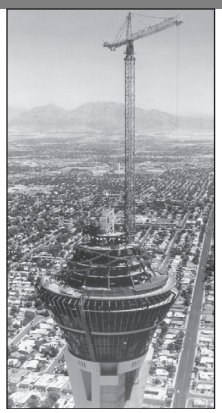

Photo courtesy of Stratosphere Corporation.

Engineering in the 21st Century. . .

Innovative Construction

We tend to remember the great civilizations of the past in part by their public works, such as the Egyptian pyramids and the medieval cathedrals of Europe, which were technically challenging to create. Perhaps it is in our nature to "push the limits," and we admire others who do so. The challenge of innovative construction continues today. As space in our cities becomes scarce, many urban planners prefer to build vertically rather than horizontally. The newest tall buildings push the limits of our abilities, not only in structural design but also in areas that we might not think of, such as elevator design and operation, aerodynamics, and construction techniques. The photo above shows the 1149-ft-high Las Vegas Stratosphere Tower, the tallest observation tower in the United States. It required many innovative techniques in its assembly. The construction crane shown in use is 400 ft tall.

Designers of buildings, bridges, and other structures will use new technologies and new materials, some based on nature's designs. Pound for pound, spider silk is stronger than steel, and structural engineers hope to use cables of synthetic spider silk fibers to build earthquake-resistant suspension bridges. *Smart* structures, which can detect impending failure from cracks and fatigue, are now close to reality, as are *active* structures that incorporate powered devices to counteract wind and other forces. The MATLAB Financial toolbox is useful for financial evaluation of large construction projects, and the MATLAB Partial Differential Equation toolbox can be used for structural design. ■

CHAPTER 2

Numeric, Cell, and Structure Arrays

OUTLINE

One of the strengths of MATLAB is the capability to handle collections of items, called *arrays,* as if they were a single entity. The array-handling feature means that MATLAB programs can be very short.

The array is the basic building block in MATLAB. The following classes of arrays are available in MATLAB 7:

Array						
numeric	character	logical	cell	structure	function handle	Java

So far we have used only numeric arrays, which are arrays containing only numeric values. Within the numeric class are the subclasses *single* (single precision), *double* (double precision), *int8*, *int16*, and *int32* (signed 8-bit, 16-bit, and 32-bit integers), and *uint8*, *uint16*, and *uint32* (unsigned 8-bit, 16-bit, and 32-bit integers). A character array is an array containing strings. The elements of logical

arrays are "true" or "false," which, although represented by the symbols 1 and 0, are not numeric quantities. We will study the logical arrays in Chapter 4. Cell arrays and structure arrays are covered in Sections 2.6 and 2.7. Function handles are treated in Chapter 3. The Java class is not covered in this text.

The first four sections of this chapter treat concepts that are essential to understanding MATLAB and therefore must be covered. Section 2.5 treats polynomial applications. Sections 2.6 and 2.7 introduce two types of arrays that are useful for some specialized applications.

2.1 One- and Two-Dimensional Numeric Arrays

We can represent the location of a point in three-dimensional space by three Cartesian coordinates x, y, and z. These three coordinates specify a *vector* $\mathbf{p}$. (In mathematical text we often use boldface type to indicate vectors.) The set of *unit vectors* $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$, whose lengths are 1 and whose directions coincide with the x, y, and z axes, respectively, can be used to express the vector mathematically as follows: $\mathbf{p} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$. The unit vectors enable us to associate the vector components x, y, z with the proper coordinate axes; therefore, when we write $\mathbf{p} = 5\mathbf{i} + 7\mathbf{j} + 2\mathbf{k}$, we know that the x, y, and z coordinates of the vector are 5, 7, and 2, respectively. We can also write the components in a specific order, separate them with a space, and identify the group with brackets, as follows: [5 7 2]. As long as we agree that the vector components will be written in the order x, y, z, we can use this notation instead of the unit-vector notation. In fact, MATLAB uses this style for vector notation. MATLAB allows us to separate the components with commas for improved readability if we desire so that the equivalent way of writing the preceding vector is [5, 7, 2]. This expression is a *row vector*, which is a horizontal arrangement of the elements.

ROW VECTOR

COLUMN VECTOR

We can also express the vector as a *column vector*, which has a vertical arrangement. A vector can have only one column, or only one row. Thus, a vector is a one-dimensional array. In general, arrays can have more than one column and more than one row.

Creating Vectors in MATLAB

The concept of a vector can be generalized to any number of components. In MATLAB a vector is simply a list of scalars, whose order of appearance in the list might be significant, as it is when specifying xyz coordinates. As another example, suppose we measure the temperature of an object once every hour. We can represent the measurements as a vector, and the 10th element in the list is the temperature measured at the 10th hour.

To create a row vector in MATLAB, you simply type the elements inside a pair of square brackets, separating the elements with a space or a comma. Brackets are required for arrays unless you use the colon operator to create the array. In this case you should not use brackets, but you can optionally use parentheses. The choice between a space or comma is a matter of personal preference, although the chance of an error is less if you use a comma. (You can also use a comma followed by a space for maximum readability.)

TRANSPOSE

To create a column vector, you can separate the elements by semicolons; alternatively, you can create a row vector and then use the *transpose* notation (`'`), which converts a row vector into a column vector, or vice versa. For example:

```
>>g = [3;7;9]
g =
    3
    7
    9
>>g = [3,7,9]'
g =
    3
    7
    9
```

The third way to create a column vector is to type a left bracket (`[`) and the first element, press **Enter**, type the second element, press **Enter**, and so on until you type the last element followed by a right bracket (`]`) and **Enter**. On the screen this sequence looks like

```
>>g = [3
7
9]
g =
    3
    7
    9
```

Note that MATLAB displays row vectors horizontally and column vectors vertically.

You can create vectors by "appending" one vector to another. For example, to create the row vector `u` whose first three columns contain the values of `r = [2,4,20]` and whose fourth, fifth, and sixth columns contain the values of `w = [9,-6,3]`, you type `u = [r,w]`. The result is the vector `u = [2,4,20,9,-6,3]`.

The colon operator (:) easily generates a large vector of regularly spaced elements. Typing

```
>>x = m:q:n
```

creates a vector `x` of values with a spacing `q`. The first value is `m`. The last value is `n` if `m - n` is an integer multiple of `q`. If not, the last value is less than `n`. For example, typing `x = 0:2:8` creates the vector `x = [0,2,4,6,8]`, whereas typing `x = 0:2:7` creates the vector `x = [0,2,4,6]`. To create a row vector `z` consisting of the values from 5 to 8 in steps of 0.1, you type `z = 5:0.1:8`. If the increment `q` is omitted, it is presumed to be 1. Thus `y = -3:2` produces the vector `y = [-3,-2,-1,0,1,2]`.

The increment `q` can be negative. In this case `m` should be greater than `n`. For example, `u = 10:-2:4` produces the vector `[10,8,6,4]`.

The `linspace` command also creates a linearly spaced row vector, but instead you specify the number of values rather than the increment. The syntax is `linspace(x1,x2,n)`, where `x1` and `x2` are the lower and upper limits and `n` is the number of points. For example, `linspace(5,8,31)` is equivalent to `5:0.1:8`. If `n` is omitted, the spacing is 1.

The `logspace` command creates an array of logarithmically spaced elements. Its syntax is `logspace(a,b,n)`, where `n` is the number of points between 10^a and 10^b. For example, `x = logspace(-1,1,4)` produces the vector `x = [0.1000, 0.4642, 2.1544, 10.000]`. If `n` is omitted, the number of points defaults to 50.

Two-Dimensional Arrays

MATRIX

An array having rows and columns is a two-dimensional array that is sometimes called a *matrix*. In mathematical text, if possible, vectors are usually denoted by boldface lowercase letters and matrices by boldface uppercase letters. An example of a matrix having three rows and two columns is

$$\mathrm{M} = \begin{bmatrix} 2 & 5 \\ -3 & 4 \\ -7 & 1 \end{bmatrix}$$

ARRAY SIZE

We refer to the *size* of an array by the number of rows and the number of columns. For example, an array with 3 rows and 2 columns is said to be a 3×2 array. *The number of rows is always stated first!* We sometimes represent a matrix $\mathbf{A}$ as $[a_{ij}]$ to indicate its elements a_{ij}. The subscripts i and j, called *indices*, indicate the row and column location of the element a_{ij}. *The row number must always come first!* For example, the element a_{32} is in row 3, column 2. Two matrices $\mathbf{A}$ and $\mathbf{B}$ are equal if they have the same size and if all their corresponding elements are equal, that is, $a_{ij} = b_{ij}$ for every value of i and j.

Creating Matrices

The most direct way to create a matrix is to type the matrix row by row, separating the elements in a given row with spaces or commas and separating the rows with semicolons. Brackets are required. For example, typing

```
>>A = [2,4,10;16,3,7];
```

creates the following matrix:

$$\mathbf{A} = \begin{bmatrix} 2 & 4 & 10 \\ 16 & 3 & 7 \end{bmatrix}$$

If the matrix has many elements, you can press **Enter** and continue typing on the next line. MATLAB knows you are finished entering the matrix when you type the closing bracket (`]`).

You can append a row vector to another row vector to create either a third row vector or a matrix (if both vectors have the same number of columns). Note the difference between the results given by `[a,b]` and `[a;b]` in the following session:

```
>>a = [1,3,5];
>>b = [7,9,11];
>>c = [a,b]
c =
     1  3  5  7  9  11
>> D = [a;b]
D =
     1  3  5
     7  9  11
```

Matrices and the Transpose Operation

The transpose operation interchanges the rows and columns. In mathematics text we denote this operation by the superscript *T*. For an $m \times n$ matrix $\mathbf{A}$ with m rows and n columns, $\mathbf{A}^T$ (read "A transpose") is an $n \times m$ matrix.

$$\mathbf{A} = \begin{bmatrix} -2 & 6 \\ -3 & 5 \end{bmatrix} \qquad \mathbf{A}^T = \begin{bmatrix} -2 & -3 \\ 6 & 5 \end{bmatrix}$$

If $\mathbf{A}^T = \mathbf{A}$, the matrix $\mathbf{A}$ is *symmetric*. Note that the transpose operation converts a row vector into a column vector, and vice versa.

If the array contains complex elements, the transpose operator (`'`) produces the *complex conjugate transpose;* that is, the resulting elements are the complex conjugates of the original array's transposed elements. Alternatively, you can use the *dot transpose* operator (`.'`) to transpose the array without producing complex conjugate elements, for example, `A.'`. If all the elements are real, the operators `'` and `.'` give the same result.

Array Addressing

Array indices are the row and column numbers of an element in an array and are used to keep track of the array's elements. For example, the notation `v(5)` refers to the fifth element in the vector `v`, and `A(2,3)` refers to the element in row 2, column 3 in the matrix `A`. *The row number is always listed first!* This notation enables you to correct entries in an array without retyping the entire array. For example, to change the element in row 1, column 3 of a matrix **D** to 6, you can type `D(1,3) = 6`.

The colon operator selects individual elements, rows, columns, or "subarrays" of arrays. Here are some examples:

- `v(:)` represents all the row or column elements of the vector `v`.
- `v(2:5)` represents the second through fifth elements; that is `v(2), v(3), v(4), v(5)`.

- `A(:,3)` denotes all the elements in the third *column* of the matrix `A`.
- `A(3,:)` denotes all the elements in the third *row* of `A`.
- `A(:,2:5)` denotes all the elements in the second through fifth columns of `A`.
- `A(2:3,1:3)` denotes all the elements in the second and third rows that are also in the first through third columns.
- `v = A(:)` creates a vector `v` consisting of all the columns of `A` stacked from first to last.
- `A(end,:)` denotes the last row in `A`, and `A(:,end)` denotes the last column.

You can use array indices to extract a smaller array from another array. For example, if you create the array **B**

$$\mathbf{B} = \begin{bmatrix} 2 & 4 & 10 & 13 \\ 16 & 3 & 7 & 18 \\ 8 & 4 & 9 & 25 \\ 3 & 12 & 15 & 17 \end{bmatrix} \tag{2.1–1}$$

by typing

```
>>B = [2,4,10,13;16,3,7,18;8,4,9,25;3,12,15,17];
```

and then type

```
>>C = B(2:3,1:3);
```

you can produce the following array:

$$\mathbf{C} = \begin{bmatrix} 16 & 3 & 7 \\ 8 & 4 & 9 \end{bmatrix}$$

EMPTY ARRAY

The *empty array* contains no elements and is expressed as `[]`. Rows and columns can be deleted by setting the selected row or column equal to the empty array. This step causes the original matrix to collapse to a smaller one. For example, `A(3,:) = []` deletes the third row in `A`, while `A(:,2:4) = []` deletes the second through fourth columns in `A`. Finally, `A([1 4],:) = []` deletes the first and fourth rows of `A`.

Suppose we type `A = [6,9,4;1,5,7]` to define the following matrix:

$$\mathbf{A} = \begin{bmatrix} 6 & 9 & 4 \\ 1 & 5 & 7 \end{bmatrix}$$

Typing `A(1,5) = 3` changes the matrix to

$$\mathbf{A} = \begin{bmatrix} 6 & 9 & 4 & 0 & 3 \\ 1 & 5 & 7 & 0 & 0 \end{bmatrix}$$

Because **A** did not have five columns, its size is automatically expanded to accept the new element in column 5. MATLAB adds zeros to fill out the remaining elements.

MATLAB does not accept negative or zero indices, but you can use negative increments with the colon operator. For example, typing `B = A(:,5:-1:1)` reverses the order of the columns in **A** and produces

$$\mathbf{B} = \begin{bmatrix} 3 & 0 & 4 & 9 & 6 \\ 0 & 0 & 7 & 5 & 1 \end{bmatrix}$$

Suppose that `C = [-4,12,3,5,8]`. Then typing `B(2,:) = C` replaces row 2 of `B` with `C`. Thus **B** becomes

$$\mathbf{B} = \begin{bmatrix} 3 & 0 & 4 & 9 & 6 \\ -4 & 12 & 3 & 5 & 8 \end{bmatrix}$$

Suppose that `D = [3,8,5;4,-6,9]`. Then typing `E = D([2,2,2],:)` repeats row 2 of `D` three times to obtain

$$\mathbf{E} = \begin{bmatrix} 4 & -6 & 9 \\ 4 & -6 & 9 \\ 4 & -6 & 9 \end{bmatrix}$$

Using `clear` to Avoid Errors

You can use the `clear` command to protect yourself from accidentally reusing an array that has the wrong dimension. Even if you set new values for an array, some previous values might still remain. For example, suppose you had previously created the 2×2 array `A = [2, 5; 6, 9]`, and then you create the 5×1 arrays `x = (1:5)'` and `y = (2:6)'`. Note that parentheses are needed here to use the transpose operator. Suppose you now redefine `A` so that its columns will be `x` and `y`. If you then type `A(:,1) = x` to create the first column, MATLAB displays an error message telling you that the number of rows in `A` and `x` must be the same. MATLAB thinks `A` should be a 2×2 matrix because `A` was previously defined to have only two rows and its values remain in memory. The `clear` command wipes `A` and all other variables from memory and avoids this error. To clear `A` only, type `clear A` before typing `A(:,1) = x`.

Some Useful Array Functions

MATLAB has many functions for working with arrays (see Table 2.1–1). Here is a summary of some of the more commonly used functions.

The `max(A)` function returns the algebraically greatest element in **A** if **A** is a vector having all real elements. It returns a row vector containing the greatest elements in each *column* if **A** is a matrix containing all real elements. If *any* of the elements are complex, `max(A)` returns the element that has the largest magnitude. The syntax `[x,k] = max(A)` is similar to `max(A)`, but it stores the maximum values in the row vector **x** and their indices in the row vector **k**.

If `A` and `B` have the same size, `C = max(A,B)` creates an array the same size, having the maximum value from each corresponding location in `A` and `B`. For example, the following **A** and **B** matrices give the **C** matrix shown.

Table 2.1–1 Basic syntax of array functions*

Command	Description
`find(x)`	Computes an array containing the indices of the nonzero elements of the array **x**.
`[u,v,w] = find(A)`	Computes the arrays **u** and **v**, containing the row and column indices of the nonzero elements of the matrix **A**, and the array **w**, containing the values of the nonzero elements. The array **w** may be omitted.
`length(A)`	Computes either the number of elements of **A** if **A** is a vector or the largest value of m or n if **A** is an $m \times n$ matrix.
`linspace(a,b,n)`	Creates a row vector of n regularly spaced values between a and b.
`logspace(a,b,n)`	Creates a row vector of n logarithmically spaced values between a and b.
`max(A)`	Returns the algebraically largest element in **A** if **A** is a vector. Returns a row vector containing the largest elements in each column if **A** is a matrix. If any of the elements are complex, `max(A)` returns the elements that have the largest magnitudes.
`[x,k] = max(A)`	Similar to `max(A)` but stores the maximum values in the row vector **x** and their indices in the row vector **k**.
`min(A)`	Same as `max(A)` but returns minimum values.
`[x,k] = min(A)`	Same as `[x,k] = max(A)` but returns minimum values.
`norm(x)`	Computes a vector's geometric length $\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$.
`size(A)`	Returns a row vector `[m n]` containing the sizes of the $m \times n$ array **A**.
`sort(A)`	Sorts each column of the array **A** in ascending order and returns an array the same size as **A**.
`sum(A)`	Sums the elements in each column of the array **A** and returns a row vector containing the sums.

*Many of these functions have extended syntax. See the text and MATLAB help for more discussion.

$$\mathbf{A} = \begin{bmatrix} 1 & 6 & 4 \\ 3 & 7 & 2 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 3 & 4 & 7 \\ 1 & 5 & 8 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} 3 & 6 & 7 \\ 3 & 7 & 8 \end{bmatrix}$$

The functions `min(A)` and `[x,k] = min(A)` are the same as `max(A)` and `[x,k] = max(A)` except that they return minimum values.

The function `size(A)` returns a row vector `[m n]` containing the sizes of the $m \times n$ array **A**. The `length(A)` function computes either the number of elements of **A** if `A` is a vector or the largest value of m or n if **A** is an $m \times n$ matrix.

For example, if

$$\mathbf{A} = \begin{bmatrix} 6 & 2 \\ -10 & -5 \\ 3 & 0 \end{bmatrix}$$

then `max(A)` returns the vector `[6,2]`; `min(A)` returns the vector `[-10,-5]`; `size(A)` returns `[3,2]`; and `length(A)` returns 3.

The `sum(A)` function sums the elements in each *column* of the array **A** and returns a row vector containing the sums. The `sort(A)` function sorts each *column* of the array **A** in ascending order and returns an array the same size as **A**.

If `A` has one or more complex elements, the `max`, `min`, and `sort` functions act on the absolute values of the elements and return the element that has the largest magnitude.

For example, if

$$\mathbf{A} = \begin{bmatrix} 6 & 2 \\ -10 & -5 \\ 3 + 4i & 0 \end{bmatrix}$$

then `max(A)` returns the vector `[-10,-5]` and `min(A)` returns the vector `[3+4i,0]`. (The magnitude of $3 + 4i$ is 5.)

The sort will be done in descending order if the form `sort(A, 'descend')` is used. The `min`, `max`, and `sort` functions can be made to act on the rows instead of the columns by transposing the array.

The complete syntax of the `sort` function is `sort(A, dim, mode)`, where `dim` selects a dimension along which to sort and `mode` selects the direction of the sort, `'ascend'` for ascending order and `'descend'` for descending order. So, for example, `sort(A,2, 'descend')` would `sort` the elements in each row of **A** in descending order.

The `find(x)` command computes an array containing the indices of the *nonzero* elements of the vector **x**. The syntax `[u,v,w] = find(A)` computes the arrays **u** and **v**, containing the row and column indices of the nonzero elements of the matrix **A**, and the array **w**, containing the values of the nonzero elements. The array **w** may be omitted.

For example, if

$$\mathbf{A} = \begin{bmatrix} 6 & 0 & 3 \\ 0 & 4 & 0 \\ 2 & 7 & 0 \end{bmatrix}$$

then the session

```
>>A = [6, 0, 3; 0, 4, 0; 2, 7, 0];
>>[u, v, w] = find(A)
```

returns the vectors

$$\mathbf{u} = \begin{bmatrix} 1 \\ 3 \\ 2 \\ 3 \\ 1 \end{bmatrix} \qquad \mathbf{v} = \begin{bmatrix} 1 \\ 1 \\ 2 \\ 2 \\ 3 \end{bmatrix} \qquad \mathbf{w} = \begin{bmatrix} 6 \\ 2 \\ 4 \\ 7 \\ 3 \end{bmatrix}$$

The vectors **u** and **v** give the (row, column) indices of the nonzero values, which are listed in **w**. For example, the second entries in **u** and **v** give the indices (3, 1), which specifies the element in row 3, column 1 of **A**, whose value is 2.

These functions are summarized in Table 2.1–1.

Magnitude, Length, and Absolute Value of a Vector

The terms *magnitude, length,* and *absolute value* are often loosely used in everyday language, but you must keep their precise meaning in mind when using MATLAB.

The MATLAB `length` command gives the number of elements in the vector. The *magnitude* of a vector **x** having real elements x_1, x_2, . . . , x_n is a scalar, given by $\sqrt{x_1^2 + x_2^2 + \cdots + x_n^2}$, and is the same as the vector's geometric length. The *absolute value* of a vector **x** is a vector whose elements are the absolute values of the elements of **x**. For example, if `x = [2,-4,5]`, its length is 3; its magnitude is $\sqrt{2^2 + (-4)^2 + 5^2} = 6.7082$; and its absolute value is `[2,4,5]`. The length, magnitude, and absolute value of `x` are computed by `length(x)`, `norm(x)`, and `abs(x)`, respectively.

Test Your Understanding

T2.1–1 For the matrix **B**, find the array that results from the operation `[B;B']`. Use MATLAB to determine what number is in row 5, column 3 of the result.

$$\mathbf{B} = \begin{bmatrix} 2 & 4 & 10 & 13 \\ 16 & 3 & 7 & 18 \\ 8 & 4 & 9 & 25 \\ 3 & 12 & 15 & 17 \end{bmatrix}$$

T2.1–2 For the same matrix **B**, use MATLAB to (a) find the largest and smallest elements in **B** and their indices and (b) sort each column in **B** to create a new matrix **C**.

The Variable Editor

The MATLAB Workspace Browser provides a graphical interface for managing the workspace. You can use it to view, save, and clear workspace variables. It includes the *Variable Editor,* a graphical interface for working with variables, including arrays. To open the Workspace Browser, type `workspace` at the Command window prompt. The browser appears as shown in Figure 2.1–1.

Keep in mind that the Desktop menus are context-sensitive. Thus their contents will change depending on which features of the browser and Variable Editor you are currently using. The Workspace Browser shows the name of each variable, its value, array size, and class. The icon for each variable illustrates its class.

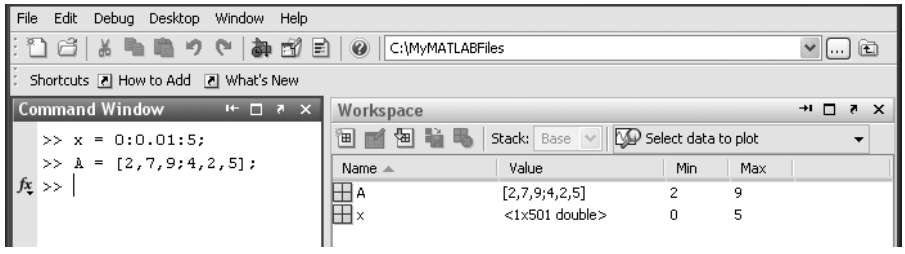

Figure 2.1–1 The Workspace Browser.

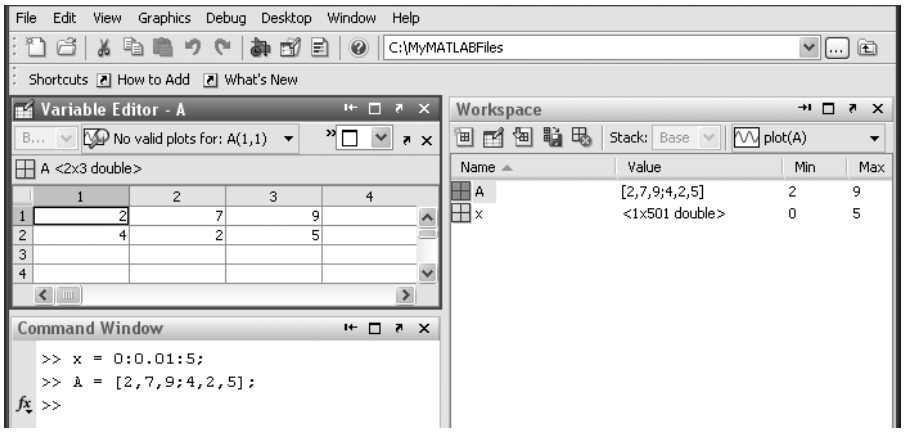

Figure 2.1–2 The Variable Editor.

From the Workspace Browser you can open the Variable Editor to view and edit a visual representation of two-dimensional numeric arrays, with the rows and columns numbered. To open the Variable Editor from the Workspace Browser, double-click on the variable you want to open. The Variable Editor opens, displaying the values for the selected variable. The Variable Editor appears as shown in Figure 2.1–2.

To open a variable, you can also right-click it and use the **Context** menu. Repeat the steps to open additional variables into the Variable Editor. In the Variable Editor, access each variable via its tab at the bottom of the window, or use the **Window** menu. You can also open the Variable Editor directly from the Command window by typing `open('var')`, where `var` is the name of the variable to be edited. Once an array is displayed in the Variable Editor, you can change a value in the array by clicking on its location, typing in the new value, and pressing **Enter**.

Right-clicking on a variable brings up the **Context** menu, which can be used to edit, save, or clear the selected variable, or to plot the rows of the variable versus its columns (this type of plot is discussed in Chapter 5).

You can also clear a variable from theWorkspace Browser by first highlighting it in the Browser, then clicking on **Delete** in the **Edit** menu.

2.2 Multidimensional Numeric Arrays

MATLAB supports multidimensional arrays. For more information, type `help datatypes`.

A three-dimensional array has the dimension $m \times n \times q$. A four-dimensional array has the dimension $m \times n \times q \times r$, and so forth. The first two dimensions are the row and column, as with a matrix. The higher dimensions are called *pages*. You can think of a three-dimensional array as layers of matrices. The first layer is page 1; the second layer is page 2, and so on. If `A` is a $3 \times 3 \times 2$ array, you can access the

element in row 3, column 2 of page 2 by typing `A(3,2,2)`. To access all of page 1, type `A(:,:,1)`. To access all of page 2, type `A(:,:,2)`. The `ndims` command returns the number of dimensions. For example, for the array `A` just described, `ndims(A)` returns the value 3.

You can create a multidimensional array by first creating a two-dimensional array and then extending it. For example, suppose you want to create a three-dimensional array whose first two pages are

$$\begin{bmatrix} 4 & 6 & 1 \\ 5 & 8 & 0 \\ 3 & 9 & 2 \end{bmatrix} \quad \begin{bmatrix} 6 & 2 & 9 \\ 0 & 3 & 1 \\ 4 & 7 & 5 \end{bmatrix}$$

To do so, first create page 1 as a 3×3 matrix and then add page 2, as follows:

```
>>A = [4,6,1;5,8,0;3,9,2];
>>A(:,:,2) = [6,2,9;0,3,1;4,7,5];
```

Another way to produce such an array is with the `cat` command. Typing `cat(n,A,B,C,...)` creates a new array by concatenating the arrays `A, B, C`, and so on along the dimension `n`. Note that `cat(1,A,B)` is the same as `[A;B]` and that `cat(2,A,B)` is the same as `[A,B]`. For example, suppose we have the 2×2 arrays **A** and **B:**

$$\mathbf{A} = \begin{bmatrix} 8 & 2 \\ 9 & 5 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 4 & 6 \\ 7 & 3 \end{bmatrix}$$

Then `C = cat(3,A,B)` produces a three-dimensional array composed of two layers; the first layer is the matrix `A`, and the second layer is the matrix `B`. The element `C(m,n,p)` is located in row *m*, column *n*, and layer *p*. Thus the element `C(2,1,1)` is 9, and the element `C(2,2,2)` is 3.

Multidimensional arrays are useful for problems that involve several parameters. For example, if we have data on the temperature distribution in a rectangular object, we could represent the temperatures as an array `T` with three dimensions.

2.3 Element-by-Element Operations

To increase the magnitude of a vector, multiply it by a scalar. For example, to double the magnitude of the vector `r = [3,5,2]`, multiply each component by 2 to obtain `[6,10,4]`. In MATLAB you type `v = 2*r`.

Multiplying a matrix **A** by a scalar *w* produces a matrix whose elements are the elements of **A** multiplied by *w*. For example:

$$3\begin{bmatrix} 2 & 9 \\ 5 & -7 \end{bmatrix} = \begin{bmatrix} 6 & 27 \\ 15 & -21 \end{bmatrix}$$

This multiplication is performed in MATLAB as follows:

```
>>A = [2,9;5,-7];
>>3*A
```

Thus multiplication of an array by a scalar is easily defined and easily carried out. However, multiplication of two *arrays* is not so straightforward. In fact, MATLAB uses two definitions of multiplication: (1) array multiplication and (2) matrix multiplication. Division and exponentiation must also be carefully defined when you are dealing with operations between two arrays. MATLAB has two forms of arithmetic operations on arrays. In this section we introduce one form, called *array operations,* which are also called *element-by-element* operations. In the next section we introduce *matrix* operations. Each form has its own applications, which we illustrate by examples.

ARRAY OPERATIONS

Array Addition and Subtraction

Array addition can be done by adding the corresponding components. To add the arrays `r = [3,5,2]` and `v = [2,-3,1]` to create `w` in MATLAB, you type `w = r + v`. The result is `w = [5,2,3]`.

When two arrays have identical size, their sum or difference has the same size and is obtained by adding or subtracting their corresponding elements. Thus $\mathbf{C} = \mathbf{A} + \mathbf{B}$ implies that $c_{ij} = a_{ij} + b_{ij}$ if the arrays are matrices. The array $\mathbf{C}$ has the same size as $\mathbf{A}$ and $\mathbf{B}$. For example,

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \end{bmatrix} + \begin{bmatrix} 9 & 8 \\ -12 & 14 \end{bmatrix} = \begin{bmatrix} 15 & 6 \\ -2 & 17 \end{bmatrix} \tag{2.3–1}$$

Array subtraction is performed in a similar way.

The addition shown in Equation (2.3–1) is performed in MATLAB as follows:

```
>>A = [6,-2;10,3];
>>B = [9,8;-12,14]
>>A+B
ans =
     15   6
     -2   17
```

Array addition and subtraction are associative and commutative. For addition these properties mean that

$$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C}) \tag{2.3–2}$$

$$\mathbf{A} + \mathbf{B} + \mathbf{C} = \mathbf{B} + \mathbf{C} + \mathbf{A} = \mathbf{A} + \mathbf{C} + \mathbf{B} \tag{2.3–3}$$

Array addition and subtraction require that both arrays be the same size. The only exception to this rule in MATLAB occurs when we add or subtract a *scalar* to or from an array. In this case the scalar is added or subtracted from each element in the array. Table 2.3–1 gives examples.

Element-by-Element Multiplication

MATLAB defines element-by-element multiplication only for arrays that are the same size. The definition of the product `x.*y`, where `x` and `y` each have *n* elements, is

```
x.*y = [x(1)y(1),  x(2)y(2) . . . , x(n)y(n)]
```

Table 2.3–1 Element-by-element operations

Symbol	Operation	Form	Example
+	Scalar-array addition	`A + b`	`[6,3]+2=[8,5]`
−	Scalar-array subtraction	`A - b`	`[8,3]-5=[3,-2]`
+	Array addition	`A + B`	`[6,5]+[4,8]=[10,13]`
−	Array subtraction	`A - B`	`[6,5]-[4,8]=[2,-3]`
.*	Array multiplication	`A.*B`	`[3,5].*[4,8]=[12,40]`
./	Array right division	`A./B`	`[2,5]./[4,8]=[2/4,5/8]`
.\	Array left division	`A.\B`	`[2,5].\[4,8]=[2\4,5\8]`

.^	Array exponentiation	`A.^B`	`[3,5].^2=[3^2,5^2]`
			`2.^[3,5]=[2^3,2^5]`
			`[3,5].^[2,4]=[3^2,5^4]`

if `x` and `y` are row vectors. For example, if

$$\mathbf{x} = [2, 4, -5] \qquad \mathbf{y} = [-7, 3, -8] \tag{2.3–4}$$

then `z = x.*y` gives

$$\mathbf{z} = [2(-7), 4(3), -5(-8)] = [-14, 12, 40]$$

This type of multiplication is sometimes called *array* multiplication.

If `u` and `v` are column vectors, the result of `u.*v` is a column vector.

Note that `x'` is a column vector with size 3×1 and thus does not have the same size as `y`, whose size is 1×3. Thus for the vectors `x` and `y` the operations `x'.*y` and `y.*x'` are not defined in MATLAB and will generate an error message. With element-by-element multiplication, it is important to remember that the dot (`.`) and the asterisk (`*`) form *one* symbol (`.*`). It might have been better to have defined a single symbol for this operation, but the developers of MATLAB were limited by the selection of symbols on the keyboard.

The generalization of array multiplication to arrays with more than one row or column is straightforward. Both arrays must be the same size. The array operations are performed between the elements in corresponding locations in the arrays. For example, the array multiplication operation `A.*B` results in a matrix `C` that has the same size as `A` and `B` and has the elements $c_{ij} = a_{ij}\, b_{ij}$. For example, if

$$\mathbf{A} = \begin{bmatrix} 11 & 5 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -7 & 8 \\ 6 & 2 \end{bmatrix}$$

then `C = A.*B` gives this result:

$$\mathbf{C} = \begin{bmatrix} 11(-7) & 5(8) \\ -9(6) & 4(2) \end{bmatrix} = \begin{bmatrix} -77 & 40 \\ -54 & 8 \end{bmatrix}$$

Vectors and Displacement

EXAMPLE 2.3–1

Suppose two divers start at the surface and establish the following coordinate system: x is to the west, y is to the north, and z is down. Diver 1 swims 55 ft west, 36 ft north, and then dives 25 ft. Diver 2 dives 15 ft, then swims east 20 ft and then north 59 ft. (a) Find the distance between diver 1 and the starting point. (b) How far in each direction must diver 1 swim to reach diver 2? How far in a straight line must diver 1 swim to reach diver 2?

■ Solution

(a) Using the xyz coordinates selected, the position of diver 1 is $\mathbf{r} = 55\mathbf{i} + 36\mathbf{j} + 25\mathbf{k}$, and the position of diver 2 is $\mathbf{r} = -20\mathbf{i} + 59\mathbf{j} + 15\mathbf{k}$. (Note that diver 2 swam east, which is in the negative x direction.) The distance from the origin of a point xyz is given by $\sqrt{x^2 + y^2 + z^2}$, that is, by the magnitude of the vector pointing from the origin to the point xyz. This distance is computed in the following session.

```
>>r = [55,36,25];w = [-20,59,15];
>>dist1 = sqrt(sum(r.*r))
dist1 =
   70.3278
```

The distance is approximately 70 ft. The distance could also have been computed from `norm(r)`.

(b) The location of diver 2 relative to diver 1 is given by the vector **v** pointing from diver 1 to diver 2. We can find this vector using vector subtraction: $\mathbf{v} = \mathbf{w} - \mathbf{r}$. Continue the above MATLAB session as follows:

```
>>v = w-r
v =
   -75     23    -10
>>dist2 = sqrt(sum(v.*v))
dist2 =
   79.0822
```

Thus to reach diver 2 by swimming along the coordinate directions, diver 1 must swim 75 ft east, 23 ft north, and 10 ft up. The straight-line distance between them is approximately 79 ft.

Vectorized Functions

The built-in MATLAB functions such as `sqrt(x)` and `exp(x)` automatically operate on array arguments to produce an array result the same size as the array argument `x`. Thus these functions are said to be *vectorized* functions.

Thus, when multiplying or dividing these functions, or when raising them to a power, you must use element-by-element operations if the arguments are arrays. For example, to compute $z = (e^y \sin x)\cos^2 x$, you must type `z = exp(y).* sin(x).*(cos(x)).^2`. Obviously, you will get an error message if the size of `x` is not the same as the size of `y`. The result `z` will have the same size as `x` and `y`.

EXAMPLE 2.3–2 Aortic Pressure Model

The following equation is a specific case of one model used to describe the blood pressure in the aorta during systole (the period following the closure of the heart's aortic valve). The variable t represents time in seconds, and the dimensionless variable y represents the pressure difference across the aortic valve, normalized by a constant reference pressure.

$$y(t) = e^{-8t} \sin\left(9.7t + \frac{\pi}{2}\right)$$

Plot this function for $t \geq 0$.

■ Solution

Note that if `t` is a vector, the MATLAB functions `exp(-8*t)` and `sin(9.7*t+pi/2)` will also be vectors the same size as `t`. Thus we must use element-by-element multiplication to compute $y(t)$.

We must decide on the proper spacing to use for the vector `t` and its upper limit. The sine function $\sin(9.7t + \pi/2)$ oscillates with a frequency of 9.7 rad/sec, which is $9.7/(2\pi) = 1.5$ Hz. Thus its period is $1/1.5 = 2/3$ sec. The spacing of `t` should be a small fraction of the period in order to generate enough points to plot the curve. Thus we select a spacing of 0.003 to give approximately 200 points per period.

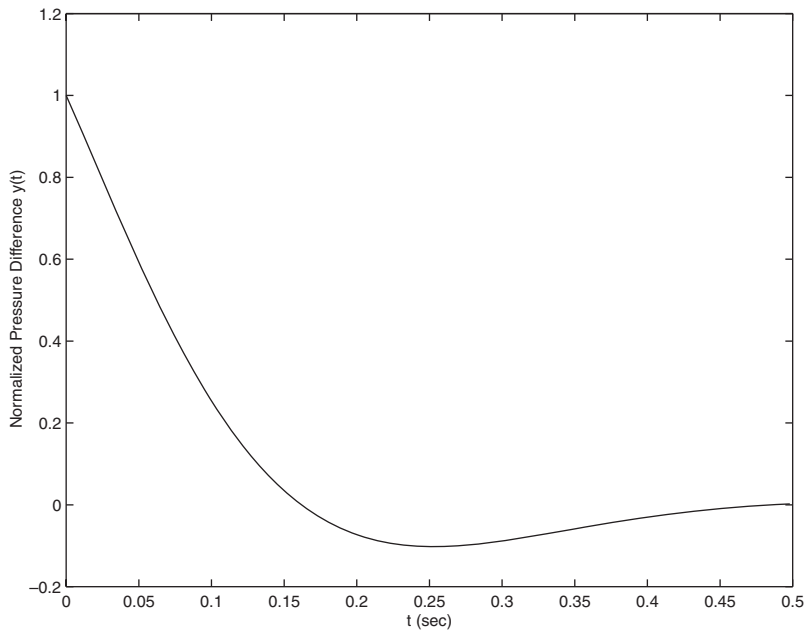

Figure 2.3–1 Aortic pressure response for Example 2.3–2.

The amplitude of the sine wave decays with time because the sine is multiplied by the decaying exponential e^{-8t}. The exponential's initial value is $e^0 = 1$, and it will be 2 percent of its initial value at $t = 0.5$ (because $e^{-8\,(0.5)} = 0.02$). Thus we select the upper limit of `t` to be 0.5. The session is

```
>>t = 0:0.003:0.5;
>>y = exp(-8*t).*sin(9.7*t+pi/2);
>>plot(t,y),xlabel('t (sec)'), . . .
     ylabel('Normalized Pressure Difference y(t)')
```

The plot is shown in Figure 2.3–1. Note that we do not see much of an oscillation despite the presence of a sine wave. This is so because the period of the sine wave is greater than the time it takes for the exponential e^{-8t} to become essentially zero.

Element-by-Element Division

The definition of element-by-element division, also called array division, is similar to the definition of array multiplication except, of course, that the elements of one array are divided by the elements of the other array. Both arrays must be the same size. The symbol for array right division is `./`. For example, if

$$\mathbf{x} = [8, 12, 15] \qquad \mathbf{y} = [-2, 6, 5]$$

then `z = x./y` gives

$$\mathbf{z} = [8/(-2), 12/6, 15/5] = [-4, 2, 3]$$

Also, if

$$\mathbf{A} = \begin{bmatrix} 24 & 20 \\ -9 & 4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -4 & 5 \\ 3 & 2 \end{bmatrix}$$

then `C = A./B` gives

$$\mathbf{C} = \begin{bmatrix} 24/(-4) & 20/5 \\ -9/3 & 4/2 \end{bmatrix} = \begin{bmatrix} -6 & 4 \\ -3 & 2 \end{bmatrix}$$

The array left division operator (`.\`) is defined to perform element-by-element division using left division. Refer to Table 2.3–1 for examples. Note that `A.\B` is not equivalent to `A./B`.

EXAMPLE 2.3–3 Transportation Route Analysis

The following table gives data for the distance traveled along five truck routes and the corresponding time required to traverse each route. Use the data to compute the average speed required to drive each route. Find the route that has the highest average speed.

	1	2	3	4	5
Distance (mi)	560	440	490	530	370
Time (hr)	10.3	8.2	9.1	10.1	7.5

■ Solution

For example, the average speed on the first route is 560/10.3 = 54.4 mi/hr. First we define the row vectors `d` and `t` from the distance and time data. Then, to find the average speed on each route using MATLAB, we use array division. The session is

```
>>d = [560, 440, 490, 530, 370]
>>t = [10.3, 8.2, 9.1, 10.1, 7.5]
>>speed = d./t
speed =
   54.3689    53.6585    53.8462    52.4752    49.3333
```

The results are in miles per hour. Note that MATLAB displays more significant figures than is justified by the three-significant-figure accuracy of the given data, so we should round the results to three significant figures before using them.

To find the highest average speed and the corresponding route, continue the session as follows:

```
>>[highest_speed, route] = max(speed)
highest_speed =
   54.3689
route =
   1
```

The first route has the highest speed.

If we did not need the speeds for every route, we could have solved this problem by combining two lines as follows: `[highest_speed, route] = max(d./t)`.

Element-by-Element Exponentiation

MATLAB enables us not only to raise arrays to powers but also to raise scalars and arrays to *array* powers. To perform exponentiation on an element-by-element basis, we must use the `.^` symbol. For example, if `x = [3, 5, 8]`, then typing `x.^3` produces the array $[3^3, 5^3, 8^3] = [27, 125, 512]$. If `x = 0:2:6`, then typing `x.^2` returns the array $[0^2, 2^2, 4^2, 6^2] = [0, 4, 16, 36]$. If

$$\mathbf{A} = \begin{bmatrix} 4 & -5 \\ 2 & 3 \end{bmatrix}$$

then `B = A.^3` gives this result:

$$\mathbf{B} = \begin{bmatrix} 4^3 & (-5)^3 \\ 2^3 & 3^3 \end{bmatrix} = \begin{bmatrix} 64 & -125 \\ 8 & 27 \end{bmatrix}$$

We can raise a scalar to an array power. For example, if `p = [2, 4, 5]`, then typing `3.^p` produces the array $[3^2, 3^4, 3^5] = [9, 81, 243]$. This example illustrates a common situation in which it helps to remember that `.^` is a *single* symbol; the dot in `3.^p` is not a decimal point associated with the number 3. The following operations, with the value of `p` given here, are equivalent and give the correct answer:

```
3.^p
3.0.^p
3..^p
(3).^p
3.^[2,4,5]
```

With array exponentiation, the power may be an array if the base is a scalar or if the power's dimensions are the same as the base dimensions. For example, if, `x = [1,2,3]` and `y = [2,3,4]`, then `y.^x` gives the answer 2964. If `A = [1,2; 3,4]`, then `2.^A` gives the array `[2,4;8,16]`.

Test Your Understanding

T2.3–1 Given the matrices

$$\mathbf{A} = \begin{bmatrix} 21 & 27 \\ -18 & 8 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -7 & -3 \\ 9 & 4 \end{bmatrix}$$

find (a) their array product, (b) their array right division (**A** divided by **B**), and (c) **B** raised to the third power element by element.
(Answers: (a) `[-147, -81; -162, 32]`, (b) `[-3, -9; -2, 2]`, and (c) `[-343, -27; 729, 64]`.)

Current and Power Dissipation in Resistors

EXAMPLE 2.3–4

The current i passing through an electrical resistor having a voltage v across it is given by Ohm's law, $i = v/R$, where R is the resistance. The power dissipated in the resistor is given by v^2/R. The following table gives data for the resistance and voltage for five resistors. Use the data to compute (a) the current in each resistor and (b) the power dissipated in each resistor.

	1	**2**	**3**	**4**	**5**
R (Ω)	10^4	2×10^4	3.5×10^4	10^5	2×10^5
v (V)	120	80	110	200	350

■ Solution

(a) First we define two row vectors, one containing the resistance values and one containing the voltage values. To find the current $i = v/R$ using MATLAB, we use array division. The session is

```
>>R = [10000, 20000, 35000, 100000, 200000];
>>v = [120, 80, 110, 200, 350];
>>current = v./R
current =
   0.0120  0.0040  0.0031  0.0020  0.0018
```

The results are in amperes and should be rounded to three significant figures because the voltage data contains only three significant figures.

(b) To find the power $P = v^2/R$, use array exponentiation and array division. The session continues as follows:

```
>>power = v.^2./R
power =
   1.4400  0.3200  0.3457  0.4000  0.6125
```

These numbers are the power dissipation in each resistor in watts. Note that the statement `v.^2./R` is equivalent to `(v.^2)./R`. Although the rules of precedence are unambiguous here, we can always put parentheses around quantities if we are unsure how MATLAB will interpret our commands.

EXAMPLE 2.3–5 A Batch Distillation Process

Consider a system for heating a liquid benzene/toluene solution to distill a pure benzene vapor. A particular batch distillation unit is charged initially with 100 mol of a 60 percent mol benzene/40 percent mol toluene mixture. Let L (mol) be the amount of liquid remaining in the still, and let x (mol B/mol) be the benzene mole fraction in the remaining liquid. Conservation of mass for benzene and toluene can be applied to derive the following relation [Felder, 1986].

$$L = 100\left(\frac{x}{0.6}\right)^{0.625}\left(\frac{1-x}{0.4}\right)^{-1.625}$$

Determine what mole fraction of benzene remains when $L = 70$. Note that it is difficult to solve this equation directly for x. Use a plot of x versus L to solve the problem.

■ Solution

This equation involves both array multiplication and array exponentiation. Note that MATLAB enables us to use decimal exponents to evaluate L. It is clear that L must be in the range $0 \leq L \leq 100$; however, we do not know the range of x, except that

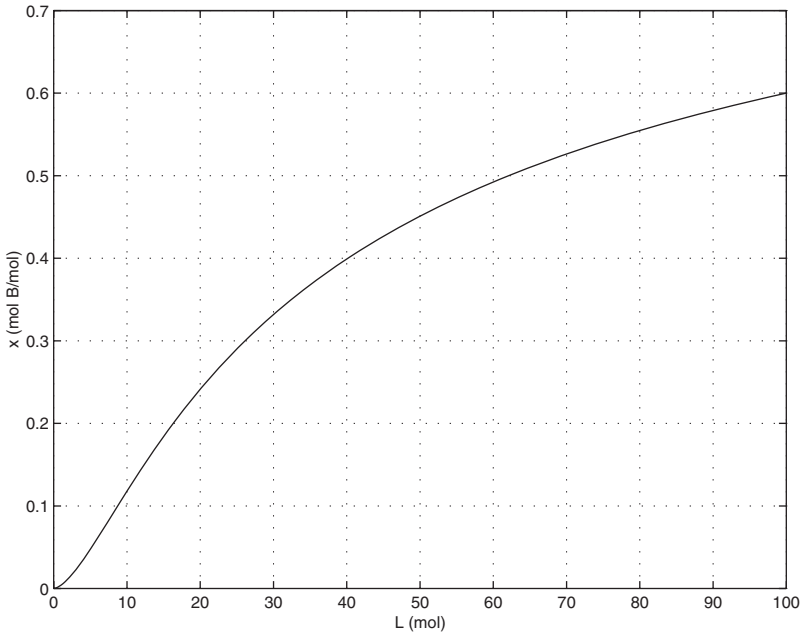

Figure 2.3–2 Plot for Example 2.3–5.

$x \geq 0$. Therefore, we must make a few guesses for the range of x, using a session like the following. We find that $L > 100$ if $x > 0.6$, so we choose `x = 0:0.001:0.6`. We use the `ginput` function to find the value of x corresponding to $L = 70$.

```
>>x = 0:0.001:0.6;
>>L = 100*(x/0.6).^(0.625).*((1-x)/0.4).^(-1.625);
>>plot(L,x),grid,xlabel('L(mol)'),ylabel('x (mol B/mol)'), ...
  [L,x] = ginput(1)
```

The plot is shown in Figure 2.3–2. The answer is $x = 0.52$ if $L = 70$. The plot shows that the remaining liquid becomes leaner in benzene as the liquid amount becomes smaller. Just before the still is empty ($L = 0$), the liquid is pure toluene.

2.4 Matrix Operations

Matrix addition and subtraction are identical to element-by-element addition and subtraction. The corresponding matrix elements are summed or subtracted. However, matrix multiplication and division are not the same as element-by-element multiplication and division.

Multiplication of Vectors

Recall that vectors are simply matrices with one row or one column. Thus matrix multiplication and division procedures apply to vectors as well, and we will introduce matrix multiplication by considering the vector case first.

The *vector dot product* $\mathbf{u} \cdot \mathbf{w}$ of the vectors $\mathbf{u}$ and $\mathbf{w}$ is a scalar and can be thought of as the perpendicular projection of $\mathbf{u}$ onto $\mathbf{w}$. It can be computed from $|\mathbf{u}||\mathbf{w}| \cos \theta$, where θ is the angle between the two vectors and $|\mathbf{u}|$, $|\mathbf{w}|$ are the magnitudes of the vectors. Thus if the vectors are parallel and in the same direction, $\theta = 0$ and $\mathbf{u} \cdot \mathbf{w} = |\mathbf{u}||\mathbf{w}|$. If the vectors are perpendicular, $\theta = 90°$ and thus $\mathbf{u} \cdot \mathbf{w} = 0$. Because the unit vectors $\mathbf{i}$, $\mathbf{j}$, and $\mathbf{k}$ have unit length,

$$\mathbf{i} \cdot \mathbf{i} = \mathbf{j} \cdot \mathbf{j} = \mathbf{k} \cdot \mathbf{k} = 1 \tag{2.4–1}$$

Because the unit vectors are perpendicular,

$$\mathbf{i} \cdot \mathbf{j} = \mathbf{i} \cdot \mathbf{k} = \mathbf{j} \cdot \mathbf{k} = 0 \tag{2.4–2}$$

Thus the vector dot product can be expressed in terms of unit vectors as

$$\mathbf{u} \cdot \mathbf{w} = (u_1\mathbf{i} + u_2\mathbf{j} + u_3\mathbf{k}) \cdot (w_1\mathbf{i} + w_2\mathbf{j} + w_3\mathbf{k})$$

Carrying out the multiplication algebraically and using the properties given by (2.4–1) and (2.4–2), we obtain

$$\mathbf{u} \cdot \mathbf{w} = u_1w_1 + u_2w_2 + u_3w_3$$

The *matrix* product of a *row* vector $\mathbf{u}$ with a *column* vector $\mathbf{w}$ is defined in the same way as the vector dot product; the result is a scalar that is the sum of the products of the corresponding vector elements; that is,

$$\begin{bmatrix} u_1 & u_2 & u_3 \end{bmatrix} \begin{bmatrix} w_1 \\ w_2 \\ w_3 \end{bmatrix} = u_1w_1 + u_2w_2 + u_3w_3$$

if each vector has three elements. Thus the result of multiplying a 1×3 vector by a 3×1 vector is a 1×1 array, that is, a scalar. This definition applies to vectors having any number of elements, as long as both vectors have the same number of elements.

Thus the result of multiplying a $1 \times n$ vector by an $n \times 1$ vector is a 1×1 array, that is, a scalar.

EXAMPLE 2.4–1 Miles Traveled

Table 2.4–1 gives the speed of an aircraft on each leg of a certain trip and the time spent on each leg. Compute the miles traveled on each leg and the total miles traveled.

■ Solution

We can define a row vector `s` containing the speeds and a row vector `t` containing the times for each leg. Thus `s = [200, 250, 400, 300]` and `t = [2, 5, 3, 4]`.

Table 2.4–1 Aircraft speeds and times per leg

	Leg			
	1	2	3	4
Speed (mi/hr)	200	250	400	300
Time (hr)	2	5	3	4

To find the miles traveled on each leg, we multiply the speed by the time. To do so, we use the MATLAB symbol `.*`, which *specifies* the multiplication `s.*t` to produce the row vector whose elements are the products of the corresponding elements in `s` and `t`:

$$\texttt{s.*t} = [200(2), 250(5), 400(3), 300(4)] = [400, 1250, 1200, 1200]$$

This vector contains the miles traveled by the aircraft on each leg of the trip.

To find the total miles traveled, we use the matrix product, denoted by `s*t'`. In this definition the product is the *sum* of the individual element products; that is,

$$\texttt{s*t'} = [200(2) + 250(5) + 400(3) + 300(4)] = 4050$$

These two examples illustrate the difference between *array* multiplication `s.*t` and *matrix* multiplication `s*t'`.

Vector-Matrix Multiplication

Not all matrix products are scalars. To generalize the preceding multiplication to a column vector multiplied by a matrix, think of the matrix as being composed of row vectors. The scalar result of each row-column multiplication forms an element in the result, which is a column vector. For example:

$$\begin{bmatrix} 2 & 7 \\ 6 & -5 \end{bmatrix} \begin{bmatrix} 3 \\ 9 \end{bmatrix} = \begin{bmatrix} 2(3) + 7(9) \\ 6(3) - 5(9) \end{bmatrix} = \begin{bmatrix} 69 \\ -27 \end{bmatrix} \tag{2.4–3}$$

Thus the result of multiplying a 2×2 matrix by a 2×1 vector is a 2×1 array, that is, a column vector. Note that the definition of multiplication requires that the number of columns in the matrix be equal to the number of rows in the vector. In general, the product $\mathbf{Ax}$, where $\mathbf{A}$ has p columns, is defined only if $\mathbf{x}$ has p rows. If $\mathbf{A}$ has m rows and $\mathbf{x}$ is a column vector, the result of $\mathbf{Ax}$ is a column vector with m rows.

Matrix-Matrix Multiplication

We can expand this definition of multiplication to include the product of two matrices $\mathbf{AB}$. The number of columns in $\mathbf{A}$ must equal the number of rows in $\mathbf{B}$. The row-column multiplications form column vectors, and these column vectors form the

matrix result. The product **AB** has the same number of rows as **A** and the same number of columns as **B**. For example,

$$\begin{bmatrix} 6 & -2 \\ 10 & 3 \\ 4 & 7 \end{bmatrix} \begin{bmatrix} 9 & 8 \\ -5 & 12 \end{bmatrix} = \begin{bmatrix} (6)(9) + (-2)(-5) & (6)(8) + (-2)(12) \\ (10)(9) + (3)(-5) & (10)(8) + (3)(12) \\ (4)(9) + (7)(-5) & (4)(8) + (7)(12) \end{bmatrix}$$

$$= \begin{bmatrix} 64 & 24 \\ 75 & 116 \\ 1 & 116 \end{bmatrix} \tag{2.4–4}$$

Use the operator * to perform matrix multiplication in MATLAB. The following MATLAB session shows how to perform the matrix multiplication shown in (2.4–4).

```
>>A = [6,-2;10,3;4,7];
>>B = [9,8;-5,12];
>>A*B
```

Element-by-element multiplication is defined for the following product:

$$[3 \quad 1 \quad 7][4 \quad 6 \quad 5] = [12 \quad 6 \quad 35]$$

However, this product is *not* defined for *matrix* multiplication, because the first matrix has three columns, but the second matrix does not have three rows. Thus if we were to type `[3, 1, 7]*[4, 6, 5]` in MATLAB, we would receive an error message.

The following product is defined in matrix multiplication and gives the result shown:

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} [y_1 \quad y_2 \quad y_3] = \begin{bmatrix} x_1y_1 & x_1y_2 & x_1y_3 \\ x_2y_1 & x_2y_2 & x_2y_3 \\ x_3y_1 & x_3y_2 & x_3y_3 \end{bmatrix}$$

The following product is also defined:

$$[10 \quad 6] \begin{bmatrix} 7 & 4 \\ 5 & 2 \end{bmatrix} = [10(7) + 6(5) \; 10(4) + 6(2)] = [100 \quad 52]$$

Evaluating Multivariable Functions

To evaluate a function of two variables, say, $z = f(x, y)$, for the values $x = x_1, x_2, \ldots, x_m$ and $y = y_1, y_2, \ldots, y_n$, define the $m \times n$ matrices:

$$\mathbf{x} = \begin{bmatrix} x_1 & x_1 & \cdots & x_1 \\ x_2 & x_2 & \cdots & x_2 \\ \vdots & \vdots & \vdots & \vdots \\ x_m & x_m & \cdots & x_m \end{bmatrix} \qquad \mathbf{y} = \begin{bmatrix} y_1 & y_2 & \cdots & y_n \\ y_1 & y_2 & \cdots & y_n \\ \vdots & \vdots & \vdots & \vdots \\ y_1 & y_2 & \cdots & y_n \end{bmatrix}$$

When the function $z = f(x, y)$ is evaluated in MATLAB using array operations, the resulting $m \times n$ matrix **z** has the elements $z_{ij} = f(x_i, y_j)$. We can extend this technique to functions of more than two variables by using multidimensional arrays.

Height versus Velocity

EXAMPLE 2.4–2

The maximum height h achieved by an object thrown with a speed v at an angle θ to the horizontal, neglecting drag, is

$$h = \frac{v^2 \sin^2\theta}{2g}$$

Create a table showing the maximum height for the following values of v and θ:

$$v = 10, 12, 14, 16, 18, 20 \text{ m/s} \qquad \theta = 50°, 60°, 70°, 80°$$

The rows in the table should correspond to the speed values, and the columns should correspond to the angles.

■ Solution

The program is shown below.

```
g = 9.8; v = 10:2:20;
theta = 50:10:80;
h = (v'.^2)*(sind(theta).^2)/(2*g);
table = [0, theta; v', h]
```

The arrays `v` and `theta` contain the given velocities and angles. The array `v` is 1×6 and the array `theta` is 1×4. Thus the term `v'.^2` is a 6×1 array, and the term `sind(theta).^2` is a 1×4 array. The product of these two arrays, `h`, is a matrix product and is a $(6 \times 1)(1 \times 4) = (6 \times 4)$ matrix.

The array `[0, theta]` is 1×5 and the array `[v', h]` is 6×5, so the matrix `table` is 7×5. The following table shows the matrix `table` rounded to one decimal place. From this table we can see that the maximum height is 8.8 m if $v = 14$ m/s and $\theta = 70°$.

0	50	60	70	80
10	3.0	3.8	4.5	4.9
12	4.3	5.5	6.5	7.1
14	5.9	7.5	8.8	9.7
16	7.7	9.8	11.5	12.7
18	9.7	12.4	14.6	16.0
20	12.0	15.3	18.0	19.8

Test Your Understanding

T2.4–1 Use MATLAB to compute the dot product of the following vectors:

$$\mathbf{u} = 6\mathbf{i} - 8\mathbf{j} + 3\mathbf{k}$$
$$\mathbf{w} = 5\mathbf{i} + 3\mathbf{j} - 4\mathbf{k}$$

Check your answer by hand. (Answer: -6.)

T2.4–2 Use MATLAB to show that

$$\begin{bmatrix} 7 & 4 \\ -3 & 2 \\ 5 & 9 \end{bmatrix} \begin{bmatrix} 1 & 8 \\ 7 & 6 \end{bmatrix} = \begin{bmatrix} 35 & 80 \\ 11 & -12 \\ 68 & 94 \end{bmatrix}$$

EXAMPLE 2.4–3 Manufacturing Cost Analysis

Table 2.4–2 shows the hourly cost of four types of manufacturing processes. It also shows the number of hours required of each process to produce three different products. Use matrices and MATLAB to solve the following. (a) Determine the cost of each process to produce 1 unit of product 1. (b) Determine the cost to make 1 unit of each product. (c) Suppose we produce 10 units of product 1, 5 units of product 2, and 7 units of product 3. Compute the total cost.

■ Solution

(a) The basic principle we can use here is that cost equals the hourly cost times the number of hours required. For example, the cost of using the lathe for product 1 is ($10/hr)(6 hr) = $60, and so forth for the other three processes. If we define the row vector of hourly costs to be `hourly_costs` and define the row vector of hours required for product 1 to be `hours_1`, then we can compute the costs of each process for product 1 using *element-by-element* multiplication. In MATLAB the session is

```
>>hourly_cost = [10, 12, 14, 9];
>>hours_1 = [6, 2, 3, 4];
>>process_cost_1 = hourly_cost.*hours_1
process_cost_1 =
   60   24   42   36
```

These are the costs of each of the four processes to produce 1 unit of product 1.

(b) To compute the total cost of 1 unit of product 1, we can use the vectors `hourly_costs` and `hours_1` but apply *matrix* multiplication instead of element-by-element multiplication, because matrix multiplication sums the individual products. The matrix multiplication gives

Table 2.4–2 Cost and time data for manufacturing processes

		Hours required to produce one unit		
Process	**Hourly cost ($)**	**Product 1**	**Product 2**	**Product 3**
Lathe	10	6	5	4
Grinding	12	2	3	1
Milling	14	3	2	5
Welding	9	4	0	3

$$[10 \;\; 12 \;\; 14 \;\; 9] \begin{bmatrix} 6 \\ 2 \\ 3 \\ 4 \end{bmatrix} = 10(6) + 12(2) + 14(3) + 9(4) = 162$$

We can perform similar multiplication for products 2 and 3, using the data in the table. For product 2:

$$[10 \;\; 12 \;\; 14 \;\; 9] \begin{bmatrix} 5 \\ 3 \\ 2 \\ 0 \end{bmatrix} = 10(5) + 12(2) + 14(3) + 9(0) = 114$$

For product 3:

$$[10 \;\; 12 \;\; 14 \;\; 9] \begin{bmatrix} 4 \\ 1 \\ 5 \\ 3 \end{bmatrix} = 10(4) + 12(1) + 14(5) + 9(3) = 149$$

These three operations could have been accomplished in one operation by defining a matrix whose columns are formed by the data in the last three columns of the table:

$$[10 \;\; 12 \;\; 14 \;\; 9] \begin{bmatrix} 6 & 5 & 4 \\ 2 & 3 & 1 \\ 3 & 2 & 5 \\ 4 & 0 & 3 \end{bmatrix} = \begin{bmatrix} 60 + 24 + 42 + 36 \\ 50 + 36 + 28 + 0 \\ 40 + 12 + 70 + 27 \end{bmatrix} = [162 \;\; 114 \;\; 149]$$

In MATLAB the session continues as follows. Remember that we must use the transpose operation to convert the row vectors into column vectors.

```
>>hours_2 = [5, 3, 2, 0];
>>hours_3 = [4, 1, 5, 3];
>>unit_cost = hourly_cost*[hours_1', hours_2', hours_3']
unit_cost =
   162    114    149
```

Thus the costs to produce 1 unit each of products 1, 2, and 3 are \$162, \$114, and \$149, respectively.

(c) To find the total cost to produce 10, 5, and 7 units, respectively, we can use matrix multiplication:

$$[10 \;\; 5 \;\; 7] \begin{bmatrix} 162 \\ 114 \\ 149 \end{bmatrix} = 1620 + 570 + 1043 = 3233$$

In MATLAB the session continues as follows. Note the use of the transpose operator on the vector `unit_cost`.

```
>>units = [10, 5, 7];
>>total_cost = units*unit_cost'
total_cost =
   3233
```

The total cost is $3233.

The General Matrix Multiplication Case

We can state the general result for matrix multiplication as follows: Suppose **A** has dimension $m \times p$ and **B** has dimension $p \times q$. If **C** is the product **AB**, then **C** has dimension $m \times q$ and its elements are given by

$$c_{ij} = \sum_{k=1}^{p} a_{ik} b_{kj} \tag{2.4–5}$$

for all $i = 1, 2, \ldots, m$ and $j = 1, 2, \ldots, q$. For the product to be defined, the matrices **A** and **B** must be *conformable;* that is, the number of *rows* in **B** must equal the number of *columns* in **A**. The product has the same number of rows as **A** and the same number of columns as **B**.

Matrix multiplication does not have the commutative property; that is, in general, $\mathbf{AB} \neq \mathbf{BA}$. Reversing the order of matrix multiplication is a common and easily made mistake.

The associative and distributive properties hold for matrix multiplication. The associative property states that

$$\mathbf{A}(\mathbf{B} + \mathbf{C}) = \mathbf{AB} + \mathbf{AC} \tag{2.4–6}$$

The distributive property states that

$$(\mathbf{AB})\mathbf{C} = \mathbf{A}(\mathbf{BC}) \tag{2.4–7}$$

Applications to Cost Analysis

Project cost data stored in tables must often be analyzed in several ways. The elements in MATLAB matrices are similar to the cells in a spreadsheet, and MATLAB can perform many spreadsheet-type calculations for analyzing such tables.

EXAMPLE 2.4–4 Product Cost Analysis

Table 2.4–3 shows the costs associated with a certain product, and Table 2.4–4 shows the production volume for the four quarters of the business year. Use MATLAB to find the quarterly costs for materials, labor, and transportation; the total material, labor, and transportation costs for the year; and the total quarterly costs.

Table 2.4–3 Product costs

Unit costs ($\$ \times 10^3$)			
Product	**Materials**	**Labor**	**Transportation**
1	6	2	1
2	2	5	4
3	4	3	2
4	9	7	3

Table 2.4–4 Quarterly production volume

Product	Quarter 1	Quarter 2	Quarter 3	Quarter 4
1	10	12	13	15
2	8	7	6	4
3	12	10	13	9
4	6	4	11	5

■ Solution

The costs are the product of the unit cost and the production volume. Thus we define two matrices: `U` contains the unit costs in Table 2.4–3 in thousands of dollars, and `P` contains the quarterly production data in Table 2.4–4.

```
>>U = [6, 2, 1;2, 5, 4;4, 3, 2;9, 7, 3];
>>P = [10, 12, 13, 15;8, 7, 6, 4;12, 10, 13, 9;6, 4, 11, 5];
```

Note that if we multiply the first column in `U` by the first column in `P`, we obtain the total materials cost for the first quarter. Similarly, multiplying the first column in `U` by the *second* column in `P` gives the total materials cost for the *second* quarter. Also, multiplying the second column in `U` by the first column in `P` gives the total *labor* cost for the first quarter, and so on. Extending this pattern, we can see that we must multiply the *transpose* of `U` by `P`. This multiplication gives the cost matrix `C`.

```
>>C = U'*P
```

The result is

$$\mathbf{C} = \begin{bmatrix} 178 & 162 & 241 & 179 \\ 138 & 117 & 172 & 112 \\ 84 & 72 & 96 & 64 \end{bmatrix}$$

Each column in **C** represents one quarter. The total first-quarter cost is the sum of the elements in the first column, the second-quarter cost is the sum of the second column, and so on. Thus because the `sum` command sums the columns of a matrix, the quarterly costs are obtained by typing

```
>>Quarterly_Costs = sum(C)
```

The resulting vector, containing the quarterly costs in thousands of dollars, is [400 351 509 355]. Thus the total costs in each quarter are $400,000; $351,000; $509,000; and $355,000.

The elements in the first row of **C** are the material costs for each quarter; the elements in the second row are the labor costs, and those in the third row are the transportation costs. Thus to find the total material costs, we must sum across the first row of **C**. Similarly, the total labor and total transportation costs are the sums across the second and third rows of **C**. Because the `sum` command sums *columns,* we must use the transpose of **C**. Thus we type the following:

```
>>Category_Costs = sum(C')
```

The resulting vector, containing the category costs in thousands of dollars, is [760 539 316]. Thus the total material costs for the year are $760,000; the labor costs are $539,000; and the transportation costs are $316,000.

We displayed the matrix **C** only to interpret its structure. If we need not display **C,** the entire analysis would consist of only four command lines.

```
>>U = [6, 2, 1;2, 5, 4;4, 3, 2;9, 7, 3];
>>P = [10, 12, 13, 15;8, 7, 6, 4;12, 10, 13, 9;6, 4, 11, 5];
>>Quarterly_Costs = sum(U'*P)
Quarterly_Costs =
   400   351   509   355
>>Category_Costs = sum((U'*P)')
Category_Costs =
   760   539   316
```

This example illustrates the compactness of MATLAB commands.

Special Matrices

NULL MATRIX

IDENTITY MATRIX

Two exceptions to the noncommutative property are the *null matrix,* denoted by **0**, and the *identity,* or *unity, matrix,* denoted by **I**. The null matrix contains all zeros and is not the same as the empty matrix [ ], which has no elements. The identity matrix is a square matrix whose diagonal elements are all equal to 1, with the remaining elements equal to 0. For example, the 2×2 identity matrix is

$$\mathbf{I} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$$

These matrices have the following properties:

$$\mathbf{0A = A0 = 0}$$
$$\mathbf{IA = AI = A}$$

MATLAB has specific commands to create several special matrices. Type `help specmat` to see the list of special matrix commands; also check Table 2.4–5. The identity matrix **I** can be created with the `eye(n)` command, where `n` is the desired dimension of the matrix. To create the 2×2 identity matrix, you type `eye(2)`. Typing `eye(size(A))` creates an identity matrix having the same dimension as the matrix **A**.

Table 2.4–5 Special matrices

Command	Description
`eye(n)`	Creates an $n \times n$ identity matrix.
`eye(size(A))`	Creates an identity matrix the same size as the matrix **A**.
`ones(n)`	Creates an $n \times n$ matrix of 1s.
`ones(m,n)`	Creates an $m \times n$ array of 1s.
`ones(size(A))`	Creates an array of 1s the same size as the array **A**.
`zeros(n)`	Creates an $n \times n$ matrix of 0s.
`zeros(m,n)`	Creates an $m \times n$ array of 0s.
`zeros(size(A))`	Creates an array of 0s the same size as the array **A**.

Sometimes we want to initialize a matrix to have all zero elements. The `zeros` command creates a matrix of all zeros. Typing `zeros(n)` creates an $n \times n$ matrix of zeros, whereas typing `zeros(m,n)` creates an $m \times n$ matrix of zeros, as will typing `A(m,n) = 0`. Typing `zeros(size(A))` creates a matrix of all zeros having the same dimension as the matrix **A**. This type of matrix can be useful for applications in which we do not know the required dimension ahead of time. The syntax of the `ones` command is the same, except that it creates arrays filled with 1s.

For example, to create and plot the function

$$f(x) = \begin{cases} 10 & 0 \leq x \leq 2 \\ 0 & 2 < x < 5 \\ -3 & 5 \leq x \leq 7 \end{cases}$$

the script file is

```
x1 = 0:0.01:2;
f1 = 10*ones(size(x1));
x2 = 2.01:0.01:4.99;
f2 = zeros(size(x2));
x3 = 5:0.01:7;
f3 = -3*ones(size(x3));
f = [f1, f2, f3];
x = [x1, x2, x3];
plot(x,f),xlabel('x'),ylabel('y')
```

(Consider what the plot would look like if the command `plot(x,f)` were replaced with the command `plot(x1,f1,x2,f2,x3,f3)`.)

Matrix Division and Linear Algebraic Equations

Matrix division uses both the right and left division operators, / and \, for various applications, a principal one being the solution of sets of linear algebraic equations. Chapter 8 covers a related topic, the matrix inverse.

You can use the left division operator (\) in MATLAB to solve sets of linear algebraic equations. For example, consider the set

$$\begin{aligned} 6x + 12y + 4z &= 70 \\ 7x - 2y + 3z &= 5 \\ 2x + 8y - 9z &= 64 \end{aligned}$$

To solve such sets in MATLAB you must create two arrays; we will call them `A` and `B`. The array `A` has as many rows as there are equations and as many columns as there are variables. The rows of `A` must contain the coefficients of x, y, and z in that order. In this example, the first row of `A` must be 6, 12, 4; the second row must be 7, −2, 3; and the third row must be 2, 8, −9. The array `B` contains the constants on the right-hand side of the equation; it has one column and as many rows as there are equations. In this example, the first row of `B` is 70, the second is 5, and the third is 64. The solution is obtained by typing `A\B`. The session is

```
>>A = [6,12,4;7,-2,3;2,8,-9];
>>B = [70;5;64];
>>Solution = A\B
Solution =
     3
     5
     -2
```

The solution is $x = 3$, $y = 5$, and $z = -2$.

LEFT DIVISION METHOD

The *left division method* works fine when the equation set has a unique solution. To learn how to deal with problems having a nonunique solution (or perhaps no solution at all!), see Chapter 8.

Test Your Understanding

T2.4–3 Use MATLAB to solve the following set of equations.

$$\begin{aligned} 6x - 4y + 8z &= 112 \\ -5x - 3y + 7z &= 75 \\ 14x + 9y - 5z &= -67 \end{aligned}$$

(Answer: $x = 2$, $y = -5$, $z = 10$.)

Matrix Exponentiation

Raising a matrix to a power is equivalent to repeatedly multiplying the matrix by itself, for example, $\mathbf{A}^2 = \mathbf{AA}$. This process requires the matrix to have the same number of rows as columns; that is, it must be a *square* matrix. MATLAB uses the symbol ^ for matrix exponentiation. To find $\mathbf{A}^2$, type `A^2`.

We can raise a scalar n to a matrix power **A**, if **A** is square, by typing `n^A`, but the applications for such a procedure are in advanced courses. However, raising a matrix to a matrix power—that is, $\mathbf{A}^{\mathbf{B}}$—is not defined, even if **A** and **B** are square.

Special Products

Many applications in physics and engineering use the cross product and dot product; for example, calculations to compute moments and force components use these special products. If **A** and **B** are vectors with three elements, the cross product command `cross(A,B)` computes the three-element vector that is the cross product $\mathbf{A} \times \mathbf{B}$. If **A** and **B** are $3 \times n$ matrices, `cross(A,B)` returns a $3 \times n$ array whose columns are the cross products of the corresponding columns in the $3 \times n$ arrays **A** and **B**. For example, the moment **M** with respect to a reference point O due to the force **F** is given by $\mathbf{M} = \mathbf{r} \times \mathbf{F}$, where **r** is the position vector from the point O to the point where the force **F** is applied. To find the moment in MATLAB, you type `M = cross(r,F)`.

The dot product command `dot(A,B)` computes a row vector of length n whose elements are the dot products of the corresponding columns of the $m \times n$ arrays **A** and **B**. To compute the component of the force **F** along the direction given by the vector **r**, you type `dot(F,r)`.

2.5 Polynomial Operations Using Arrays

MATLAB has some convenient tools for working with polynomials. Type `help polyfun` for more information on this category of commands. We will use the following notation to describe a polynomial:

$$f(x) = a_1x^n + a_2x^{n-1} + a_3x^{n-2} + \cdots + a_{n-1}x^2 + a_nx + a_{n+1}$$

We can describe a polynomial in MATLAB with a row vector whose elements are the polynomial's coefficients, *starting with the coefficient of the highest power of x*. This vector is $[a_1, a_2, a_3, \ldots, a_{n-1}, a_n, a_{n+1}]$. For example, the vector `[4,-8,7,-5]` represents the polynomial $4x^3 - 8x^2 + 7x - 5$.

Polynomial roots can be found with the `roots(a)` function, where `a` is the array containing the polynomial coefficients. For example, to obtain the roots of $x^3 + 12x^2 + 45x + 50 = 0$, you type `y = roots([1,12,45,50])`. The answer `(y)` is a *column* array containing the values $-2, -5, -5$.

The `poly(r)` function computes the coefficients of the polynomial whose roots are specified by the array `r`. The result is a *row* array that contains the polynomial's coefficients. For example, to find the polynomial whose roots are 1 and $3 \pm 5i$, the session is

```
>>p = poly([1,3+5i, 3-5i])
p =
    1    -7   40   -34
```

Thus the polynomial is $x^3 - 7x^2 + 40x - 34$.

Polynomial Addition and Subtraction

To add two polynomials, add the arrays that describe their coefficients. If the polynomials are of different degrees, add zeros to the coefficient array of the lower-degree polynomial. For example, consider

$$f(x) = 9x^3 - 5x^2 + 3x + 7$$

whose coefficient array is `f = [9,-5,3,7]` and

$$g(x) = 6x^2 - x + 2$$

whose coefficient array is `g = [6,-1,2]`. The degree of $g(x)$ is 1 less that of $f(x)$. Therefore, to add $f(x)$ and $g(x)$, we append one zero to `g` to "fool" MATLAB into thinking $g(x)$ is a third-degree polynomial. That is, we type `g = [0 g]` to obtain `[0,6,-1,2]` for `g`. This vector represents $g(x) = 0x^3 + 6x^2 - x + 2$. To add the polynomials, type `h = f+g`. The result is `h = [9,1,2,9]`, which corresponds to $h(x) = 9x^3 + x^2 + 2x + 9$. Subtraction is done in a similar way.

Polynomial Multiplication and Division

To multiply a polynomial by a scalar, simply multiply the coefficient array by that scalar. For example, $5h(x)$ is represented by `[45,5,10,45]`.

Multiplication and division of polynomials are easily done with MATLAB. Use the `conv` function (it stands for "convolve") to multiply polynomials and use the `deconv` function (`deconv` stands for "deconvolve") to perform synthetic division. Table 2.5–1 summarizes these functions, as well as the `poly`, `polyval`, and `roots` functions.

Table 2.5–1 Polynomial functions

Command	Description
`conv(a,b)`	Computes the product of the two polynomials described by the coefficient arrays `a` and `b`. The two polynomials need not be of the same degree. The result is the coefficient array of the product polynomial.
`[q,r] = deconv(num,den)`	Computes the result of dividing a numerator polynomial, whose coefficient array is `num`, by a denominator polynomial represented by the coefficient array `den`. The quotient polynomial is given by the coefficient array `q`, and the remainder polynomial is given by the coefficient array `r`.
`poly(r)`	Computes the coefficients of the polynomial whose roots are specified by the vector `r`. The result is a *row* vector that contains the polynomial's coefficients arranged in descending order of power.
`polyval(a,x)`	Evaluates a polynomial at specified values of its independent variable `x`, which can be a matrix or a vector. The polynomial's coefficients of descending powers are stored in the array `a`. The result is the same size as `x`.
`roots(a)`	Computes the roots of a polynomial specified by the coefficient array `a`. The result is a *column* vector that contains the polynomial's roots.

The product of the polynomials $f(x)$ and $g(x)$ is

$$\begin{aligned} f(x)g(x) &= (9x^3 - 5x^2 + 3x + 7)(6x^2 - x + 2) \\ &= 54x^5 - 39x^4 + 41x^3 + 29x^2 - x + 14 \end{aligned}$$

Dividing $f(x)$ by $g(x)$ using synthetic division gives a quotient of

$$\frac{f(x)}{g(x)} = \frac{9x^3 - 5x^2 + 3x + 7}{6x^2 - x + 2} = 1.5x - 0.5833$$

with a remainder of $-0.5833x + 8.1667$. Here is the MATLAB session to perform these operations.

```
>>f = [9,-5,3,7];
>>g = [6,-1,2];
>>product = conv(f,g)
product =
   54    -39    41    29    -1    14
>>[quotient, remainder] = deconv(f,g)
quotient =
   1.5    -0.5833
remainder =
   0    0    -0.5833      8.1667
```

The `conv` and `deconv` functions do not require that the polynomials be of the same degree, so we did not have to fool MATLAB as we did when adding the polynomials.

Plotting Polynomials

The `polyval(a,x)` function evaluates a polynomial at specified values of its independent variable `x`, which can be a matrix or a vector. The polynomial's coefficient array is `a`. The result is the same size as `x`. For example, to evaluate the polynomial $f(x) = 9x^3 - 5x^2 + 3x + 7$ at the points $x = 0, 2, 4, \ldots, 10$, type

```
>>f = polyval([9,-5,3,7],[0:2:10]);
```

The resulting vector `f` contains six values that correspond to $f(0), f(2), f(4), \ldots, f(10)$.

The `polyval` function is very useful for plotting polynomials. To do this, you should define an array that contains many values of the independent variable x in order to obtain a smooth plot. For example, to plot the polynomial $f(x) = 9x^3 - 5x^2 + 3x + 7$ for $-2 \le x \le 5$, you type

```
>>x = -2:0.01:5;
>>polyval([9,-5,3,7], x);
>>plot(x,f),xlabel('x'),ylabel('f(x)'),grid
```

Polynomial derivatives and integrals are covered in Chapter 9.

Test Your Understanding

T2.5–1 Use MATLAB to obtain the roots of

$$x^3 + 13x^2 + 52x + 6 = 0$$

Use the `poly` function to confirm your answer.

T2.5–2 Use MATLAB to confirm that

$$(20x^3 - 7x^2 + 5x + 10)(4x^2 + 12x - 3) = 80x^5 + 212x^4 - 124x^3 + 121x^2 + 105x - 30$$

T2.5–3 Use MATLAB to confirm that

$$\frac{12x^3 + 5x^2 - 2x + 3}{3x^2 - 7x + 4} = 4x + 11$$

with a remainder of $59x - 41$.

T2.5–4 Use MATLAB to confirm that

$$\frac{6x^3 + 4x^2 - 5}{12x^3 - 7x^2 + 3x + 9} = 0.7108$$

when $x = 2$.

T2.5–5 Plot the polynomial

$$y = x^3 + 13x^2 + 52x + 6$$

over the range $-7 \leq x \leq 1$.

EXAMPLE 2.5–1 Earthquake-Resistant Building Design

Buildings designed to withstand earthquakes must have natural frequencies of vibration that are not close to the oscillation frequency of the ground motion. A building's natural frequencies are determined primarily by the masses of its floors and by the lateral stiffness of its supporting columns (which act as horizontal springs). We can find these frequencies by solving for the roots of a polynomial called the structure's *characteristic* polynomial (characteristic polynomials are discussed further in Chapter 9). Figure 2.5–1 shows the exaggerated motion of the floors of a three-story building. For such a building, if each floor has a mass m and the columns have stiffness k, the polynomial is

$$(\alpha - f^2)[(2\alpha - f^2)^2 - \alpha^2] + \alpha^2 f^2 - 2\alpha^3$$

where $\alpha = k/4m\pi^2$ (models such as these are discussed in greater detail in [Palm, 2010]). The building's natural frequencies in cycles per second are the positive roots of this equation. Find the building's natural frequencies in cycles per second for the case where $m = 1000$ kg and $k = 5 \times 10^6$ N/m.

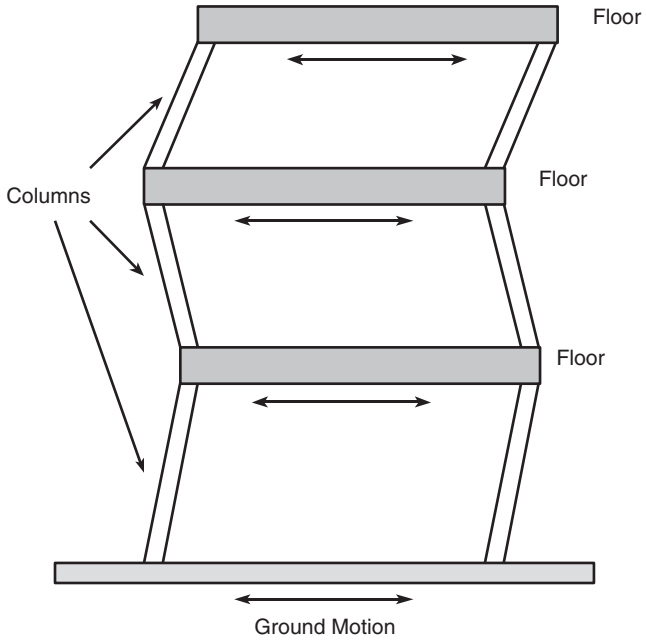

Figure 2.5–1 Simple vibration model of a building subjected to ground motion.

■ Solution

The characteristic polynomial consists of sums and products of lower-degree polynomials. We can use this fact to have MATLAB do the algebra for us. The characteristic polynomial has the form

$$p_1\left(p_2^2 - \alpha^2\right) + p_3 = 0$$

where

$$p_1 = \alpha - f^2 \qquad p_2 = 2\alpha - f^2 \qquad p_3 = \alpha^2 f^2 - 2\alpha^3$$

The MATLAB script file is

```
k = 5e+6;m = 1000;
alpha = k/(4*m*pi^2);
p1 = [-1,0,alpha];
p2 = [-1,0,2*alpha];
p3 = [alpha^2,0,-2*alpha^3];
p4 = conv(p2,p2)-(0,0,0,0,alpha^2];
p5 = conv(p1,p4);
p6 = p5+[0,0,0,0,p3];
r = roots(p6)
```

The resulting positive roots and thus the frequencies, rounded to the nearest integer, are 20, 14, and 5 Hz.

2.6 Cell Arrays

The *cell array* is an array in which each element is a *bin,* or *cell,* which can contain an array. You can store different classes of arrays in a cell array, and you can group data sets that are related but have different dimensions. You access cell arrays using the same indexing operations used with ordinary arrays.

This is the only section in the text that uses cell arrays. Coverage of this section is therefore optional. Some more advanced MATLAB applications, such as those found in some of the toolboxes, do use cell arrays.

Creating Cell Arrays

CELL INDEXING

CONTENT INDEXING

You can create a cell array by using assignment statements or by using the `cell` function. You can assign data to the cells by using either *cell indexing* or *content indexing*. To use cell indexing, enclose in parentheses the cell subscripts on the left side of the assignment statement and use the standard array notation. Enclose the cell contents on the right side of the assignment statement in braces {}.

EXAMPLE 2.6–1 An Environment Database

Suppose you want to create a 2×2 cell array `A`, whose cells contain the location, the date, the air temperature (measured at 8 A.M., 12 noon, and 5 P.M.), and the water temperatures measured at the same time in three different points in a pond. The cell array looks like the following.

Walden Pond	June 13, 1997
[60 72 65]	$\begin{bmatrix} 55 & 57 & 56 \\ 54 & 56 & 55 \\ 52 & 55 & 53 \end{bmatrix}$

■ Solution

You can create this array by typing the following either in interactive mode or in a script file and running it.

```
A(1,1) = {'Walden Pond'};
A(1,2) = {'June 13, 1997'};
A(2,1) = {[60,72,65]};
A(2,2) = {[55,57,56;54,56,55;52,55,53]};
```

If you do not yet have contents for a particular cell, you can type a pair of empty braces { } to denote an empty cell, just as a pair of empty brackets `[]` denotes an empty numeric array. This notation creates the cell but does not store any contents in it.

To use content indexing, enclose in braces the cell subscripts on the left side, using the standard array notation. Then specify the cell contents on the right side of the assignment

operator. For example:

```
A{1,1} = 'Walden Pond';
A{1,2} = 'June 13, 1997';
A{2,1} = [60,72,65];
A{2,2} = [55,57,56;54,56,55;52,55,53];
```

Type `A` at the command line. You will see

```
A =
    'Walden Pond'  'June 13, 1997'
     [1x3 double]   [3x3 double]
```

You can use the `celldisp` function to display the full contents. For example, typing `celldisp(A)` displays

```
A{1,1} =
   Walden Pond
A{2,1} =
   60  72  65
   .
   .
   .
   etc.
```

The `cellplot` function produces a graphical display of the cell array's contents in the form of a grid. Type `cellplot(A)` to see this display for the cell array `A`. Use commas or spaces with braces to indicate columns of cells and use semicolons to indicate rows of cells (just as with numeric arrays). For example, typing

```
B = {[2,4], [6,-9;3,5]; [7;2], 10};
```

creates the following 2×2 cell array:

[2 4]	$\begin{bmatrix} 6 & -9 \\ 3 & 5 \end{bmatrix}$
[7 2]	10

You can preallocate empty cell arrays of a specified size by using the `cell` function. For example, type `C = cell(3,5)` to create the 3×5 cell array `C` and fill it with empty matrices. Once the array has been defined in this way, you can use assignment statements to enter the contents of the cells. For example, type `C(2,4) = {[6,-3,7]}` to put the 1×3 array in cell (2,4) and type `C(1,5) = {1:10}` to put the numbers from 1 to 10 in cell (1,5). Type `C(3,4) = {'30 mph'}` to put the string in cell (3,4).

Accessing Cell Arrays

You can access the contents of a cell array by using either cell indexing or content indexing. To use cell indexing to place the contents of cell (3,4) of the array `C` in the new variable `Speed`, type `Speed = C(3,4)`. To place the contents of the cells in rows 1 to 3, columns 2 to 5 in the new cell array `D`, type `D = C(1:3,2:5)`. The new cell array `D` will have three rows, four columns, and 12 arrays. To use content indexing to access some of or all the contents in a *single cell,* enclose the cell index expression in braces to indicate that you are assigning the contents, not the cells themselves, to a new variable. For example, typing `Speed = C{3,4}` assigns the contents `'30 mph'` in cell (3,4) to the variable `Speed`. You cannot use content indexing to retrieve the contents of more than one cell at a time. For example, the statements `G = C{1,:}` and `C{1,:} = var`, where `var` is some variable, are both invalid.

You can access subsets of a cell's contents. For example, to obtain the second element in the 1×3-row vector in the (2,4) cell of array `C` and assign it to the variable `r`, you type `r = C{2,4}(1,2)`. The result is `r = -3`.

2.7 Structure Arrays

Structure arrays are composed of *structures.* This class of arrays enables you to store dissimilar arrays together. The elements in structures are accessed using *named fields.* This feature distinguishes them from cell arrays, which are accessed using the standard array indexing operations.

FIELD

Structure arrays are used in this text only in this section. Some MATLAB toolboxes do use structure arrays.

A specific example is the best way to introduce the terminology of structures. Suppose you want to create a database of students in a course, and you want to include each student's name, Social Security number, email address, and test scores. Figure 2.7–1 shows a diagram of this data structure. Each type of data (name, Social Security number, and so on) is a *field,* and its name is the *field name.* Thus our database has four fields. The first three fields each contain a text string, while

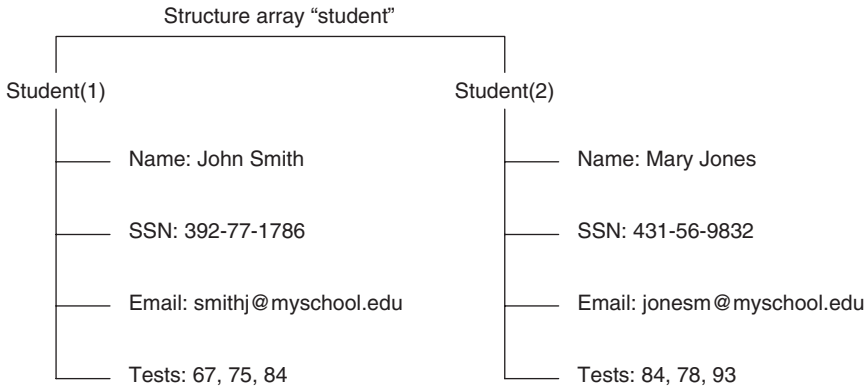

Figure 2.7–1 Arrangement of data in the structure array `student`.

the last field (the test scores) contains a vector having numerical elements. A *structure* consists of all this information for a single student. A *structure array* is an array of such structures for different students. The array shown in Figure 2.7–1 has two structures arranged in one row and two columns.

Creating Structures

You can create a structure array by using assignment statements or by using the `struct` function. The following example uses assignment statements to build a structure. Structure arrays use the dot notation (.) to specify and to access the fields. You can type the commands either in the interactive mode or in a script file.

A Student Database

EXAMPLE 2.7–1

Create a structure array to contain the following types of student data:

- Student name.
- Social Security number.
- Email address.
- Test scores.

Enter the data shown in Figure 2.7–1 into the database.

■ Solution

You can create the structure array by typing the following either in the interactive mode or in a script file. Start with the data for the first student.

```
student.name = 'John Smith';
student.SSN = '392-77-1786';
student.email = 'smithj@myschool.edu';
student.tests = [67,75,84];
```

If you then type

```
>>student
```

at the command line, you will see the following response:

```
name: 'John Smith'
SSN: = '392-77-1786'
email: = 'smithj@myschool.edu'
tests: = [67 75 84]
```

To determine the size of the array, type `size(student)`. The result is `ans = 1 1`, which indicates that it is a 1×1 structure array.

To add a second student to the database, use a subscript 2 enclosed in parentheses after the structure array's name and enter the new information. For example, type

```
student(2).name = 'Mary Jones';
student(2).SSN = '431-56-9832';
student(2).email = 'jonesm@myschool.edu';
student(2).tests = [84,78,93];
```

This process "expands" the array. Before we entered the data for the second student, the *dimension* of the structure array was 1×1 (it was a single structure). Now it is a 1×2 array consisting of two structures, arranged in one row and two columns. You can confirm this information by typing `size(student)`, which returns `ans = 1 2`. If you now type `length(student)`, you will get the result `ans = 2`, which indicates that the array has two elements (two structures). When a structure array has more than one structure, MATLAB does not display the individual field contents when you type the structure array's name. For example, if you now type `student`, MATLAB displays

```
>>student =
1x2 struct array with fields:
   name
   SSN
   email
   tests
```

You can also obtain information about the fields by using the `fieldnames` function (see Table 2.7–1). For example:

```
>>fieldnames(student)
ans =
    'name'
    'SSN'
    'email'
    'tests'
```

As you fill in more student information, MATLAB assigns the same number of fields and the same field names to each element. If you do not enter some information—for example, suppose you do not know someone's email address—MATLAB assigns an empty matrix to that field for that student.

Table 2.7–1 Structure functions

Function	Description
`names = fieldnames(S)`	Returns the field names associated with the structure array `S` as `names`, a cell array of strings.
`isfield(S,'field')`	Returns 1 if `'field'` is the name of a field in the structure array `S` and 0 otherwise.
`isstruct(S)`	Returns 1 if the array `S` is a structure array and 0 otherwise.
`S = rmfield(S,'field')`	Removes the field `'field'` from the structure array `S`.
`S = struct('f1','v1','f2', 'v2', ...)`	Creates a structure array with the fields `'f1', 'f2', ...` having the values `'v1', 'v2', ....`

The fields can he different sizes. For example, each name field can contain a different number of characters, and the arrays containing the test scores can be different sizes, as would be the case if a certain student did not take the second test.

In addition to the assignment statement, you can build structures using the `struct` function, which lets you "preallocate" a structure array. To build a structure array named `sa_1`, the syntax is

```
sa_1 = struct('field1','values1','field2',values2', ...)
```

where the arguments are the field names and their values. The values arrays `values1, values2,...` must all be arrays of the same size, scalar cells, or single values. The elements of the values arrays are inserted into the corresponding elements of the structure array. The resulting structure array has the same size as the values arrays, or is 1×1 if none of the values arrays is a cell. For example, to preallocate a 1×1 structure array for the student database, you type

```
student = struct('name','John Smith', 'SSN', ...
'392-77-1786','email','smithj@myschool.edu', ...
'tests',[67,75,84])
```

Accessing Structure Arrays

To access the contents of a particular field, type a period after the structure array name, followed by the field name. For example, typing `student(2).name` displays the value `'Mary Jones'`. Of course, we can assign the result to a variable in the usual way. For example, typing `name2 = student(2).name` assigns the value `'Mary Jones'` to the variable `name2`. To access elements within a field, for example, John Smith's second test score, type `student(1).tests(2)`. This entry returns the value `75`. In general, if a field contains an array, you use the array's subscripts to access its elements. In this example the statement `student(1).tests(2)` is equivalent to `student(1,1).tests(2)` because `student` has one row.

To store all the information for a particular structure—say, all the information about Mary Jones—in another structure array named `M`, you type `M = student(2)`. You can also assign or change values of field elements. For example, typing `student(2).tests(2) = 81` changes Mary Jones's second test score from 78 to 81.

Modifying Structures

Suppose you want to add phone numbers to the database. You can do this by typing the first student's phone number as follows:

```
student(1).phone = '555-1653'
```

All the other structures in the array will now have a `phone` field, but these fields will contain the empty array until you give them values.

To delete a field from every structure in the array, use the `rmfield` function. Its basic syntax is

```
new_struc = rmfield(array,'field');
```

where `array` is the structure array to be modified, `'field'` is the field to be removed, and `new_struc` is the name of the new structure array so created by the removal of the field. For example, to remove the Social Security field and call the new structure array `new_student`, type

```
new_student = rmfield(student,'SSN');
```

Using Operators and Functions with Structures

You can apply the MATLAB operators to structures in the usual way. For example, to find the maximum test score of the second student, you type `max(student(2).tests)`. The answer is 93.

The `isfield` function determines whether a structure array contains a particular field. Its syntax is `isfield(S, 'field')`. It returns a value of 1 (which means "true") if `'field'` is the name of a field in the structure array `S`. For example, typing `isfield(student, 'name')` returns the result `ans = 1`.

The `isstruct` function determines whether an array is a structure array. Its syntax is `isstruct(S)`. It returns a value of 1 if `S` is a structure array and 0 otherwise. For example, typing `isstruct(student)` returns the result `ans = 1`, which is equivalent to "true."

Test Your Understanding

T2.7–1 Create the structure array `student` shown in Figure 2.7–1 and add the following information about a third student: name: Alfred E. Newman; SSN: 555-12-3456; e-mail: newmana@myschool.edu; tests: 55, 45, 58.

T2.7–2 Edit your structure array to change Newman's second test score from 45 to 53.

T2.7–3 Edit your structure array to remove the SSN field.

2.8 Summary

You should now be able to perform basic operations and use arrays in MATLAB. For example, you should be able to

- Create, address, and edit arrays.
- Perform array operations including addition, subtraction, multiplication, division, and exponentiation.
- Perform matrix operations including addition, subtraction, multiplication, division, and exponentiation.

Table 2.8–1 Guide to commands introduced in Chapter 2

Special characters	Use
'	Transposes a matrix, creating complex conjugate elements.
.'	Transposes a matrix without creating complex conjugate elements.
;	Suppresses screen printing; also denotes a new row in an array.
:	Represents an entire row or column of an array.
	Tables
	Array functions Table 2.1–1
	Element-by-element operations Table 2.3–1
	Product costs Table 2.4–3
	Quarterly production volume Table 2.4–4
	Special matrices Table 2.4–5
	Polynomial functions Table 2.5–1
	Structure functions Table 2.7–1

- Perform polynomial algebra.
- Create databases using cell and structure arrays.

Table 2.8–1 is a reference guide to all the MATLAB commands introduced in this chapter.

Key Terms with Page References

Problems

You can find the answers to problems marked with an asterisk at the end of the text.

Section 2.1

1. *a.* Use two methods to create the vector **x** having 100 regularly spaced values starting at 5 and ending at 28.

b. Use two methods to create the vector **x** having a regular spacing of 0.2 starting at 2 and ending at 14.
c. Use two methods to create the vector **x** having 50 regularly spaced values starting at -2 and ending at 5.

2. *a.* Create the vector **x** having 50 logarithmically spaced values starting at 10 and ending at 1000.
b. Create the vector **x** having 20 logarithmically spaced values starting at 10 and ending at 1000.

3.* Use MATLAB to create a vector **x** having six values between 0 and 10 (including the endpoints 0 and 10). Create an array **A** whose first row contains the values $3x$ and whose second row contains the values $5x - 20$.

4. Repeat Problem 3 but make the first column of **A** contain the values $3x$ and the second column contain the values $5x - 20$.

5. Type this matrix in MATLAB and use MATLAB to carry out the following instructions.

$$\mathbf{A} = \begin{bmatrix} 3 & 7 & -4 & 12 \\ -5 & 9 & 10 & 2 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{bmatrix}$$

a. Create a vector **v** consisting of the elements in the second column of **A**.
b. Create a vector **w** consisting of the elements in the second row of **A**.

6. Type this matrix in MATLAB and use MATLAB to carry out the following instructions.

$$\mathbf{A} = \begin{bmatrix} 3 & 7 & -4 & 12 \\ -5 & 9 & 10 & 2 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{bmatrix}$$

a. Create a 4×3 array **B** consisting of all elements in the second through fourth columns of **A**.
b. Create a 3×4 array **C** consisting of all elements in the second through fourth rows of **A**.
c. Create a 2×3 array **D** consisting of all elements in the first two rows and the last three columns of **A**.

7.* Compute the length and absolute value of the following vectors:
a. $\mathbf{x} = [2, 4, 7]$
b. $\mathbf{y} = [2, -4, 7]$
c. $\mathbf{z} = [5 + 3i, -3 + 4i, 2 - 7i]$

8. Given the matrix

$$\mathbf{A} = \begin{bmatrix} 3 & 7 & -4 & 12 \\ -5 & 9 & 10 & 2 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{bmatrix}$$

a. Find the maximum and minimum values in each column.
b. Find the maximum and minimum values in each row.

9. Given the matrix

$$\mathbf{A} = \begin{bmatrix} 3 & 7 & -4 & 12 \\ -5 & 9 & 10 & 2 \\ 6 & 13 & 8 & 11 \\ 15 & 5 & 4 & 1 \end{bmatrix}$$

a. Sort each column and store the result in an array **B**.
b. Sort each row and store the result in an array **C**.
c. Add each column and store the result in an array **D**.
d. Add each row and store the result in an array **E**.

10. Consider the following arrays.

$$\mathbf{A} = \begin{bmatrix} 1 & 4 & 2 \\ 2 & 4 & 100 \\ 7 & 9 & 7 \\ 3 & \pi & 42 \end{bmatrix} \qquad \mathbf{B} = \ln(\mathbf{A})$$

Write MATLAB expressions to do the following.

a. Select just the second row of **B**.
b. Evaluate the sum of the second row of **B**.
c. Multiply the second column of **B** and the first column of **A** element by element.
d. Evaluate the maximum value in the vector resulting from element-by-element multiplication of the second column of **B** with the first column of **A.**
e. Use element-by-element division to divide the first row of **A** by the first three elements of the third column of **B**. Evaluate the sum of the elements of the resulting vector.

Section 2.2

11.* *a.* Create a three-dimensional array **D** whose three "layers" are these matrices:

$$\mathbf{A} = \begin{bmatrix} 3 & -2 & 1 \\ 6 & 8 & -5 \\ 7 & 9 & 10 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 6 & 9 & -4 \\ 7 & 5 & 3 \\ -8 & 2 & 1 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} -7 & -5 & 2 \\ 10 & 6 & 1 \\ 3 & -9 & 8 \end{bmatrix}$$

b. Use MATLAB to find the largest element in each layer of **D** and the largest element in **D**.

Section 2.3

12.* Given the matrices

$$\mathbf{A} = \begin{bmatrix} -7 & 11 \\ 4 & 9 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 4 & -5 \\ 12 & -2 \end{bmatrix} \quad \mathbf{C} = \begin{bmatrix} -3 & -9 \\ 7 & 8 \end{bmatrix}$$

Use MATLAB to

a. Find $\mathbf{A} + \mathbf{B} + \mathbf{C}$.
b. Find $\mathbf{A} - \mathbf{B} + \mathbf{C}$.
c. Verify the associative law

$$(\mathbf{A} + \mathbf{B}) + \mathbf{C} = \mathbf{A} + (\mathbf{B} + \mathbf{C})$$

d. Verify the commutative law

$$\mathbf{A} + \mathbf{B} + \mathbf{C} = \mathbf{B} + \mathbf{C} + \mathbf{A} = \mathbf{A} + \mathbf{C} + \mathbf{B}$$

13.* Given the matrices

$$\mathbf{A} = \begin{bmatrix} 56 & 32 \\ 24 & -16 \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} 14 & -4 \\ 6 & -2 \end{bmatrix}$$

Use MATLAB to

a. Find the result of **A** times **B** using the array product.
b. Find the result of **A** divided by **B** using array right division.
c. Find **B** raised to the third power element by element.

14.* The mechanical work W done in using a force F to push a block through a distance D is $W = FD$. The following table gives data on the amount of force used to push a block through the given distance over five segments of a certain path. The force varies because of the differing friction properties of the surface.

	Path segment				
	1	**2**	**3**	**4**	**5**
Force (N)	400	550	700	500	600
Distance (m)	3	0.5	0.75	1.5	5

Use MATLAB to find (*a*) the work done on each segment of the path and (*b*) the total work done over the entire path.

15. Plane A is heading southwest at 300 mi/hr, while plane B is heading west at 150 mi/hr. What are the velocity and the speed of plane A relative to plane B?

16. The following table shows the hourly wages, hours worked, and output (number of widgets produced) in one week for five widget makers.

	Worker				
	1	**2**	**3**	**4**	**5**
Hourly wage ($)	5	5.50	6.50	6	6.25
Hours worked	40	43	37	50	45
Output (widgets)	1000	1100	1000	1200	1100

Use MATLAB to answer these questions:

a. How much did each worker earn in the week?
b. What is the total salary amount paid out?
c. How many widgets were made?
d. What is the average cost to produce one widget?
e. How many hours does it take to produce one widget on average?
f. Assuming that the output of each worker has the same quality, which worker is the most efficient? Which is the least efficient?

17. Two divers start at the surface and establish the following coordinate system: x is to the west, y is to the north, and z is down. Diver 1 swims 60 ft east, then 25 ft south, and then dives 30 ft. At the same time, diver 2 dives 20 ft, swims east 30 ft and then south 55 ft.

a. Compute the distance between diver 1 and the starting point.
b. How far in each direction must diver 1 swim to reach diver 2?
c. How far in a straight line must diver 1 swim to reach diver 2?

18. The potential energy stored in a spring is $kx^2/2$, where k is the spring constant and x is the compression in the spring. The force required to compress the spring is kx. The following table gives the data for five springs:

	Spring				
	1	**2**	**3**	**4**	**5**
Force (N)	11	7	8	10	9
Spring constant k (N/m)	1000	600	900	1300	700

Use MATLAB to find (*a*) the compression x in each spring and (*b*) the potential energy stored in each spring.

19. A company must purchase five kinds of material. The following table gives the price the company pays per ton for each material, along with the number of tons purchased in the months of May, June, and July:

		Quantity purchased (tons)		
Material	**Price ($/ton)**	**May**	**June**	**July**
1	300	5	4	6
2	550	3	2	4
3	400	6	5	3
4	250	3	5	4
5	500	2	4	3

Use MATLAB to answer these questions:

a. Create a 5×3 matrix containing the amounts spent on each item for each month.
b. What is the total spent in May? in June? in July?
c. What is the total spent on each material in the three-month period?
d. What is the total spent on all materials in the three-month period?

20. A fenced enclosure consists of a rectangle of length L and width $2R$, and a semicircle of radius R, as shown in Figure P20. The enclosure is to be built to have an area A of 1600 ft^2. The cost of the fence is $40/ft for the curved portion and $30/ft for the straight sides. Use the `min` function to determine with a resolution of 0.01 ft the values of R and L required to minimize the total cost of the fence. Also compute the minimum cost.

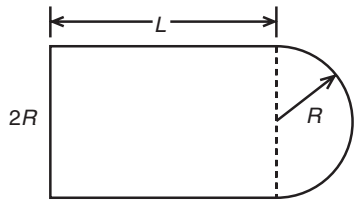

Figure P20

21. A water tank consists of a cylindrical part of radius r and height h, and a hemispherical top. The tank is to be constructed to hold 500 m^3 of fluid when filled. The surface area of the cylindrical part is $2\pi rh$, and its volume is $\pi r^2 h$. The surface area of the hemispherical top is given by $2\pi r^2$, and its volume is given by $2\pi r^3/3$. The cost to construct the cylindrical part of the tank is $300/m^2 of surface area; the hemispherical part costs $400/m^2. Plot the cost versus r for $2 \leq r \leq 10$ m, and determine the radius that results in the least cost. Compute the corresponding height h.

22. Write a MATLAB assignment statement for each of the following functions, assuming that w, x, y, and z are row vectors of equal length and that c and d are scalars.

$$f = \frac{1}{\sqrt{2\pi c/x}} \qquad E = \frac{x + w/(y + z)}{x + w/(y - z)}$$

$$A = \frac{e^{-c/(2x)}}{(\ln y)\sqrt{dz}} \qquad S = \frac{x(2.15 + 0.35y)^{1.8}}{z(1 - x)^y}$$

23. *a.* After a dose, the concentration of medication in the blood declines due to metabolic processes. The *half-life* of a medication is the time required after an initial dosage for the concentration to be reduced by one-half. A common model for this process is

$$C(t) = C(0)e^{-kt}$$

where $C(0)$ is the initial concentration, t is time (in hours), and k is called the *elimination rate constant,* which varies among individuals. For a particular bronchodilator, k has been estimated to be in the range $0.047 \leq k \leq 0.107$ per hour. Find an expression for the half-life in terms of k, and obtain a plot of the half-life versus k for the indicated range.

b. If the concentration is initially zero and a constant delivery rate is started and maintained, the concentration as a function of time is described by

$$C(t) = \frac{a}{k}(1 - e^{-kt})$$

where a is a constant that depends on the delivery rate. Plot the concentration after 1 hr, $C(1)$, versus k for the case where $a = 1$ and k is in the range $0.047 \leq k \leq 0.107$ per hour.

24. A cable of length L_c supports a beam of length L_b, so that it is horizontal when the weight W is attached at the beam end. The principles of statics can be used to show that the tension force T in the cable is given by

$$T = \frac{L_b L_c W}{D\sqrt{L_b^2 - D^2}}$$

where D is the distance of the cable attachment point to the beam pivot. See Figure P24.

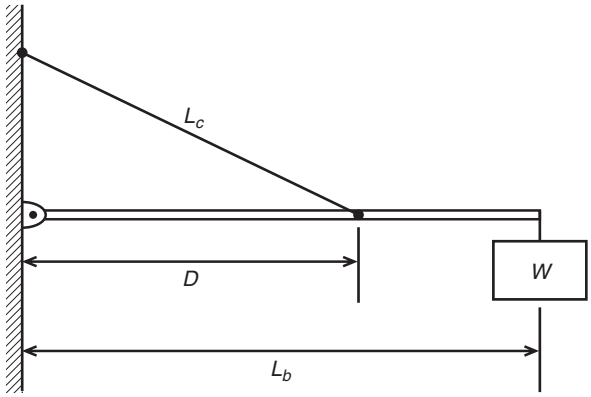

Figure P24

a. For the case where $W = 400$ N, $L_b = 3$ m, and $L_c = 5$ m, use element-by-element operations and the `min` function to compute the value of D that minimizes the tension T. Compute the minimum tension value.

b. Check the sensitivity of the solution by plotting T versus D. How much can D vary from its optimal value before the tension T increases 10 percent above its minimum value?

Section 2.4

25.* Use MATLAB to find the products **AB** and **BA** for the following matrices:

$$\mathbf{A} = \begin{bmatrix} 11 & 5 \\ -9 & -4 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} -7 & -8 \\ 6 & 2 \end{bmatrix}$$

26. Given the matrices

$$\mathbf{A} = \begin{bmatrix} 4 & -2 & 1 \\ 6 & 8 & -5 \\ 7 & 9 & 10 \end{bmatrix} \qquad \mathbf{B} = \begin{bmatrix} 6 & 9 & -4 \\ 7 & 5 & 3 \\ -8 & 2 & 1 \end{bmatrix} \qquad \mathbf{C} = \begin{bmatrix} -4 & -5 & 2 \\ 10 & 6 & 1 \\ 3 & -9 & 8 \end{bmatrix}$$

Use MATLAB to

a. Verify the associative property

$$\mathbf{A(B + C) = AB + AC}$$

b. Verify the distributive property

$$\mathbf{(AB)C = A(BC)}$$

27. The following tables show the costs associated with a certain product and the production volume for the four quarters of the business year. Use MATLAB to find (*a*) the quarterly costs for materials, labor, and transportation;

(*b*) the total material, labor, and transportation costs for the year; and (*c*) the total quarterly costs.

	Unit product costs ($\$ \times 10^3$)		
Product	**Materials**	**Labor**	**Transportation**
1	7	3	2
2	3	1	3
3	9	4	5
4	2	5	4
5	6	2	1

	Quarterly production volume			
Product	**Quarter 1**	**Quarter 2**	**Quarter 3**	**Quarter 4**
1	16	14	10	12
2	12	15	11	13
3	8	9	7	11
4	14	13	15	17
5	13	16	12	18

28.* Aluminum alloys are made by adding other elements to aluminum to improve its properties, such as hardness or tensile strength. The following table shows the composition of five commonly used alloys, which are known by their alloy numbers (2024, 6061, and so on) [Kutz, 1999]. Obtain a matrix algorithm to compute the amounts of raw materials needed to produce a given amount of each alloy. Use MATLAB to determine how much raw material of each type is needed to produce 1000 tons of each alloy.

	Composition of aluminum alloys				
Alloy	**%Cu**	**%Mg**	**%Mn**	**%Si**	**%Zn**
2024	4.4	1.5	0.6	0	0
6061	0	1	0	0.6	0
7005	0	1.4	0	0	4.5
7075	1.6	2.5	0	0	5.6
356.0	0	0.3	0	7	0

29. Redo Example 2.4–4 as a script file to allow the user to examine the effects of labor costs. Allow the user to input the four labor costs in the following table. When you run the file, it should display the quarterly costs and the category costs. Run the file for the case where the unit labor costs are \$3000, \$7000, \$4000, and \$8000, respectively.

Product costs

	Unit costs ($ × 10^3)		
Product	**Materials**	**Labor**	**Transportation**
1	6	2	1
2	2	5	4
3	4	3	2
4	9	7	3

Quarterly production volume

Product	**Quarter 1**	**Quarter 2**	**Quarter 3**	**Quarter 4**
1	10	12	13	15
2	8	7	6	4
3	12	10	13	9
4	6	4	11	5

30. Vectors with three elements can represent position, velocity, and acceleration. A mass of 5 kg, which is 3 m away from the x axis, starts at $x = 2$ m and moves with a speed of 10 m/s parallel to the y axis. Its velocity is thus described by $\mathbf{v} = [0, 10, 0]$, and its position is described by $\mathbf{r} = [2, 10t + 3, 0]$. Its angular momentum vector $\mathbf{L}$ is found from $\mathbf{L} = m(\mathbf{r} \times \mathbf{v})$, where m is the mass. Use MATLAB to

a. Compute a matrix **P** whose 11 rows are the values of the position vector **r** evaluated at the times $t = 0, 0.5, 1, 1.5, \ldots, 5$ s.

b. What is the location of the mass when $t = 5$ s?

c. Compute the angular momentum vector **L.** What is its direction?

31.* The *scalar triple product* computes the magnitude M of the moment of a force vector **F** about a specified line. It is $M = (\mathbf{r} \times \mathbf{F}) \cdot \mathbf{n}$, where **r** is the position vector from the line to the point of application of the force and **n** is a unit vector in the direction of the line.

Use MATLAB to compute the magnitude M for the case where $\mathbf{F} = [12, -5, 4]$ N, $\mathbf{r} = [-3, 5, 2]$ m, and $\mathbf{n} = [6, 5, -7]$.

32. Verify the identity

$$\mathbf{A} \times (\mathbf{B} \times \mathbf{C}) = \mathbf{B}(\mathbf{A} \cdot \mathbf{C}) - \mathbf{C}(\mathbf{A} \cdot \mathbf{B})$$

for the vectors $\mathbf{A} = 7\mathbf{i} - 3\mathbf{j} + 7\mathbf{k}$, $\mathbf{B} = -6\mathbf{i} + 2\mathbf{j} + 3\mathbf{k}$, and $\mathbf{C} = 2\mathbf{i} + 8\mathbf{j} - 8\mathbf{k}$.

33. The area of a parallelogram can be computed from $|\mathbf{A} \times \mathbf{B}|$, where **A** and **B** define two sides of the parallelogram (see Figure P33). Compute the area of a parallelogram defined by $\mathbf{A} = 5\mathbf{i}$ and $\mathbf{B} = \mathbf{i} + 3\mathbf{j}$.

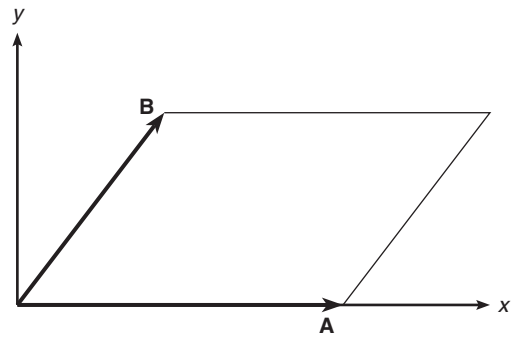

Figure P33

34. The volume of a parallelepiped can be computed from $|\mathbf{A} \cdot (\mathbf{B} \times \mathbf{C})|$, where **A**, **B**, and **C** define three sides of the parallelepiped (see Figure P34). Compute the volume of a parallelepiped defined by $\mathbf{A} = 5\mathbf{i}$, $\mathbf{B} = 2\mathbf{i} + 4\mathbf{j}$, and $\mathbf{C} = 3\mathbf{i} - 2\mathbf{k}$.

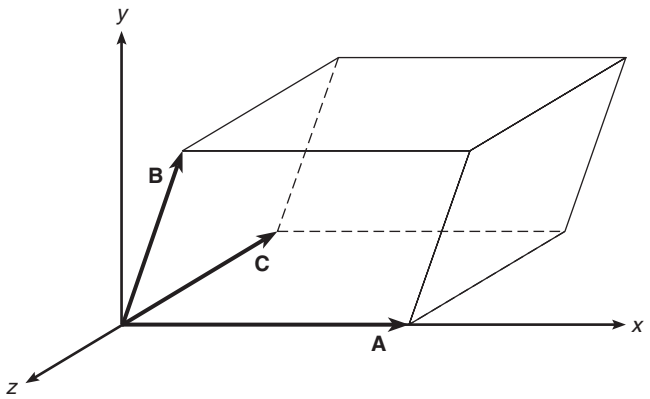

Figure P34

Section 2.5

35. Use MATLAB to plot the polynomials $y = 3x^4 - 6x^3 + 8x^2 + 4x + 90$ and $z = 3x^3 + 5x^2 - 8x + 70$ over the interval $-3 \leq x \leq 3$. Properly label the plot and each curve. The variables y and z represent current in milliamperes; the variable x represents voltage in volts.

36. Use MATLAB to plot the polynomial $y = 3x^4 - 5x^3 - 28x^2 - 5x + 200$ on the interval $-1 \leq x \leq 1$. Put a grid on the plot and use the `ginput` function to determine the coordinates of the peak of the curve.

37. Use MATLAB to find the following product:

$$(10x^3 - 9x^2 - 6x + 12)(5x^3 - 4x^2 - 12x + 8)$$

38.* Use MATLAB to find the quotient and remainder of

$$\frac{14x^3 - 6x^2 + 3x + 9}{5x^2 + 7x - 4}$$

39.* Use MATLAB to evaluate

$$\frac{24x^3 - 9x^2 - 7}{10x^3 + 5x^2 - 3x - 7}$$

at $x = 5$.

40. The *ideal gas law* provides one way to estimate the pressures and volumes of a gas in a container. The law is

$$P = \frac{RT}{\hat{V}}$$

More accurate estimates can be made with the *van der Waals* equation

$$P = \frac{RT}{\hat{V} - b} - \frac{a}{\hat{V}^2}$$

where the term b is a correction for the volume of the molecules and the term $a/\hat{V}^2$ is a correction for molecular attractions. The values of a and b depend on the type of gas. The gas constant is R, the *absolute* temperature is T, and the gas specific volume is $\hat{V}$. If 1 mol of an ideal gas were confined to a volume of 22.41 L at 0ºC (273.2 K), it would exert a pressure of 1 atm. In these units, $R = 0.08206$.

For chlorine (Cl_2), $a = 6.49$ and $b = 0.0562$. Compare the specific volume estimates $\hat{V}$ given by the ideal gas law and the van der Waals equation for 1 mol of Cl_2 at 300 K and a pressure of 0.95 atm.

41. Aircraft A is flying east at 320 mi/hr, while aircraft B is flying south at 160 mi/hr. At 1:00 P.M. the aircraft are located as shown in Figure P41.

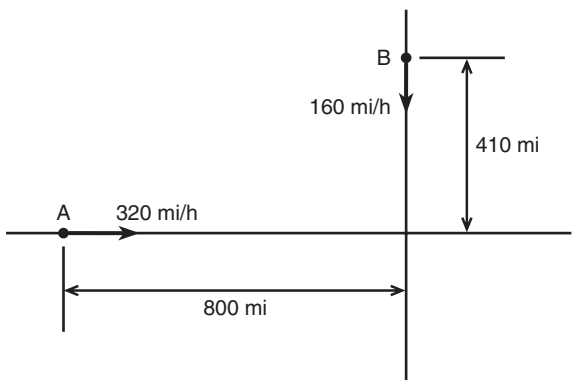

Figure P41

a. Obtain the expression for the distance D between the aircraft as a function of time. Plot D versus time until D reaches its minimum value.

b. Use the `roots` function to compute the time when the aircraft are first within 30 mi of each other.

42. The function

$$y = \frac{3x^2 - 12x + 20}{x^2 - 7x + 10}$$

approaches ∞ as $x \to 2$ and as $x \to 5$. Plot this function over the range $0 \leq x \leq 7$. Choose an appropriate range for the y axis.

43. The following formulas are commonly used by engineers to predict the lift and drag of an airfoil:

$$L = \frac{1}{2}\rho C_L S V^2$$

$$D = \frac{1}{2}\rho C_D S V^2$$

where L and D are the lift and drag forces, V is the airspeed, S is the wing span, ρ is the air density, and C_L and C_D are the *lift* and *drag* coefficients. Both C_L and C_D depend on α, the angle of attack, the angle between the relative air velocity and the airfoil's chord line.

Wind tunnel experiments for a particular airfoil have resulted in the following formulas.

$$C_L = 4.47 \times 10^{-5}\alpha^3 + 1.15 \times 10^{-3}\alpha^2 + 6.66 \times 10^{-2}\alpha + 1.02 \times 10^{-1}$$

$$C_D = 5.75 \times 10^{-6}\alpha^3 + 5.09 \times 10^{-4}\alpha^2 + 1.8 \times 10^{-4}\alpha + 1.25 \times 10^{-2}$$

where α is in degrees.

Plot the lift and drag of this airfoil versus V for $0 \leq V \leq 150$ mi/hr (you must convert V to ft /sec; there is 5280 ft/mi). Use the values $\rho = 0.002378$ slug/ft^3 (air density at sea level), $\alpha = 10^{\circ}$, and $S = 36$ ft. The resulting values of L and D will be in pounds.

44. The lift-to-drag ratio is an indication of the effectiveness of an airfoil. Referring to Problem 43, the equations for lift and drag are

$$L = \frac{1}{2}\rho C_L S V^2$$

$$D = \frac{1}{2}\rho C_D S V^2$$

where, for a particular airfoil, the lift and drag coefficients versus angle of attack α are given by

$$C_L = 4.47 \times 10^{-5}\alpha^3 + 1.15 \times 10^{-3}\alpha^2 + 6.66 \times 10^{-2}\alpha + 1.02 \times 10^{-1}$$

$$C_D = 5.75 \times 10^{-6}\alpha^3 + 5.09 \times 10^{-4}\alpha^2 + 1.81 \times 10^{-4}\alpha + 1.25 \times 10^{-2}$$

Using the first two equations, we see that the lift-to-drag ratio is given simply by the ratio C_L/C_D.

$$\frac{L}{D} = \frac{\frac{1}{2}\rho C_L S V^2}{\frac{1}{2}\rho C_D S V^2} = \frac{C_L}{C_D}$$

Plot *L/D* versus α for $-2° \leq \alpha \leq 22°$. Determine the angle of attack that maximizes *L/D*.

Section 2.6

45. *a.* Use both cell indexing and content indexing to create the following 2×2 cell array.

Motor 28C	Test ID 6
$\begin{bmatrix} 3 & 9 \\ 7 & 2 \end{bmatrix}$	[6 5 1]

b. What are the contents of the (1,1) element in the (2,1) cell in this array?

46. The capacitance of two parallel conductors of length *L* and radius *r,* separated by a distance *d* in air, is given by

$$C = \frac{\pi \epsilon L}{\ln[(d - r)/r]}$$

where ϵ is the permittitivity of air ($\epsilon = 8.854 \times 10^{-12}$ F/m). Create a cell array of capacitance values versus *d*, *L*, and *r* for $d = 0.003$, 0.004, 0.005, and 0.01 m; $L = 1$, 2, 3 m; and $r = 0.001$, 0.002, 0.003 m. Use MATLAB to determine the capacitance value for $d = 0.005$, $L = 2$, and $r = 0.001$.

Section 2.7

47. *a.* Create a structure array that contains the conversion factors for converting units of mass, force, and distance between the metric SI system and the British Engineering System.

b. Use your array to compute the following:

- The number of meters in 48 ft.
- The number of feet in 130 m.
- The number of pounds equivalent to 36 N.
- The number of newtons equivalent to 10 lb.
- The number of kilograms in 12 slugs.
- The number of slugs in 30 kg.

48. Create a structure array that contains the following information fields concerning the road bridges in a town: bridge location, maximum load (tons), year built, year due for maintenance. Then enter the following data into the array:

Location	Max. load	Year built	Due for maintenance
Smith St.	80	1928	2011
Hope Ave.	90	1950	2013
Clark St.	85	1933	2012
North Rd.	100	1960	2012

49. Edit the structure array created in Problem 48 to change the maintenance data for the Clark St. bridge from 2102 to 2018.

50. Add the following bridge to the structure array created in Problem 48.

Location	Max. load	Year built	Due for maintenance
Shore Rd.	85	1997	2014