1 THE PRIMARY MATLAB DATA STRUCTURE

As we have previously stated, the basic data element in the MATLAB system is the array. A scalar is represented as a $1 \times 1$ array—that is, an array with one row and one column. Vectors are one-dimensional arrays. An $m \times 1$ array is called a column vector, where $m$ is the number of rows in the single-column array. A $1 \times n$ array is called a row vector, where $n$ is the number of columns in the single-row array. Array elements may be numbers, characters, strings, other arrays, or structures. Recall that the elements in an array must be uniform. A special type of array, called a cell array, allows nonuniform elements.

MATLAB supports multidimensional arrays. A matrix is a special case of an array. A matrix is a rectangular array containing elements of any specified algebraic system, usually real or complex numbers. The English mathematician Arthur Cayley first introduced the concept of a matrix in the mid-19th century. Matrices are employed to help solve systems of linear equations and to perform linear transformations. This chapter describes several applications of matrix algebra to scientific and engineering problems.

MATLAB arrays are, by default, self-dimensioning. That is, if we initialize an array with a set of values, MATLAB automatically allocates the correct amount of space for the array. If you append an element to an array, MATLAB automatically resizes the array to handle the new element. This is different from many programming languages, where memory allocation and array sizing takes a considerable amount of programming effort.

SECTION

1 The Primary MATLAB Data Structure
2 Entering Arrays and Matrices
3 Accessing and Manipulation Array Elements
4 Element-by-Element Array Operations
5 Binary Matrix Operations
6 Unary Matrix Operations
7 Multidimensional Array
8 Useful Array Functions

OBJECTIVES

After reading this chapter, you should be able to

- Create arrays and matrices.
- Access elements in arrays and matrices.
- Add, modify, and delete elements from arrays.
- Perform element-by-element arithmetic operations on arrays.
- Perform vector and matrix multiplication.
- Perform matrix exponentiation.
- Compute the transpose, determinant, and inverse of a matrix.
Matrix $A$ consisting of $m$ rows and $n$ columns is said to be of order $m \times n$. If $m = n$, then matrix $A$ is called a square matrix. The following matrix $A$ is a square matrix of order three:

$$
A = \begin{bmatrix}
2 & 4 & 6 \\
3 & 5 & 7 \\
1 & 2 & 3
\end{bmatrix}
$$

Recall that you can reference the cells in a matrix by subscripts, representing the row number and column number, respectively. Thus, $A(2,3) = 7$. We call the collection of cells in a matrix for which the row numbers equal the column numbers the main diagonal. The main diagonal of $A$ is $[2, 5, 3]$.

## 2 ENTERING ARRAYS AND MATRICES

You can create arrays and matrices several ways in MATLAB. You have already been shown how to enter arrays in the Command window by typing text commands. There are several other ways to enter arrays in MATLAB. You can enter arrays by loading a script file that creates the arrays. You can view and edit arrays and matrices by using a graphical user interface called the Array Editor. Finally, you can quickly enter several types of special matrices by using some of MATLAB’s built-in matrix generators.

### 2.1 Command Line Entry

Let us review how to enter arrays in the Command window. The syntax includes brackets for the whole array and delimiters for rows and columns. Elements in the same row are separated by commas or spaces. A new row is created by using a semicolon or a new line. The whole array is bracketed by square braces. The array

$$
\begin{bmatrix}
1 & 3 & 5 \\
2 & 4 & 6 \\
7 & 7 & 7
\end{bmatrix}
$$

is entered as

```matlab
>> [1, 3, 5; 2, 4, 6; 7, 7, 7]
ans =
1     3     5
2     4     6
7     7     7
```

You can also create the same array by using spaces to separate the elements in the same row instead of commas:

```matlab
>> [1 3 5; 2 4 6; 7 7 7];
```

You can also create the same array by moving to a new line every time you designate a new row:

```matlab
>> [1 3 5
   2 4 6
   7 7 7]
```

Note that the continuation symbol (\ldots) is not required.

Moreover, you can enter array elements in series more concisely by using the colon operator.
2.2 The Array Editor

The Array Editor is a graphical interface that displays the contents of workspace objects and allows you to edit them. If the Workspace window is not visible, select View → Workspace from the Menu bar. Enter the following command in the Command window:

\[
>> A = 5 : 0.5 : 7
\]

\[
A = \\
5.0000  5.5000  6.0000  6.5000  7.0000
\]

The workspace contents will now include a \(1 \times 5\) array named \(A\) as depicted in Figure 1.

To see the Array Editor window, you should double-click anywhere on the line of the variable that you want to edit. In our example, click the mouse on the variable \(A\) in the Workspace window. The Array Editor will appear as depicted in Figure 2.

From the Array Editor, you can click on any cell in the array and edit the cell contents. You can also change the array dimensions by changing the sizes in the window titled “Size”. You can modify the display format by choosing a format from the drop-down menu titled “Numeric format”.

2.3 Formatting Output

Numbers may be formatted several ways for display on the screen. The formatting does not affect the way the numbers are stored internally. Table 1 describes the MATLAB numeric formats. The numeric formats that you choose in the Array Editor window will return to the default of type “short” when you exit MATLAB. If you want to save your favorite format, then choose File → Preferences → Array Editor from the Menu bar. Preferences saved in this manner will persist when you exit and restart MATLAB.
You can also modify the display format by using the `format` command in the Command window. In addition, you can change the display format by choosing File → Preferences → Command Window from the Menu bar. The syntax of the `format` command is

\[
\begin{align*}
\text{format} & \quad \text{format-type} \\
\text{format} & \quad (\text{\text NumberOfSpaces} \text{\text format-type}) \\
\text{format} &
\end{align*}
\]

where `format-type` is one of the types listed in Table 1. Note that every MATLAB command may be represented in command form or functional form. The command form uses the command name followed by one or more spaces and then uses the command arguments—for example,

\[
\begin{align*}
\text{>> format long}
\end{align*}
\]

The functional form uses the command named, followed by the arguments (if any) in parentheses. When using the functional form, you must place quotes around a string argument—for example,

\[
\begin{align*}
\text{>> format (\text{\text 'long'})}
\end{align*}
\]

We will not repeat the functional form in every example in the text, but it is understood that it may be used in place of the command form. The functional form is useful when programming because it allows you to manipulate the argument as a variable.

The `format` command without arguments resets the format to the default type. The default format is “short”.

In addition to numeric formats, the compact and loose formats may be used to add or to delete extra line feeds in the display. Here are some examples:

\[
\begin{align*}
\text{>> format loose} \\
\text{>> log(2) ans =} & \quad 0.6931 \\
\text{>> format compact} \\
\text{>> log(2) ans =} & \quad 0.6931
\end{align*}
\]

\[
\begin{table}
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Type} & \textbf{Format} & \textbf{Precision} \\
\hline
short & fixed point, scaled & 5 digits \\
short e & floating point & 5 digits \\
short g & fixed or floating point (most accurate) & as many significant figures as possible with 5 digits \\
long & fixed point, scaled & 15 digits \\
long e & floating point & 15 digits \\
long g & fixed or floating point (most accurate) & as many significant figures as possible with 15 digits \\
rat & rational expression & approximates a rational fraction \\
hex & hexadecimal & base 16 \\
\hline
\end{tabular}
\caption{Numeric Formats.}
\end{table}
\]
If you forget what format you are utilizing, you can display the current format by using the `get` function. The `get` function gets an object attribute—in this case, the `format` attribute. The first argument is the number of the graphics object that contains the attribute. A graphics object in MATLAB is simply a named graphical structure (e.g., the Command window). The object number for the Command window screen is 0, as shown here:

```matlab
>> get(0, 'format')
ans =
long
```

As an aside, you can see all of the attributes for an object by using the `get` function with the object number alone as an argument. The following command returns all of the attributes of the Command window:

```matlab
>> get(0)
CallbackObject = []
Language = english
CurrentFigure = []
Diary = off
DiaryFile = diary
Echo = off
ErrorMessage = Error: Expected a variable, function, or constant, found ”)

FixedWidthFontName = Courier
Format = long
...
(many more lines of output)
```

### 2.4 Built-In Matrix Generators

We use several types of arrays so frequently that MATLAB has provided special functions to support the generation of these arrays. We call a matrix in which all of the elements are zero a zero matrix. You can create a zero array or matrix by using the `zeros` function. The syntax of the `zeros` function is

```matlab
zeros(dim1, dim2, dim3, ...)
```

If you specify a single, scalar parameter `dim1`, MATLAB returns a square zero matrix of order `dim1`. For example, the following command creates a $2 \times 2$ matrix of zeros:

```matlab
>> A = zeros(2)
A =
   0     0
   0     0
```

If you specify multiple parameters, a multidimensional zero array of order `dim1 \times dim2 \times dim3`... is returned. The following command creates a $2 \times 4$ matrix of zeros:

```matlab
>> A = zeros(2, 4)
A =
   0     0     0     0
   0     0     0     0
```
We call a matrix in which all of the elements are the number one a \textit{ones matrix}. You can create a ones matrix by using the \textit{ones} function. The syntax of the \textit{ones} function is identical to the syntax of the \textit{zeros} function. The following command creates a $3 \times 2$ matrix of ones:

\begin{verbatim}
>> A = ones(3, 2)
A =
  1  1
  1  1
  1  1
\end{verbatim}

Similarly, you can generate an array of pseudorandom numbers by using one of MATLAB's several random array generator functions. One of these, the \textit{rand} function, generates an array of random numbers whose elements are uniformly distributed in the range $(0, 1)$. A uniform distribution is one in which there is an equal probability of occurrence for any value within the given range $(0, 1)$—for example,

\begin{verbatim}
>> A = rand(2,5)
A =
  0.9501  0.6068  0.8913  0.4565  0.8214
  0.2311  0.4860  0.7621  0.0185  0.4447
\end{verbatim}

Another commonly used matrix form is the identity matrix. An \textit{identity matrix} is a matrix in which every element of the main diagonal is one and every other element is zero. You can generate an $n \times n$ identity matrix by using the \textit{eye} function with the syntax, as shown here:

\begin{verbatim}
eye(n)
\end{verbatim}

Here is an example:

\begin{verbatim}
>> eye(3)
an =
  1  0  0
  0  1  0
  0  0  1
\end{verbatim}

You can use two arguments to specify both dimensions. An $m \times n$ identity matrix with ones on the diagonal and zeros elsewhere can be generated by using the syntax

\begin{verbatim}
eye(m, n)
\end{verbatim}

For example, we might have

\begin{verbatim}
>> eye(4,3)
an =
  1  0  0
  0  1  0
  0  0  1
  0  0  0
\end{verbatim}

The \textit{eye} function does not support more than two dimensions. Specifying more than two dimensions will result in a syntax error—for example,

\begin{verbatim}
>> A = eye(3,4,5)
??? Error using ==> eye
Too many input arguments.
\end{verbatim}
3 ACCESSING AND MANIPULATING ARRAY ELEMENTS

3.1 Accessing Elements of an Array

You have already accessed array elements by using subscripts. Let us review what you have learned and cover a few more tricks for accessing array elements. We will use the following two-dimensional array A for the next few examples:

```matlab
>> A = [1 3 5; 2 4 6; 3 5 7]
A =
    1     3     5
    2     4     6
    3     5     7
```

An element in a two-dimensional array can be accessed by using two subscripts, the first for the row and the second for the column; for example,

```matlab
>> A(2,3)
an =
    6
```

You can also access elements in a two-dimensional array by using a single subscript. In this case, imagine the columns lay end to end as follows:

```matlab
A = [ 1
      2
      3
      3
      4
      5
      5
      6
      7 ]
```

This makes more sense if we look at how MATLAB stores arrays internally. Data are stored internally in a linear sequence of memory locations. MATLAB stretches out an array into a single sequence for storage.

We think of array A as a two-dimensional array. If A were stored one row at a time in memory, it might look like this:

```
1 3 5 2 4 6 3 5 7
```

This is called \textit{row-major order}. However, if A were stored one column at a time in memory, it would look like the following:

```
1 2 3 3 4 5 5 6 7
```

This is called \textit{column-major order}. MATLAB stores arrays in column-major order. MATLAB functions are written to take advantage of the underlying storage mechanism to speed up array operations.

The following examples demonstrate how to access an array element by using a single subscript:

```matlab
>> A(1)
an =
    1
>> A(4)
```
We have already used the colon operator to generate arrays. You can also use the colon operator to access multiple array elements simultaneously. You do this by using the colon operator to define a subscript range. For example, the use of 1:2:9 as a subscript returns the first, third, fifth, seventh, and ninth elements of A:

\[
>> A(1:2:9) \\
\text{ans} = \\
1 \\
3 \\
4 \\
5 \\
7
\]

When used alone, the colon denotes all rows or columns. The following command returns all columns of row two from array A:

\[
>> A(2,:) \\
\text{ans} = \\
2 \\
4 \\
6
\]

The following command returns the second and third rows of the first and second columns from array A:

\[
>> A(2:3, 1:2) \\
\text{ans} = \\
2 \\
4 \\
3 \\
5
\]

### 3.2 Expanding the Size of an Array

You can dynamically expand an array simply by adding more elements—for example,

\[
>> A = [3 5 7] \\
A = \\
3 \\
5 \\
7 \\
>> A = [A 9] \\
A = \\
3 \\
5 \\
7 \\
9
\]

When appending arrays to multidimensional arrays, the newly appended parts must conform to the dimensions of the original array. For example, if adding a new row to a two-dimensional array, the row must have the same number of columns as the original array:

\[
>> A = [3 5 7]; \\
>> B = [1 3 5]; \\
>> C = [A; B] \\
C = \\
3 \\
5 \\
7 \\
1 \\
3 \\
5
\]

If you try to append to an array and the appended part does not conform dimensionally, an error will result. Here's an example:

\[
>> A = [3 5 7]; \\
>> B = [2 4];
\]
Accessing and Manipulating Array Elements

PROGRAMMING TIP 1!

MATLAB supports preallocation of arrays by allowing the creation of an array that is filled with all zeros or ones. If you are using very large arrays in your programs, preallocation is more efficient than slowly growing an array.

If you know the size of your array ahead of time (e.g., 20,000), create a zero-filled array by using

\[
\text{zero}(20000);
\]

which is much faster than extending the size of the array one element at a time.

>> C = [A; B]
??? Error using ==> vertcat
All rows in the bracketed expression must have the same number of columns.

3.3 Deleting Array Elements

You can delete array elements by replacing them with the empty array, which we designate as [ ]. In the following example, the second element of vector A is removed:

\[
\begin{align*}
\text{>> A} &= [3 \ 5 \ 7]; \\
\text{>> A}(2) &= [ ] \\
\text{A} &= 3 \ 7
\end{align*}
\]

You cannot remove a single element from a multidimensional array, since the array would no longer be conformant. This results in an error, as shown in this example:

\[
\begin{align*}
\text{>> A} &= [1 \ 3 \ 5; \ 2 \ 4 \ 6] \\
\text{A} &= \\
&\begin{bmatrix} 
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix} \\
\text{>> A}(2,3) &= [ ] \\
??? \text{ Indexed empty matrix assignment is not allowed.}
\end{align*}
\]

You can use the colon operator in deletion operations. The colon operator allows deletion of whole rows or columns. In the next example, the second row of the $2 \times 3$ array A is removed:

\[
\begin{align*}
\text{>> A} &= [1 \ 3 \ 5; \ 2 \ 4 \ 6] \\
\text{A} &= \begin{bmatrix} 
1 & 3 & 5 \\
2 & 4 & 6
\end{bmatrix} \\
\text{>> A}(2,:) &= [ ] \\
\text{A} &= \begin{bmatrix} 
1 & 3 & 5
\end{bmatrix}
\end{align*}
\]

The following example removes the first, third, and fifth columns from array A:

\[
\begin{align*}
\text{>> A} &= [1 \ 2 \ 3 \ 4 \ 5 \ 6; \ 7 \ 8 \ 9 \ 10 \ 11 \ 12] \\
\text{A} &= \begin{bmatrix} 
1 & 2 & 3 & 4 & 5 & 6 \\
7 & 8 & 9 & 10 & 11 & 12
\end{bmatrix}
\end{align*}
\]
>> A(:, 1:2:5) = []
A =
  2  4  6
  8 10 12

PRACTICE 1!

Let array

>> A = [ 1 0 1 0
         0 2 0 2
         3 1 3 1 ]

Write commands that will perform each of the following operations on array A:

1. Return the second column of A.
2. Return the first and third rows of A.
3. Delete the first and second columns of A.
4. Append the column vector [7; 8; 9] to A.

Re-create array A again before each problem. Check your answers by using MATLAB.

4 ELEMENT-BY-ELEMENT ARRAY OPERATIONS

4.1 Array Addition

MATLAB performs addition or subtraction of two arrays of the same order by adding or subtracting each pair of respective elements. The result is an array of the same order. For example, given that

\[ A = [1 \ 3 \ 5] \text{ and } B = [10 \ 12 \ 14] \]

\[ A + B \text{ is calculated as follows:} \]

\[ [A(1) + B(1) \ A(2) + B(2) \ A(3) + B(3)] = [11 \ 15 \ 19]. \]

Here is another example, this time we are using two-dimensional arrays. Given that

\[ A = \begin{bmatrix} 1 & 3 & 5 \\ 2 & 4 & 6 \\ 7 & 7 & 7 \end{bmatrix} \]

and

\[ B = \begin{bmatrix} -5 & 6 & 14 \\ 0 & -2 & 4 \\ 2 & 8 & 3 \end{bmatrix} \]

the sum of A and B is

\[ A + B = \begin{bmatrix} -4 & 9 & 19 \\ 2 & 2 & 10 \\ 9 & 15 & 10 \end{bmatrix} \]

If two arrays are not of the same order, we say they are not conformable for addition or subtraction. For example, the following matrices C and D are not conformable for addition or subtraction because C’s dimensions are \(1 \times 3\) and D’s dimensions are \(2 \times 3\):
As you see, attempting to add them will result in an error.
The addition of arrays is commutative—that is,
\[ A + B = B + A. \]

The addition and subtraction of arrays is associative—that is,
\[ A + (B + C) = (A + B) + C. \]

### 4.2 Array Multiplication

MATLAB performs array multiplication by multiplying each pair of respective elements in two arrays of the same order. The symbol for array multiplication is a period followed by an asterisk (\(.*\)). The following example demonstrates array multiplication:

\[
\begin{align*}
\gg A &= [1, 3, 5; 2, 4, 6] \\
A &= \\
&\begin{bmatrix} 1 & 3 & 5 \\
2 & 4 & 6 \\
\end{bmatrix} \\
\gg B &= [2, 3, 4; -1, -2, -3] \\
B &= \\
&\begin{bmatrix} 2 & 3 & 4 \\
-1 & -2 & -3 \\
\end{bmatrix} \\
\gg A .* B \\
\text{ans} &= \\
&\begin{bmatrix} 2 & 9 & 20 \\
-2 & -8 & -18 \\
\end{bmatrix}
\end{align*}
\]

To be conformable for array multiplication, the two arrays must be of the same order, unless one array is a scalar, in which case, each element of the other array is multiplied by the scalar—for example,

\[
\begin{align*}
\gg A &= [5] \\
A &= \\
&5 \\
\gg B &= [2, 4, 6] \\
B &= \\
&\begin{bmatrix} 2 & 4 & 6 \\
\end{bmatrix} \\
\gg A .* B \\
\text{ans} &= \\
&\begin{bmatrix} 10 & 20 & 30 \\
\end{bmatrix}
\end{align*}
\]
In this case (where at least one operand is a scalar), the period before the multiplication symbol is not required. The “*” alone will produce the same result. (See Section 5.2, titled “Matrix Multiplication” for details.) Here’s an example:

```plaintext
>> A*B
ans =
   10   20   30
```

4.3 Array Right Division

MATLAB performs array right division of arrays \( A \) and \( B \) by dividing each element in array \( A \) by the respective element in array \( B \). The symbol for array right division is a period followed by a forward slash (\( ./ \)). To be conformable for array right division, the two arrays must be of the same order, unless one array is a scalar. The following example demonstrates array right division:

```plaintext
>> A = [2, 4, 6]
A =
  2   4   6
>> B = [2, 2, 2]
B =
  2   2   2
>> A./B
ans =
  1   2   3
```

4.4 Array Left Division

MATLAB performs array left division of arrays \( A \) and \( B \) by dividing each element in array \( B \) by the respective element in array \( A \). The symbol for array left division is a period followed by a back slash (\( .\)\). To be conformable for array left division, the two arrays must be of the same order, unless one array is a scalar. The following example demonstrates array left division, using the arrays \( A \) and \( B \) from the previous example:

```plaintext
>> A.\B
ans =
   1.0000   0.5000   0.3333
```

4.5 Array Exponentiation

MATLAB performs array exponentiation of arrays \( A \) and \( B \) by raising each element in array \( A \) to the power of its respective element in array \( B \). The symbol for array exponentiation is a period followed by the caret symbol (\( .^\)\). To be conformable for array exponentiation, the two arrays must be of the same order, unless one array is a scalar. The following example demonstrates array exponentiation:

```plaintext
>> A = [2, 3, 4]
A =
  2   3   4
>> B = [3, 2, 0.5]
B =
  3.0000   2.0000   0.5000
```
A binary operation is a mathematical computation performed by using two matrices as inputs. Binary matrix operations are not as straightforward to compute as element-by-element operations. Binary matrix operations have many applications, such as the solution of systems of linear equations.

5.1 Vector Multiplication

We will first describe vector multiplication mathematically and then show you how to perform the operation by using MATLAB. Two vectors $a$ and $b$ are multiplied by computing their dot product. The dot product, sometimes called the inner product, is calculated by adding the products of each pair of respective elements in vectors $a$ and $b$. To be conformable for vector multiplication $a$ must be a row vector and $b$ must be a column vector. In addition, the vectors must contain the same number of elements, unless one is a scalar. If row vector

$$a = [a_1 \ a_2 \ \ldots \ \ a_n],$$

and column vector

$$b = \begin{bmatrix} \ b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

the dot product

$$a \cdot b = a_1b_1 + a_2b_2 + \ldots + a_nb_n.$$
The MATLAB symbol for vector multiplication is the asterisk (*)—for example,

\[
A = [1, 5, -6] \\
A = 
\begin{bmatrix} 
1 & 5 & -6 
\end{bmatrix} \\
B = [-2; -4; 0] \\
B = 
\begin{bmatrix} 
-2 \\
-4 \\
0 
\end{bmatrix} \\
C = A \times B \\
C = -22
\]

The result was calculated as follows:

\[
A*B = (1*-2) + (5*-4) + (-6*0) = -22
\]

Note how this differs from array multiplication, which would fail, since \(A\) and \(B\) are not conformable for array multiplication.

If you attempt to use nonconformable vectors, MATLAB returns an error. Here's an example:

\[
\begin{bmatrix} 
1 & 2 & 3 
\end{bmatrix} \\
\begin{bmatrix} 
2 & 3 & 4 
\end{bmatrix}
\]

\[
??? Error using ==> * \\
Inner matrix dimensions must agree.
\]

5.2 Matrix Multiplication
MATLAB performs the multiplication of matrix \(A\) by a scalar by multiplying each element of \(A\) by the scalar. Any array or matrix can be multiplied by a scalar. The following is an example:

\[
A = [1, 3; -2, 0] \\
A = 
\begin{bmatrix} 
1 & 3 \\
-2 & 0 
\end{bmatrix} \\
B = A \times 5 \\
B = 
\begin{bmatrix} 
5 & 15 \\
-10 & 0 
\end{bmatrix}
\]

MATLAB performs multiplication of nonscalar \(A\) and \(B\) by computing the dot products of each row in \(A\) with each column in \(B\). Each result becomes a row in the resulting matrix. We will try to make this clearer by walking through an example:

\[
\begin{bmatrix} 
1 & 3 & 5 \\
2 & 4 & 6 
\end{bmatrix}
\]
$$\text{Note that the number of rows in } A(m_A = 2) \text{ equals the number of columns in } B(n_B = 2). \text{ To be conformable for matrix multiplication, the number of rows in } A \text{ must equal the number of columns in } B. \text{ The result will be an } m_A \times n_B \text{ matrix. In the example, the result will be a } 2 \times 2 \text{ matrix.}
$$

The first step is to compute the dot product of row one of } A \text{ and column one of } B:\n
$$(1 \times -2) + (3 \times 3) + (5 \times 12) = 67$$

Place the result in cell (1, 1) of the result matrix. Next, compute the dot product of row one of } A \text{ and column two of } B:\n
$$(1 \times 4) + (3 \times 8) + (5 \times -2) = 18$$

Place the result in cell (1, 2) of the result matrix. Next, compute the dot product of row two of } A \text{ and column one of } B:\n
$$(2 \times -2) + (4 \times 3) + (6 \times 12) = 80$$

Place the result in cell (2, 1) of the result matrix. Finally, compute the dot product of row two of } A \text{ and column two of } B:\n
$$(2 \times 4) + (4 \times 8) + (6 \times -2) = 28$$

Place the result in cell (2, 2) of the result matrix. The resulting product is

$$\gg \text{ A*B}
\begin{array}{c}
\text{ans =}
\end{array}
\begin{bmatrix}
67 \\
80 \\
18 \\
28
\end{bmatrix}$$

For most cases of } A \text{ and } B, \text{ matrix multiplication is not commutative; that is,}

$$AB \neq BA.$$  

### 5.3 Matrix Division

The operations for left and right matrix division are not straightforward. We will not walk through the underlying algorithm for their computation in this text. However, we will show you an application of the left matrix division operator.

A common and useful application of matrices is the representation of systems of linear equations. The linear system

$$\begin{align*}
3x_1 + 2x_2 + x_3 &= 5 \\
x_1 + 2x_2 + 3x_3 &= 13 \\
-5x_1 - 10x_2 - 5x_3 &= 0
\end{align*}$$

can be represented compactly as the matrix product } AX = B:

$$\begin{bmatrix}
3 & 2 & 1 \\
1 & 2 & 3 \\
-5 & -10 & -5
\end{bmatrix}
\begin{bmatrix}
x_1 \\
x_2 \\
x_3
\end{bmatrix}
= 
\begin{bmatrix}
5 \\
13 \\
0
\end{bmatrix}$$
MATLAB uses a complex algorithm to compute the solution to a linear system of the form $AX = B$. The operation is denoted by the matrix left division operator (the back-slash) $X = A\backslash B$.

The solution to the preceding linear system can be determined as follows:

```matlab
>> A = [3 2 1; 1 2 3; -5 -10 -5];
>> B = [5; 13; 0];
>> X = A \ B
X =
  2.5000
-4.5000
  6.5000
```

Verify that MATLAB produced a correct answer by substituting the results into the original three equations. You will learn more about solutions to linear systems when you take a course in linear algebra.

### 6 UNARY MATRIX OPERATIONS

Unary matrix operations are mathematical computations that are performed by using a single matrix as an input.

#### 6.1 Transpose

We call the matrix that is created by exchanging the rows and columns of matrix $A$ the transpose of $A$. For example, given

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$$
the transpose of $A$, denoted in mathematics as $A^T$, is

$$A^T = \begin{bmatrix} 1 & 4 & 7 \\ 2 & 5 & 8 \\ 3 & 6 & 9 \end{bmatrix}$$

The MATLAB prime operator (‘’) returns the transpose of its argument—for example,

>> A = [1, 2, 3; 4, 5, 6; 7, 8, 9]
A =
1     2     3
4     5     6
7     8     9
>> A'
ans =
1     4     7
2     5     8
3     6     9

6.2 Determinant

The determinant of a matrix is a transformation of a square matrix that results in a scalar. We denote the determinant of a matrix $A$ mathematically as $|A|$ or $\text{det} A$. In this text, we will use the second notation, since it resembles the MATLAB function for computing a determinant.

If a matrix has a single entry, then the determinant of the matrix is the value of the entry. For example, if $A = [3]$, the determinant of $A = 3$. We write this as

$$\text{det} A = 3.$$

If a square matrix $A$ has order 2, then the determinant of $A$ is calculated as follows:

$$\text{det} \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} = a_{11} \cdot a_{22} - a_{21} \cdot a_{12}$$

MATLAB has a function that computes the determinant named $\text{det}$. The syntax for the $\text{det}$ function is

$$\text{det} \ (A)$$

where $A$ must be a square matrix—for example,

A =
2     3
6     4
>> det(A)
an =
-10

First, we will show you how to calculate mathematically the determinant of a matrix with order $n > 2$. Then we will show you how to use MATLAB to perform the same computation.

The strategy for calculating the determinant of a matrix with order $n > 2$ involves subdividing the matrix into smaller sections called minors and cofactors. If row $i$ and column $j$ of a square matrix $A$ are deleted, the determinant of the resulting matrix is called
the minor of \( a_{ij} \). We denote the minor as \( M_{ij} \). For example, given

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

then the minor of \( a_{12} \) (deleting row 1 and column 2) is

\[
M_{12} = \det \begin{bmatrix}
4 & 6 \\
7 & 9
\end{bmatrix}
\]

The cofactor of \( a_{ij} \) is denoted as \( A_{ij} \) and is calculated as follows:

\[
A_{ij} = (-1)^{i+j} M_{ij}
\]

In our example, the cofactor of \( a_{12} \) is

\[
A_{12} = (-1)^{1+2} (4 \cdot 9 - (6 \cdot 7)) = 6.
\]

The general form for the calculation of a determinant is

\[
\det A = a_{11}A_{11} + a_{12}A_{12} + \cdots + a_{nn}A_{nn}
\]

where \( i \) is any row in square matrix \( A \) of order \( n \). The answer is the same no matter which row is chosen. A similar formula works by choosing any column in \( A \). Let us follow the example and expand \( A \) around row 2:

\[
\begin{align*}
\det A &= 4 \cdot (-1)^{3+1} \cdot \det \begin{bmatrix} 2 & 3 \\ 8 & 9 \end{bmatrix} + 5 \cdot (-1)^{2+2} \cdot \det \begin{bmatrix} 1 & 3 \\ 7 & 9 \end{bmatrix} \\
&\quad + 6 \cdot (-1)^{2+3} \cdot \det \begin{bmatrix} 1 & 2 \\ 7 & 8 \end{bmatrix} \\
&= (-4 \cdot -6) + (5 \cdot -12) + (-6 \cdot -6) \\
&= 0.
\end{align*}
\]

We find that by using MATLAB to compute the determinant of \( A \) results in

\[
A = \begin{bmatrix}
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9
\end{bmatrix}
\]

\[
>> \text{det (A)}
\]

\[
\text{ans} = 0
\]

As you can see, the determinant of a high order matrix is tedious to calculate by hand. Moreover, because the calculation of a higher order determinant is computationally intensive and involves a series of recursive steps, the rounding error can be significant.

### 6.3 Inverse

The inverse of a square matrix \( A \), if it exists, is defined to be a square matrix such that

\[
AA^{-1} = I,
\]

where \( I \) is the identity matrix of the same order as \( A \). The matrix inverse operation is denoted mathematically by using a negative one exponent, \( A^{-1} \).
There is a method for determining if and when the inverse of a matrix exists. It depends on understanding the concept of matrix singularity. A square matrix is singular if and only if its determinant is equal to zero. Otherwise, a matrix is nonsingular. Furthermore, a square matrix has an inverse if and only if it is nonsingular. So, a square matrix $A$ has an inverse if and only if $det(A) \neq 0$.

However, on a computer, zero is not always zero. Computer representations of real numbers are usually approximations. Thus, calculations can result in highly accurate, but approximate results. If the determinant of a matrix is close to zero, MATLAB will give a warning that the inverse of $A$ may not be correct.

The syntax for MATLAB’s inverse function `inv` is

```
inv(square-matrix)
```

Recall from the previous example that the determinant of the matrix

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

is singular (i.e., $det(A) = 0$) and should not have an inverse. MATLAB returns a warning noting this:

```
>> inv(A)
Warning: Matrix is close to singular or badly scaled.
Results may be inaccurate. RCOND = 1.541976e-018.
ans =
   1.0e+016 *
-0.4504    0.9007   -0.4504
   0.9007   -1.8014    0.9007
-0.4504    0.9007   -0.4504
```

### 6.4 Matrix Exponentiation

MATLAB computes the positive integer power of a square matrix $A$ by multiplying $A$ times itself the requisite number of times. The multiplication operation that is performed is matrix multiplication, not element-by-element multiplication—for example,

```
>> A = [1, 2; 3, 4]
A =
   1    2
   3    4
>> A^2
ans =
    7    10
   15    22
>> A^3
ans =
   37    54
   81   118
```

The negative integer power of a square matrix $A$ is computed by performing matrix multiplication of the inverse of $A$ the requisite number of times. For example, to compute the second negative root of $A$, we type
Arrays and Matrix Operations

>> A^-2
ans =
5.5000  -2.5000
-3.7500   1.7500

This only works if the matrix is nonsingular. MATLAB issues a warning if the computed
determinant of A is equal or very close to zero. Here’s an example:

>> A = [1,1; 0,0]
A =
1 1
0 0
>> det(A)
ans =
0
>> A^-2
Warning: Matrix is singular to working precision.
ans =
Inf   Inf
Inf   Inf

PRACTICE 4!

Given the square matrices

A = [ 2 0; 1 -5]
B = [ 3 -2 0; 4 1 5; 0 -3 4]

compute the following operations by hand and then check your answers by using
MATLAB:
1. A'
2. det(A)
3. B'
4. det(B)

Compute the following with MATLAB:
5. A^2
6. inv(A)
7. inv(B)
8. A^-2

7 MULTIDIMENSIONAL ARRAYS

We have previously used examples of two-dimensional arrays. Many of MATLAB’s array
operations can be extended to more than two dimensions.

The following command creates a three-dimensional array of order 2 × 3 × 2. Since
MATLAB cannot display the whole array at once, it displays the array a page at a
time. There are two pages in the following example, as the third dimension takes two levels:

>> A = ones(2,3,2)
A(:,:,1) =
1 1 1
1 1 1
2 * 3 * 2.
MATLAB contains scores of useful functions for manipulating and extracting information from arrays. This section presents a few of the most commonly used array functions.

**ndims**

The `ndims` function returns the number of dimensions of its argument—for example,

```matlab
>> A = ones(2,3,2);
>> ndims(A)
an = 3
```

**size**

The `size` function returns the length of each dimension, or the order of the array. The result is a vector that contains the size of dimension 1, dimension 2, dimension 3, etc. Here’s an example,

```matlab
>> A = zeros(2,3,2,4);
>> size(A)
an = 2 3 2 4
```

You can also use the `size` function to return the size of each dimension to a separate variable—for example,

```matlab
>> [m, n, s, t] = size(A)
m = 2
n = 3
s = 2
t = 4
```

**diag**

The `diag` function returns the elements of the main diagonal. For a matrix, `diag` returns the elements with equal row and column indices (i.e., elements (1,1), (2,2), (3,3), etc.):

```matlab
>> A = [1 3 5; 2 4 6; 0 2 4]
A =
    1 3 5
    2 4 6
    0 2 4
```
The main diagonal is also called the zero diagonal. A second argument may be passed to \texttt{diag} that specifies the \textit{n\textsuperscript{th}} diagonal above or below zero. If the second argument is positive, the \textit{n\textsuperscript{th}} diagonal above the zero diagonal is returned, as in this example:

\begin{verbatim}
>> diag(A,1)
ans =
  3
  6
\end{verbatim}

If the second argument is negative, the \textit{n\textsuperscript{th}} diagonal below the zero diagonal is returned. Here’s an example:

\begin{verbatim}
>> diag(A,-1)
ans =
  2
  2
\end{verbatim}

\textit{length}

The \textit{length} function returns the length of the largest dimension of an array. For a one-dimensional array (vector), this equals the number of elements in the vector. The length of \texttt{A} in the following example is three, which is the size of the largest dimension:

\begin{verbatim}
>> A = [1 3; 2 4; 0 2];
>> length(A)
ans = 3
\end{verbatim}

\textit{reshape}

The \textit{reshape} function reshapes an array. It has the syntax

\begin{verbatim}
reshape(A, m, n, p, ...)
\end{verbatim}

where \texttt{A} is the array to be reshaped, and \texttt{m}, \texttt{n}, \texttt{p}, \ldots are the new dimensions. The number of elements in the old array must equal the number of elements in the new array. Consider the array

\begin{verbatim}
>> A = ones(2,6,2);
\end{verbatim}

Since the number of elements in \texttt{A} is \(2 \times 6 \times 2 = 24\), we should be able to reshape \texttt{A} into any order in which the product of the dimensions equals 24—for example,

\begin{verbatim}
>> reshape(A,2,12)
ans =
  1 1 1 1 1 1 1 1 1 1 1 1
  1 1 1 1 1 1 1 1 1 1 1 1
\end{verbatim}

An attempt to reshape an array into a nonconforming array results in an error. Here’s an example:
>> reshape(A,3,5)
??? Error using ==> reshape
To RESHAPE the number of elements must not change.

We shall consider another example. The following transformation makes sense, since you know that MATLAB stores arrays in column-major order:

```matlab
>> A = [ 1 2 3; 4 5 6; 7 8 9; 10 11 12]
A =
    1     2     3
    4     5     6
    7     8     9
    10    11    12
>> reshape(A, 2, 6)
ans =
    1     7     2     8     3     9
    4    10     5    11     6    12
```

**sort**

The `sort` function sorts arrays. When used on a vector, the sort is in ascending order:

```matlab
>> A = [4 2 3 9 1 2];
>> sort(A)
ans =
    1     2     2     3     4     9
```

When used on a two-dimensional array, MATLAB performs the sort on each column:

```matlab
>> A = [5 0 4; 2 2 1]
A =
    5     0     4
    2     2     1
>> sort(A)
ans =
    2     0     1
    5     2     4
```

For more than two dimensions, MATLAB performs the sort on the first dimension with the size greater than one. We call a dimension of size one a *singleton dimension*. Another way of stating this rule is that the sort is performed on the first nonsingleton dimension.

You can specify the dimension on which to sort as a second argument. For example, if we want to sort the two-dimensional array `A` across rows instead of down columns, we could use the following command:

```matlab
>> A = [5 0 4; 2 2 1]
A =
    5     0     4
    2     2     1
>> sort(A,2)
ans =
    0     4     5
    1     2     2
```

You can perform descending sorts by using the colon operator.
max, min, mean, median

The max, min, mean, and median functions each work in a similar fashion to the sort function. Given a vector argument, the functions return the maximum, minimum, mean, or median value, respectively. If given a two-dimensional array, each function returns a vector that contains the result of the operation on each column.

Because these functions each work in a similar fashion, we will demonstrate their use with the min function. First, we will use a vector as an example:

```matlab
>> A = [3 2 -6 1 10];
>> min(A)
ans =
   -6
```

Next, we will show an example that uses a two-dimensional array:

```matlab
>> A = [2 1 3; 4 2 2; 5 0 -2]
A =
    2     1     3
    4     2     2
    5     0    -2
>> min(A)
ans =
    2     0    -2
```

Note that min returns the minimum for each column.

Each of the five columns in matrix A represents the four exam grades for a student in a MATLAB programming class:

```
A = [89 97 55 72 95
     100 92 63 85 91
     82 96 71 91 82
     90 98 48 83 70 ]
```

1. Give a command that sorts each student’s grades and returns a matrix with the sorted grades.
2. Give a command that computes the mean of each student’s grades and returns a vector with the results.
3. Give a command that computes the median of each student’s grades and returns a vector with the results.
4. Give a single command that returns the overall mean grade for all five students in the course.

Now, change your view of matrix A. Assume that each of the four rows in matrix A represents the five exam grades of a student. Note: Each row represents a student.

5. Give a command that sorts each student’s grades and returns a matrix with the sorted grades.
6. Give a command that computes the mean of each student’s grades and returns a vector with the results.
7. Give a command that computes the median of each student’s grades and returns a vector with the results.
8. Give a single command that returns the overall mean grade for all five students in the course.
The calculation of the number of communication paths is important in a variety of fields, for example, the control of network router traffic. Scientists use the same theory to model behavior in fields such as human communication, political influence, and the flow of money through organizations.

A common example, used to demonstrate principles of communication routes, is the number of roads connecting cities. In this diagram we depict four cities along with the roads connecting them:

Table 2 shows the number of direct routes between each pair of cities. A direct route does not go through any intermediate city. For example, there are two direct routes between City 1 and City 4. The table expresses this information redundantly. You can see the routes between City 1 and City 4 by looking at either (row 1, column 4) or (row 4, column 1). We have presented the data in such a manner, so that it can be stored in a square matrix.

The square matrix $A$ summarizes the connectivity between the cities. For example, $A(1,4) = 2$ indicates that there are two direct routes from City 1 to City 4:

$$A =
\begin{bmatrix}
0 & 1 & 1 & 2 \\
1 & 0 & 0 & 0 \\
1 & 0 & 0 & 1 \\
2 & 0 & 1 & 0
\end{bmatrix}
$$

Note that $A$ is symmetric. This means that the cells above the main diagonal are a mirror image of the cells below the diagonal when reflected along the diagonal. Symmetry is also defined as $A(n, m) = A(m, n)$ for any $m$ and $n$.

It is known that the matrix $A^2$ represents the number of ways to travel between any two cities by passing through only one intermediate city.

$$B = A^2
\begin{bmatrix}
6 & 0 & 2 & 1 \\
0 & 1 & 1 & 2 \\
2 & 1 & 2 & 2 \\
1 & 2 & 2 & 5
\end{bmatrix}
$$

Matrix $B$ summarizes the number routes between pairs of cities if the route contains one intermediate city: Note the six ways to travel from City 1 back to City 1 by passing through exactly one other city: $a-a$, $c-c$, $c-d$, $d-c$, $d-d$, and $e-e$. The two ways to travel from City 2 to City 4 by passing through exactly one other city are $e-c$ and $e-d$.

We count traveling in one direction differently than traveling the same route in the opposite direction. Thus, there are two routes from City 1 to City 4, $c-d$ and $d-c$.

<table>
<thead>
<tr>
<th></th>
<th>City 1</th>
<th>City 2</th>
<th>City 3</th>
<th>City 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>CITY 1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>CITY 2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>CITY 3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>CITY 4</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**KEY TERMS**

- cofactor
- colon operator
- column-major order
- column vector
- command form
- compact format
- conformable
- determinant
- dot product
- functional form
- identity matrix
- inner product
- inverse
- loose format
- main diagonal
- matrix
- minor
- ones matrix
- order
- row-major order
- row vector
NEW MATLAB FUNCTIONS, COMMANDS, AND RESERVED WORDS

clock—returns current date and time
det—returns the determinant of a square matrix
diag—returns the diagonal of a matrix
etime—returns time elapsed between 2 times
eye—returns identity matrix
format—formats numeric output
get—returns the named properties of an object
inv—returns the inverse of a square matrix
length—returns the number of elements in a vector
max—returns the maximum element(s) along the first non-singleton dimension
mean—returns the mean element(s) along the first non-singleton dimension
median—returns the median element(s) along the first non-singleton dimension
min—returns the minimum element(s) along the first non-singleton dimension
ndims—returns the number of dimensions of an array
ones—returns an array of ones
rand—returns uniformly distributes pseudo-random numbers in [0,1]
reshape—reshapes an array
size—returns the order (size) of an array
sort—sorts an array in ascending order
zeros—creates an array of zeros

SOLUTIONS TO PRACTICE PROBLEMS

1. 1. A(:,2)
2. A(1:2:3,:)
3. A(:,1:2) = []
4. A = [A [7; 8; 9]]

2. 1. A+B = [6 4 6; 10 9 10]
2. A*3 = [6 0 6; 3 0 3]
3. A.*3 = [6 0 6; 3 0 3]
4. A.^3 = [8 0 8; 1 0 1]
5. (A + B)./B = [1.500 1.000 1.500;
1.1111 1.0000 1.1111]
   [ 3 Inf 3
    10 Inf 10]

3. 1. A*B = 60
2. A*C = ??? Error using ==* 
   Inner matrix dimensions must agree.
3. B*C =
   [ 8 48 0 0
   -24 -144 0 0
   8 48 0 0
   -24 -144 0 0]
4. C*B = -136
5. $A \times B = \begin{bmatrix} 24 & 144; 6 & 36 \end{bmatrix}$
6. $B \times A = \begin{bmatrix} 60 & -52; 0 & 0 \end{bmatrix}$

4. 1. $A' = \begin{bmatrix} 2 & 1; 0 & -5 \end{bmatrix}$
2. $\text{det}(A) = -10$
3. $B' = \begin{bmatrix} 3 & 4 & 0 \\
-2 & 1 & -3 \\
0 & 5 & 4 \end{bmatrix}$
4. $\text{det}(B) = 89$
5. $A^2 = \begin{bmatrix} 4 & 0; -3 & 25 \end{bmatrix}$
6. $\text{inv}(A) = \begin{bmatrix} 0.5000 & 0; 0.1000 & -0.2000 \end{bmatrix}$
7. $\text{inv}(B) = \begin{bmatrix} 0.2135 & 0.0899 & -0.1124 \\
-0.1798 & 0.1348 & -0.1685 \\
-0.1348 & 0.1011 & 0.1236 \end{bmatrix}$
8. $A^{-2} = \begin{bmatrix} 0.2500 & 0; 0.0300 & 0.0400 \end{bmatrix}$

Problems

Section 1.
What is the order and main diagonal of the following matrices?
1. $\begin{bmatrix} 3, & 4; 5, & 6; 7, & 8 \end{bmatrix}$
2. $\begin{bmatrix} 2 & 3 & 4 & 5; 6 & 7 & 8 & 9 \end{bmatrix}$
3. $\begin{bmatrix} 2 & 1 & 0; 2 & -3 & 1; 4 & 0 & 0; 3 & 2 & 1 \end{bmatrix}$

Verify your answers by using appropriate MATLAB functions.

Section 2.
4. Create a vector $A$ that contains the following fractions:
   ```
   >> A
   A =
   1/2 2/3 3/4 4/5 5/6
   ```
   What command changes your format so the vector displays rational fractions instead of decimals?
5. What command creates a $4 \times 5$ matrix that contains all zeros?

Section 3.
6. The loads in kilograms on the center points of five beams are
   400.3
   521.1
Create a row vector named “Loads” that contains the five values. What is a single command that replaces the second and fourth values of “Loads” with zero? What is a single command that deletes the third and fifth elements of “Loads”?

7. Re-create the original row vector “Loads” from the previous problem. The lengths in meters of the five beams are, respectively,

14.3
6.2
22.6
2.4
10.2

Create a row vector named “Lengths” that contains the five beam lengths in meters. In a single command, create a matrix named “Beams” by concatenating “Loads” and “Lengths”. “Beams” should have two rows with the load values on the first row and the respective lengths on the second row. Your answer should look like the following:

```
>> Beams =
400.3000 521.1000 212.1000 349.5000 322.2000
```

Section 4.

8. Assume that the loads for the five beams in Problem 6 are distributed evenly across the length of each beam. Using array arithmetic, and the original vectors “Loads” and “Lengths”, create a vector that represents the average load in kg/m for each beam.

9. The command `rand(1,n)` produces a row vector of $n$ uniformly distributed, pseudorandom numbers between 0.0 and 1.0. Use array arithmetic and the `rand` function to create 100 uniformly distributed pseudorandom numbers between 8.0 and 10.0.

Section 5.

10. Express the following linear system in matrix form as matrices $A$ and $B$:

\[
\begin{align*}
3x_1 + 2x_2 &= 4 \\
-5x_1 + 10x_2 &= 0
\end{align*}
\]

11. Use the MATLAB left matrix division operator to find the solution of the linear system in the previous problem.

Section 6.

12. The transpose of the transpose of a matrix equals the original matrix. This can be stated as $(A^T)^T = A$. Using MATLAB, demonstrate that the theorem is true for the following matrix:

\[
A = \begin{bmatrix} 1 & 2 & 4 & 6 ; & 4 & 3 & 2 & 1 \end{bmatrix}
\]
13. Experiment with the transpose operator on a few example matrices. What conclusion do you reach about the main diagonal of a matrix and the main diagonal of its transpose?

14. Given a matrix representation of a system of linear equations $AX = B$, if the determinant of $A$ equals zero, the system does not have a unique solution. Two possibilities are that the system has no solutions and that the system has an infinite number of solutions. Determine if the following system has a unique solution:

$$
2x_1 + 3x_2 + 4x_3 = 10
$$

$$
-x_1 + 3x_2 - x_3 = 12
$$

$$
\frac{3}{2}x_1 + 2x_2 + 2x_3 = 0
$$

15. The inverse of an array $A$ multiplied by itself should equal the identity matrix of the same order as $A$. Show how you would test this assumption. Use matrix multiplication and the `inv`, `eye`, and `size` functions.

16. The following matrix represents the numbers of direct paths between four network routers:

<table>
<thead>
<tr>
<th></th>
<th>R1</th>
<th>R2</th>
<th>R3</th>
<th>R4</th>
</tr>
</thead>
<tbody>
<tr>
<td>R1</td>
<td>0</td>
<td>2</td>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>R2</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>R3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>R4</td>
<td>3</td>
<td>2</td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>

How many paths are there from router two to router four if each path passes through exactly one other router?

Section 7.

17. Create a $2 \times 4 \times 3$ array of random numbers. Replace the cell contents in the third page of the array with zeros.

18. Create a three-dimensional array of order $6 \times 2 \times 3$. Fill page one with 1’s, page two with 2’s, and page 3 with 3’s. Can you solve the problem in a single command?

Section 8.

19. Create the following array:

$$
A = [1:10; 11:20; 21:30].
$$

Reshape $A$ into a two-column array. What is the bottom number in each column?

Challenge Problem

20. Reread Programming Tip 1. Test the assertion in the tip by writing a program that creates a $1 \times 20000$ row vector of ones in a single command. Write another program that creates a $1 \times 1$ row vector and then builds a $1 \times 20000$ vector of ones a single cell at a time using a loop.

Time both programs and compare the efficiency of the two methods. **Hints:** The function `clock` returns a six-element vector containing the current date and time. The meaning of each element in the vector is [year, month, day, hour, minute, seconds].
The elapsed time function, \( etime(t_2, t_1) \), returns the elapsed time in seconds between time \( t_2 \) and time \( t_1 \). The following code segment computes the time taken to execute the code between \( t_1 \) and \( t_2 \):

\[
\begin{align*}
  t_1 &= \text{clock} \\
  \ldots \\
  \ldots \\
  t_2 &= \text{clock} \\
  \text{ElapsedTime} &= \text{etime}(t_2, t_1)
\end{align*}
\]