\mathrm{C}}{\begin{bmatrix} 35 & 46 & 17 & 19 \\ 9 & 22 & 20 & 5 \end{bmatrix}}$$

No MATLAB, o operador $*$ executará a multiplicação de matrizes:

```
>> A = [3 8 0; 1 2 5];
>> B = [1 2 3 1; 4 5 1 2; 0 2 3 0];
>> C = A*B
C =
   35   46   17   19
    9   22   20    5
```

PRÁTICA 2.4

Quando duas matrizes têm as mesmas dimensões, são quadradas, tanto a multiplicação de arrays quanto a de matrizes podem ser executadas com elas. Para as duas matrizes seguintes, execute, à mão, A.*B, A*B e B*A e, em seguida, verifique os resultados no MATLAB.

$$\underset{A}{\begin{bmatrix} 1 & 4 \\ 3 & 3 \end{bmatrix}} \quad \underset{B}{\begin{bmatrix} 1 & 2 \\ -1 & 0 \end{bmatrix}}$$

2.4.1 Multiplicação de Matriz por Vetores

Como os vetores são apenas casos especiais de matrizes, as operações com matrizes descritas anteriormente (adição, subtração, multiplicação escalar, multiplicação, transposição) também funcionam com vetores, desde que as dimensões estejam corretas.

Para vetores, já vimos que a transposta de um vetor de linha é um vetor de coluna, e a transposta de um vetor de coluna é um vetor de linha.

Para multiplicar vetores, eles devem ter o mesmo número de elementos, mas um deve ser um vetor de linha e o outro um vetor de coluna. Por exemplo, para um vetor de coluna c um vetor de linha l:

$$\mathrm{c} = \begin{bmatrix} 5 \\ 3 \\ 7 \\ 1 \end{bmatrix} \quad \mathrm{l} = [6 \;\; 2 \;\; 3 \;\; 4]$$

Note que l é $1 \; x \; 4$, e c é $4 \; x \; 1$, então

$$[\mathrm{l}]_{1\,x\,4}[\mathrm{c}]_{4\,x\,1} = [\mathrm{e}]_{1\,x\,1}$$

ou, em outras palavras, um escalar:

$$[6 \;\; 2 \;\; 3 \;\; 4]\begin{bmatrix} 5 \\ 3 \\ 7 \\ 1 \end{bmatrix} = 6^*5 + 2^*3 + 3^*7 + 4^*1 = 61$$

Enquanto $[\mathrm{c}]_{4\,x\,1}[\mathrm{l}]_{1\,x\,4} = [\mathrm{M}]_{4\,x\,4}$, ou em outras palavras, uma matriz $4 \; x \; 4$:

$$\begin{bmatrix} 5 \\ 3 \\ 7 \\ 1 \end{bmatrix}[6 \;\; 2 \;\; 3 \;\; 4] = \begin{bmatrix} 30 & 10 & 15 & 20 \\ 18 & 6 & 9 & 12 \\ 42 & 14 & 21 & 28 \\ 6 & 2 & 3 & 4 \end{bmatrix}$$

No MATLAB, essas operações são realizadas usando o operador *, que é o operador de multiplicação de matrizes. Primeiro, o vetor de colunas c e o vetor de linhas l são criados.

```
>> c = [5 3 7 1]';
```

```
>> l = [6 2 3 4];
>> l*c
ans =
   61
>> c*l
ans =
   30   10   15   20
   18    6    9   12
   42   14   21   28
    6    2    3    4
```

Existem também operações específicas para vetores: o ***produto escalar*** e ***produto cruzado***. O ***produto escalar***, ou ***produto interno***, de dois vetores a e b é escrito como $a * b$ e é definido como

$$a_1b_1+a_2b_2+a_3b_3+\ldots+a_nb_n=\sum_{i=1}^{n} a_ib_i$$

onde, tanto a quanto b têm n elementos, e a_i e b_i representam elementos dos vetores. Em outras palavras, isso é como a multiplicação de matrizes ao multiplicar um vetor de linha a por um vetor de coluna b e o resultado é um escalar. Isso pode ser realizado usando o operador * e transpondo o segundo vetor, ou usando a função **dot** no MATLAB:

```
>> vet1 = [4 2 5 1];
>> vet2 = [3 6 1 2];
>> vet1*vet2'
ans =
   31
>> dot(vet1, vet2)
ans =
   31
```

O ***produto cruzado*** ou ***produto externo*** $a\ x\ b$ de dois vetores a e b é definido apenas quando ambos a e b têm três elementos. Este pode ser definido como uma multiplicação de matrizes, de uma matriz composta dos elementos de a, de uma maneira particular mostrada aqui, e o vetor de coluna b.

$$a\ x\ b = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix} = [a_2b_3 - a_3b_2, a_3b_1 - a_1b_3, a_1b_2 - a_2b_1]$$

O MATLAB tem uma função interna para fazer isso.

```
>> vet1 = [4 2 5];
>> vet2 = [3 6 1];
>> cross(vet1, vet2)
ans =
  -28   11   18
```

2.5 VETORES LÓGICOS

Os vetores lógicos usam expressões relacionais que resultam em valores lógicos **verdadeiro/falso**.

2.5.1 Expressões Relacionais com Vetores e Matrizes

Operadores relacionais podem ser usados com vetores e matrizes. Por exemplo, vamos dizer que haja um vetor vet, e queremos comparar todos os elementos no vetor com 5 para determinar se cada elemento de vet é maior que 5 ou não. O resultado seria um vetor (com o mesmo tamanho que o original) com valores lógicos **verdadeiro** ou **falso**.

```
>> vet = [5 9 3 4 6 11];
>> ehmaior = vet > 5
ehmaior=
  0   1   0   0   1   1
```

Observe que isso cria um vetor que consiste em todos os valores lógicos **verdadeiros** ou **falsos**. Embora o resultado seja um vetor de **uns** e **zeros** e operações numéricas possam ser feitas com o vetor $ehmaior$, seu tipo é **logical** em vez de **double**.

```
>> doubles = ehmaior + 5
doubles=
   5   6   5   5   6   6
>> whos
  Name       Size             Bytes  Class

  doubles    1x6                 48  double array
  ehmaior    1x6                  6  logical array
  vet        1x6                 48  double array
```

Para determinar quantos elementos no vetor vetsão maiores que 5, a função **sum** pode ser usada no vetor resultante $ehmaior$:

```
>> sum(ehmaior)
ans =
   3
```

O que fizemos foi criar um **vetor lógico** $ehmaior$. Este vetor lógico pode ser usado para indexar o vetor original. Por exemplo, se apenas são desejados os elementos do vetor que são maiores que 5:

```
>> vet(ehmaior)
ans =
    9    6   11
```

Isso é chamado de **indexação lógica**. Apenas são retornados os elementos de vet para os quais os elementos correspondentes, no vetor lógico $ehmaior$, é **verdadeiro lógico**.

PERGUNTA RÁPIDA!

Por que o seguinte não funciona?

```
>> vet = [5 9 3 4 6 11];
>> v = [0 1 0 0 1 1];
>> vet(v)
Subscript indices must either be real positive integers or
 logicals.
```

Resposta

A diferença entre o vetor neste exemplo e o vetor$ehmaior$, éque $ehmaior$é um vetor de valores lógicos ($1s$ e $0s$ do tipo **logical**), enquanto $[0\ 1\ 0\ 0\ 1\ 1]$, por padrão, é um vetor de valores do tipo **double**. **Somente $1s$ e $0s$ do tipo logical podem ser usados para indexar um vetor**.

Então, modelando o tipo da variável v, funcionaria:

```
>> v = logical(v);
>> vet(v)
ans =
    9     6    11
```

Para criar um vetor ou matriz com todos os elementos do tipo **lógical** ($1s$ ou $0s$), as funções **true** e **false** podem ser usadas.

```
>> false(2)
ans =
  0  0
  0  0
>> true(1,5)
ans =
  1  1  1  1  1
```

As funções **true** e **false** são mais rápidas e gerenciam a memória com mais eficiência do que usar **logical** com as funções **zeros** ou **ones**.

2.5.2 Funções Lógicas Internas

Existem funções internas no MATLAB, que são úteis em conjunto com vetores ou matrizes do tipo **logical**; duas dessas funções são **any** e **all**. A função **any** retorna **verdadeiro lógico** se algum elemento de um vetor representa **verdadeiro**, e **falso**, se não. A função **all** retorna **verdadeiro lógico** somente se todos os elementos representarem **verdadeiro**. Aqui estão alguns exemplos.

```
>> any(ehmaior)
ans =
  1
>> all(true(1,3))
ans =
  1
```

Para a seguinte variável $vet2$, alguns elementos, mas não todos, são **verdadeiros**; consequentemente, **any** retorna **verdadeiro**, mas **all** retorna **falso**.

```
>> vet2 = logical([1 1 0 1])
vet2 =
  1  1  0  1
>> any(vet2)
ans =
  1
>> all(vet2)
ans =
  0
```

A função **find** retorna os índices de um vetor que atendem aos critérios fornecidos. Por exemplo, para encontrar todos os elementos em um vetor maiores que 5:

```
>> vet = [5 3 6 7 2]
vet =
   5   3   6   7   2
>> find(vet > 5)
ans =
   3   4
```

Para matrizes, a função **find** usará a **indexação linear** ao retornar os índices que atendem aos critérios especificados. Por exemplo:

```
>> mata = randi(10, 2, 4)
mata =
    7   10    6    9
    7    6    4    6
>> find(mata == 6)
ans =
   4
   5
   8
```

Para ambos, vetores e matrizes, será retornado um vetor vazio se nenhum elemento corresponder ao critério. Por exemplo,

```
>> find(mata == 11)
ans =
   Empty matrix: 0-by-1
```

A função **isequal** é útil na comparação de matrizes. No MATLAB, usando o operador de igualdade com matrizes retornará 1 ou 0 para cada elemento; a função **all** poderia então ser usada na matriz resultante para determinar se todos os elementos eram iguais ou não. A função interna **isequal** também realiza isso:

```
>> vet1 = [1 3 -4 2 99];
>> vet2 = [1 2 -4 3 99];
>> vet1 == vet2
ans =
  1  0  1  0  1
>> all(vet1 == vet2)
ans =
  0
>> isequal(vet1, vet2)
ans =
  0
```

No entanto, uma diferença é que, se as duas matrizes não tiverem as mesmas dimensões, a função **isequal** retornará 0**lógico**, enquanto o uso do operador de igualdade resultará em uma mensagem de erro.

PERGUNTA RÁPIDA!

Se tivermos um vetorvet que, erroneamente, armazena valores negativos, como podemos eliminar esses valores negativos?

Resposta

Um método é determinar onde eles estão e excluir esses elementos:

```
>> vet = [11 -5 33 2 8 -4 25];
>> neg = find(vet < 0)
neg =
   2   6
>> vet(neg) = []
vet =
   11   33    2    8   25
```

Alternativamente, podemos apenas usar um vetor lógico em vez de **find**:

```
>> vet = [11 -5 33 2 8 -4 25];
>> vet(vet < 0) = []
vet =
   11   33    2    8   25
```

PRÁTICA 2.5

Modifique o resultado visto na *Pergunta Rápida* anterior! Em vez de excluir os elementos "indesejados", retenha apenas os "desejados". (Dica: faça isso de duas maneiras, usando **find** e usando um vetor lógico com a expressão $vet >= 0$).

O MATLAB também possui operadores **ou** e **e** que trabalham com arrays, elemento a elemento:

Operador	**Significado**
|	ou para arrays(elemento a elemento)
&	e para arrays (elemento a elemento)

Estes operadores irão comparar, elemento a elemento, quaisquer dois vetores ou matrizes, desde que sejam do mesmo tamanho e retornarão um vetor ou matriz do mesmo tamanho de $1s$ e $0s$do tipo **logical**. Os operadores || e && são usados apenas com escalares, não com matrizes. Por exemplo:

```
>> v1 = logical([1 0 1 1]);
>> v2 = logical([0 0 1 0]);
>> v1&v2
ans =
  0  0  1  0
>> v1|v2
ans =
  1  0  1  1
>> v1&&v2
Operands to the || and && operators must be convertible to
 logical scalar values.
```

Assim como com os operadores numéricos, é importante conhecer as regras de precedência do operador. A Tabela 2.1 mostra as regras para os operadores que foram vistos até o momento, na ordem de precedência.

Tabela 2.1 Regras de Precedência de Operadores	
Operador	Precedência
parênteses ( )	Mais alta
transposta e potência ', ^, .^	
unário, negação – (não ~)	
multiplicação, divisão *, /, \, .*, ./, .\	
adição, subtração +, –	
relacional<, <=, >, >===, ~=	
e (elemento a elemento) &	
ou (elemento a elemento) |	
e (escalares) &&	
ou (escalares) ||	
atribuição=	Mais baixa

2.6 APLICAÇÕES: ASFUNÇÕES DIFF E MESHGRID

Duas funções que podem ser úteis no trabalho com aplicações com vetores e matrizes são **diff** e **meshgrid**. A função **diff** retorna as diferenças entre elementos consecutivos em um vetor. Por exemplo,

```
>> diff([4 7 15 32])
ans =
    3    8   17
>> diff([4 7 2 32])
ans =
    3   -5   30
```

Para um vetor v com comprimento n, o comprimento de **diff(v)** será $n-1$. Para uma matriz, a função **diff** irá operar em cada coluna.

```
>> mat = randi(20, 2, 3)
mat =
   11    7   16
   18    3   11
>> diff(mat)
ans =
    7   -4   -5
```

Por exemplo, um vetor que armazena sinais pode conter valores positivos e negativos. (Para simplificar, não vamos, no entanto, assumir valores zero). Para muitas aplicações, é útil encontrar os ***cruzamentos de zero***, ou onde o sinal passa de positivo para negativo ou vice-versa. Isso pode ser feito usando as funções **sign**, **diff** e **find**.

```
>> vet = [0.2 -0.1 -0.2 -0.1 0.1 0.3 -0.2];
>> vs = sign(vet)
vs=
    1   -1   -1   -1    1    1   -1
>> dvs = diff(vs)
dvs=
   -2    0    0    2    0   -2
>> find(dvs ~= 0)
ans =
    1    4    6
```

Isso mostra que os cruzamentos de sinal estão entre os elementos 1 e 2, 4 e 5 e, 6 e 7.

A função **meshgrid** pode especificar as coordenadas xey dos pontos nas imagens, ou pode ser usada para calcular funções de duas variáveis x e y. Ela recebe como argumentos de entrada dois vetores e retorna como argumentos de saída duas matrizes que especificam valores x e y separadamente. Por exemplo, as coordenadas xey de uma imagem $2x3$ seriam especificadas pelas coordenadas:

```
(1,1) (2,1) (3,1)
(1,2) (2,2) (3,2)
```

As matrizes que especificam separadamente as coordenadas são criadas pela função **meshgrid**, onde x itera de 1 a 3 e y itera de1 a 2:

```
>> [x y] = meshgrid(1:3,1:2)
x =
   1   2   3
   1   2   3
y =
   1   1   1
   2   2   2
```

Como outro exemplo, digamos que queremos avaliar uma função f de duas variáveis x e y:

$$f(x,y) = 2 * x + y$$

onde$x$ varia de 1 a 4 e y varia de 1 a 3. Podemos fazer issocriando matrizes xey, usando **meshgrid** e, em seguida, a expressão para calcular f usa multiplicação escalar e adição de array.

```
>> [x y] = meshgrid(1:4,1:3)
x =
   1   2   3   4
   1   2   3   4
   1   2   3   4
y =
   1   1   1   1
   2   2   2   2
   3   3   3   3
>> f = 2*x+y
f =
    3    5    7    9
    4    6    8   10
    5    7    9   11
```

Explorar Outros Recursos Interessantes

- Existem muitas funções que criam matrizes especiais (por exemplo, **hilb** (para uma matriz de Hilbert), **magic** e **pascal**).
- A função **gallery**, que pode retornar muitos tipos diferentes de matrizes detestes para problemas.
- A função **ndims** para encontrar o número de dimensões de um argumento.
- A função **shiftdim**.
- A função **circshift**. Como você pode usá-la para mudar um vetor de linha, resultando em outro vetor de linha?

- Como remodelar (reshape) uma matriz tridimensional.
- Passar matrizes tridimensionais para funções. Por exemplo, se você passar uma matriz $3\ x\ 5\ x\ 2$ para a função soma, qual seriamas dimensões (tamanho) do resultado?

Resumo

Armadilhas Comuns

- Tentar criar uma matriz que não tenha o mesmo número de valores em cada linha.
- Confundir multiplicação de array e multiplicação de matrizes. Operações com arrays, incluindo multiplicação, divisão e exponenciação, são executadas elemento a elemento (assim as matrizes devem ter o mesmo tamanho); os operadores são .*, ./, .\ e .^. Para que a multiplicação de matrizes seja possível,as dimensões internas devem concordar e o operador é *.
- Tentar usar uma matriz de $1s$ e $0s$ do tipo **double** para indexar em um array (em vez disso, deve ser usado o tipo **logical**).
- Esquecer que para operações baseadas em multiplicação, o ponto deve ser usado no operador. Em outras palavras, para multiplicar, dividir, dividir ou elevar a um expoente, elemento a elemento, os operadores são .*, ./, .\ e .^.
- Tentar usar || ou && com matrizes. Sempre use | e & ao trabalhar com matrizes; || e && são usados apenas com escalares.

Diretrizes de Estilo de Programação

- Se possível, não tente estender vetores ou matrizes, pois não é muito eficiente.
- Não use apenas um único índice ao referenciar elementos em uma matriz; em vez disso, use os subscritos de linha e coluna (use indexação de subscritos em vez de indexação linear).
- Para ser geral, nunca assuma que as dimensões de qualquer array (vetor ou matriz) são conhecidas. Em vez disso, use a função **length** ou **numel** para determinar o número de elementos em um vetor e a função **size** para uma matriz:

```
len = length(vet);

[nl, nc] = size(mat);
```

- Use **true** em vez de **logical(1)** e **false** em vez de **logical(0)**, especialmente ao criar vetores ou matrizes.

Funções e Comandos do MATLAB			
linspace	end	max	any
logspace	reshape	sum	all
zeros	fliplr	prod	find
ones	flipud	cumsum	isequal
length	rot90	cumprod	diff
size	repmat	dot	meshgrid
numel	min	cross	

Operadores do MATLAB	
dois-pontos:	multiplicação de matrizes *
transposta '	ou elementar para matrizes |
operadores de arrays .^, .*, ./, .\	e elementar para matrizes &

Exercícios

1. Usando o operador dois-pontos, crie os seguintes vetores de linha:

```
   2   3   4   5   6   7
1.1000   1.3000   1.5000   1.7000
   8   6   4   2
```

2. Dê a expressão MATLAB que criaria um vetor (em uma variável chamada vet) de 50 elementos que variam, igualmente espaçados, de 0 a 2π:

3. Escreva uma expressão usando **linspace** que resultará no mesmo que $2{:}\ 0.2{:}\ 3$.

4. Usando o operador dois-pontos e também a função **linspace**, crie os seguintes vetores de linha:

```
  -5  -4  -3  -2  -1
   5   7   9
   8   6   4
```

5. Crie uma variável $meufim$ que armazena um inteiro randômico no intervalo inclusivo de 5 a 9. Usando o operador dois-pontos, crie um vetor que itera de 1 a $meufim$em passos de 3.

6. Usando o operador dois-pontos e o operador de transposição, crie um vetor de coluna que tenha os valores -1 a 1 em passos de 0.5.

7. Escreva uma expressão que referencie apenas aos elementos de índices ímpares em um vetor, independentemente do comprimento do vetor. Teste sua expressão em vetores que tenham um número ímpar e um número par de elementos.

8. Encontre uma maneira $eficiente$ de gerar a seguinte matriz:

```
mat =
    7     8     9    10
   12    10     8     6
```

   Então, dê expressões que, para a matriz mat,

   - referencie o elemento na primeira linha, terceira coluna
   - referencie toda a segunda linha
   - referencie as duas primeiras colunas.

9. Gere uma variável matrizmat,$2\ x\ 4$. Verifique se o número de elementos é o produto do número de linhas e colunas.

10. Gere uma variável matriz mat, $2\ x\ 4$. Substitua a primeira linha por $1{:}\ 4$. Substitua a terceira coluna (você decide com quais valores).

11. Gerar uma matriz $2\ x\ 3$ de números randômicos

    - reais, com cada elemento no intervalo $(0, 1)$
    - reais, com cada elemento no intervalo $(0, 10)$
    - inteiros, com cada elemento no intervalo inclusivo de 5 a 20.

12. Crie uma variável $linhas$ que seja um inteiro randômico no intervalo inclusivo de 1 a 5. Crie uma variável $colunas$ que seja um inteiro randômico no intervalo inclusivo de 1 a 5.

Crie uma matriz com todos os elementos zeros com as dimensões dadas pelos valores de $linhas$ e $colunas$.

13. A função interna **clock** retorna um vetor que contém seis elementos: os três primeiros são a data atual (ano, mês, dia) e os três últimos representam a hora atual em horas, minutos e segundos. Os segundos é um número real, mas todos os outros são inteiros. Armazene o resultado de **clock** em uma variável chamada $meurelogio$. Em seguida, armazene os três primeiros elementos dessa variável em uma variável $hoje$ e os três últimos elementos em uma variável $agora$. Use a função **fix** na variável vetor *agora* para obter apenas a parte inteira da hora atual.

14. Crie uma variável matriz mat. Encontre quantas expressões você puder para referenciar o último elemento da matriz, sem assumir que você sabe quantos elementos, linhas ou colunas ela possui (ou seja, torne suas expressões gerais).

15. Crie uma variável vetor vet. Encontre quantas expressões você puder para referenciar o último elemento do vetor, sem assumir que você sabe quantos elementos ele possui (ou seja, torne suas expressões gerais).

16. Crie uma variável matrizmat,$2\ x\ 3$. Passe esta variável matriz para cada uma das seguintes funções e certifique-se de entender o resultado: **fliplr**, **flipud** e **rot90**. De quantas maneiras diferentes você pode remodelá-la (usando **reshape**)?

17. Crie uma matriz $3\ x\ 5$ de números reais randômicos. Exclua a terceira linha.

18. Crie uma matriz tridimensional e obtenha seu tamanho (usando **size**).

19. Crie uma matriz tridimensional com as dimensões $2\ x\ 4\ x\ 3$, em que a primeira "camada" é toda de 0s, a segunda é toda de 1s e a terceira é de 5s.

20. Crie um vetor x que consista em 20valores igualmente espaçados no intervalo de $-p$ a $+p$. Crie um vetor y que seja **sin(x)**.

21. Crie uma matriz $3\ x\ 5$ de inteiros randômicos, cada um no intervalo inclusivo de -5 a 5. Obtenha o sinal de cada elemento.

22. Crie uma matriz *4 x 6* de inteiros randômicos, cada um no intervalo inclusivo de -5 a 5; armazene-o em uma variável. Crie outra matriz que armazene para cada elemento o valor absoluto do elemento correspondente na matriz original.

23. Encontre a soma $3 + 5 + 7 + 9 + 11$.

24. Encontre a soma dos primeiros n termos da série harmônica, onde n é um inteiromaior que um.

$$1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \frac{1}{5} + \cdots$$

25. Encontre a soma dos cinco primeiros termos da série geométrica

$$1 + \frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \cdots$$

26. Encontre a soma a seguir, criando primeiro, vetores para os numeradores e para os denominadores:

$$\frac{3}{1}+\frac{5}{2}+\frac{7}{3}+\frac{9}{4}$$

27. Crie uma matriz e encontre o produto de cada linha e coluna usando **prod**.

28. Crie um vetor $1\ x\ 6$ de números inteiros randômicos, cada um no intervalo inclusivo de 1 a 20. Use funções internas para encontrar os valores mínimo e máximo no vetor. Também crie um vetor de somas cumulativas usando **cumsum**.

29. Escreva uma expressão relacional para uma variável vetor que irá verificar se o último valor em um vetor criado por **cumsum** é o mesmo que o resultado retornado por **sum**.

30. Crie um vetor de cinco inteiros randômicos, cada um no intervalo inclusivo de -10 a 10. Execute cada um dos seguintes procedimentos para esse vetor:

    - subtraia 3 de cada elemento
    - conte quantos são positivos
    - obtenha o valor absoluto de cada elemento
    - encontre o valor máximo.

31. Crie uma matriz $3\ x\ 5$. Execute um dos seguintes procedimentos para a matriz:

    - Encontre o valor máximo em cada coluna.
    - Encontre o valor máximo em cada linha.
    - Encontre o valor máximo na matriz inteira.

32. O valor de $\pi^2/6$ pode ser aproximado pela soma da série

$$1+\frac{1}{3}+\frac{1}{9}+\frac{1}{27}+\cdots$$

    onde acima, é mostrado os quatro primeiros termos da série. Crie variáveis para testar isso.

33. Em uma universidade, os estudantes preenchem formulários de avaliação nos quais a escala é $1-5$. Um é suposto ser o melhor e 5o pior. No entanto, no formulário, a escala foi invertida, de modo que 1 foi o pior e 5 o melhor. Todos os programas de computador que lidam com esses dados esperam que estejam ao contrário. Portanto, os dados precisam ser "invertidos". Por exemplo, se um vetor de resultados de avaliação é:

    ```
    >> avals = [5 3 2 5 5 4 1 2]
    ```

    deve ser realmente `[1 3 4 1 1 2 5 4]`.

34. Um vetor v armazena, para vários funcionários da Corporação de Células de Combustível Verde, as horas que eles trabalharam numa semana, cada uma seguida pelo valor de pagamento por hora. Por exemplo, se a variável armazena

    ```
    >> v
    v =
     33.0000 10.5000 40.0000 18.0000 20.0000  7.5000
    ```

    Isso significa que o primeiro funcionário trabalhou 33 horas a R$10,50 por hora, o segundo trabalhou 40 horas a R$18,00 por hora, e assim por diante. Escreva um código que separe isso em dois vetores: um que armazene as horas trabalhadas e outro que armazene os

valores das horas. Em seguida, use o operador de multiplicação de array para criar um novo vetor, armazenando o pagamento total para cada funcionário.

35. Uma empresa está calibrando alguns instrumentos de medição e mediu, separadamente, o raio e a altura de um cilindro 10 vezes; as medições estão armazenadas nas variáveis vetor$r$ e h. Encontre o volume de cada medição, que é dada por $\pi r^2 h$. Use também,primeiro, indexação lógica para garantir que todas as medidas sejam válidas (> 0).

36. Para as seguintes matrizes A, B e C:

$$\mathrm{A} = \begin{bmatrix} 1 & 4 \\ 3 & 2 \end{bmatrix} \quad \mathrm{B} = \begin{bmatrix} 2 & 1 & 3 \\ 1 & 5 & 6 \\ 3 & 6 & 0 \end{bmatrix} \quad \mathrm{C} = \begin{bmatrix} 3 & 2 & 5 \\ 4 & 1 & 2 \end{bmatrix}$$

- dê o resultado de 3*A
- dê o resultado de A*C
- Existem outras multiplicações de matrizes que podem ser realizadas? Se assim for, liste-as.

37. Para os seguintes vetores e matrizes A, B e C:

$$\mathrm{A} = \begin{bmatrix} 4 & 1 & -1 \\ 2 & 3 & 0 \end{bmatrix} \quad \mathrm{B} = \begin{bmatrix} 1 & 4 \end{bmatrix} \quad \mathrm{C} = \begin{bmatrix} 2 \\ 3 \end{bmatrix}$$

Execute as seguintes operações, se possível. Se não, diga que não pode ser feito!

A*B
B*C
C*B

38. A variável matriz $matchuva$ armazena o total de precipitação em milímetros para alguns distritos, para os anos de $2010 - 2013$. Cada linha tem os valores de precipitação para um determinado distrito. Por exemplo, se $matchuva$ tiver o valor:

```
>> matchuva
matchuva =
   25   33   29   42
   53   44   40   56
      etc.
```

o distrito 1 teve 25 milímetros em 2010, 33 em 2011, etc. Escreva a(s) expressão(ões) que encontrará o número do distrito que teve a maior precipitação total durante todo o período de quatro anos.

39. Gere um vetor de 20 inteiros randômicos, cada um no intervalo de 50 a 100. Crie uma variável $pares$que armazene todos os números pares do vetor e uma variável $impares$que armazene os números ímpares.

40. Suponha que a função **diff** não exista. Escreva sua(s) própria(s) expressão(ões) para realizar a mesma coisa para um vetor.

41. Avalie a função f de duas variáveis x e y, onde x varia de 1 a 2 e y varia de 1 a 5.

$f(x, y) = 3 * x - y$

42. Crie uma variável vetor vet; que pode ter qualquer comprimento. Em seguida, escreva os comandos de atribuição que armazenariam a primeira metade do vetor em uma variável e a segunda metade em outra. Certifique-se de que seus comandos de atribuição sejam gerais e trabalhe se vet tiver um número par ou ímpar de elementos. (Dica: use uma função de arredondamento, tal como **fix**).

    Algumas operações são mais fáceis de fazer se uma matriz for particionada em blocos (em particular, se for realmente grande). O particionamento em blocos também permite a utilização de computação em grade (*grid*) ou paralela, onde as operações são distribuídas por uma grade de computadores.

    Por exemplo, se

$$\mathrm{A} = \begin{bmatrix} 1 & -3 & 2 & 4 \\ 2 & 5 & 0 & 1 \\ -2 & 1 & 5 & -3 \\ -1 & 3 & 1 & 2 \end{bmatrix}$$

    Apode ser particionada em

$$\begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

    onde

$$\mathrm{A}_{11} = \begin{bmatrix} 1 & -3 \\ 2 & 5 \end{bmatrix}, \mathrm{A}_{12} = \begin{bmatrix} 2 & 4 \\ 0 & 1 \end{bmatrix}, \mathrm{A}_{21} = \begin{bmatrix} -2 & 1 \\ -1 & 3 \end{bmatrix}, \mathrm{A}_{22} = \begin{bmatrix} 5 & -3 \\ 1 & 2 \end{bmatrix}.$$

    Se B é do mesmo tamanho,

$$\mathrm{B} = \begin{bmatrix} 2 & 1 & -3 & 0 \\ 1 & 4 & 2 & -1 \\ 0 & -1 & 5 & -2 \\ 1 & 0 & 3 & 2 \end{bmatrix}.$$

    Divida-o em

$$\begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix}.$$

43. Crie as matrizes A e B e particione-as no MATLAB. Mostre que a adição de matriz, a subtração da matriz e a multiplicação escalar podem ser executadas bloco a bloco e concatenadas para o resultado geral.

44. Para multiplicação de matrizes usando os blocos

$$\mathrm{A} * \mathrm{B} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \begin{bmatrix} B_{11} & B_{12} \\ B_{21} & B_{22} \end{bmatrix} = \begin{bmatrix} A_{11}B_{11} + A_{12}B_{21} & A_{11}B_{12} + A_{12}B_{22} \\ A_{21}B_{11} + A_{22}B_{21} & A_{21}B_{12} + A_{22}B_{22} \end{bmatrix}$$

    Faça isso no MATLAB para as matrizes fornecidas.

Referência

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