Appendix E

MATLAB

MATLAB has become a powerful tool of technical professionals worldwide. The term *MATLAB* is an abbreviation for *MAT*rix *LAB*oratory, implying that *MATLAB* is a computational tool that uses matrices and vectors (or arrays) to carry out numerical analysis, signal processing, and scientific visualization tasks. Because *MATLAB* uses matrices as its fundamental building blocks, one can write mathematical expressions involving matrices just as easily as one would on paper. *MATLAB* is available for Macintosh, Unix, and Windows operating systems. A student version of *MATLAB* is available for personal computers (PCs). A copy of *MATLAB* can be obtained from

The Mathworks, Inc.
3 Apple Hill Drive
Natick, MA 01760-2098
Phone:(508) 647-7000
Website: http://www.mathworks.com

A brief introduction to *MATLAB* is presented in this appendix. What is presented is sufficient for solving problems in this book. More about *MATLAB* can be found in *MATLAB* books and from on-line help. The best way to learn *MATLAB* is to work with it after having learned the basics.

E.1 *MATLAB* Fundamentals

The Command window is the primary area where you interact with *MATLAB*. A little later, we will learn how to use the text editor to create M-files, which allow for execution of sequences of commands. For now, we focus on how to work in the Command window. We will first learn how to use *MATLAB* as a calculator.

Using *MATLAB* as a Calculator

The following are algebraic operators used in *MATLAB*:

+	Addition
–	Subtraction
*	Multiplication
^	Exponentiation
/	Right division (`a/b` means $a \div b$)
\	Left division (`a\b` means $b \div a$)

To begin to use *MATLAB,* we use these operators. Type commands to the *MATLAB* prompt "`>>`" in the Command window

(correct any mistakes by backspacing) and press the Enter key. For example,

```
>>  a = 2; b = 4;c = -6;
>>  dat = b^2 - 4*a*c
dat =
    64
>>  e = sqrt(dat)/10
e =
    0.8000
```

The first command assigns the values 2, 4, and −6 to the variables `a`, `b`, and `c`, respectively. *MATLAB* does not respond because this line ends with a colon. The second command sets `dat` to $b^2 - 4ac$ and *MATLAB* returns the answer as 64. Finally, the third line sets `e` equal to the square root of `dat` and divides by 10. *MATLAB* prints the answer as 0.8. Other mathematical functions, listed in Table E.1, can be used similarly to how the function `sqrt` is used here. Table E.1 provides just a tiny sample of *MATLAB* functions. Others can be obtained from the on-line help. To get help, type

```
>> help
```

A long list of topics will come up. For a specific topic, type the command name. For example, to get help on "log to base 2," type

```
>> help log2
```

A help message on the log function will be displayed. Note that *MATLAB* is case sensitive, so `sin(a)` is not the same as `sin(A)`.

TABLE E.1

Typical elementary math functions.

Function	Remark
`abs(x)`	Absolute value or complex magnitude of `x`
`acos, acosh(x)`	Inverse cosine and inverse hyperbolic cosine of `x` in radians
`acot, acoth(x)`	Inverse cotangent and inverse hyperbolic cotangent of `x` in radians
`angle(x)`	Phase angle (in radian) of a complex number `x`
`asin, asinh(x)`	Inverse sine and inverse hyperbolic sine of `x` in radians
`atan, atanh(x)`	Inverse tangent and inverse hyperbolic tangent of `x` in radians
`conj(x)`	Complex conjugate of `x`
`cos, cosh(x)`	Cosine and hyperbolic cosine of `x` in radians
`cot, coth(x)`	Cotangent and hyperbolic cotangent of `x` in radians
`exp(x)`	Exponential of `x`
`fix`	Round toward zero
`imag(x)`	Imaginary part of a complex number `x`
`log(x)`	Natural logarithm of `x`
`log2(x)`	Logarithm of `x` to base 2
`log10(x)`	Common logarithms (base 10) of `x`
`real(x)`	Real part of a complex number `x`
`sin, sinh(x)`	Sine and hyperbolic sine of `x` in radians
`sqrt(x)`	Square root of `x`
`tan, tanh`	Tangent and hyperbolic tangent of `x` in radians

Try the following examples:

```
>> 3^(log10(25.6))
>> y = 2* sin(pi/3)
>> exp(y+4-1)
```

In addition to operating on mathematical functions, *MATLAB* allows one to work easily with vectors and matrices. A vector (or array) is a special matrix with one row or one column. For example,

```
>> a = [1 -3 6 10 -8 11 14];
```

is a row vector. Defining a matrix is similar to defining a vector. For example, a 3×3 matrix can be entered as

```
>> A = [1 2 3; 4 5 6; 7 8 9]
```

or as

```
>> A = [ 1 2 3
         4 5 6
         7 8 9]
```

In addition to the arithmetic operations that can be performed on a matrix, the operations in Table E.2 can be implemented.

TABLE E.2

Matrix operations.

Operation	Remark
`A'`	Finds the transpose of matrix `A`
`det(A)`	Evaluates the determinant of matrix `A`
`inv(A)`	Calculates the inverse of matrix `A`
`eig(A)`	Determines the eigenval ues of matrix `A`
`diag(A)`	Finds the diagonal elements of matrix `A`

Using the operations in Table E.2, we can manipulate matrices as follows:

```
>> B = A'
B =
     1   4   7
     2   5   8
     3   6   9
>> C = A + B
C =
     2   6  10
     6  10  14
    10  14  18
>> D = A^3 - B*C
D =
   372   432   492
   948  1131  1314
  1524  1830  2136
>> e = [1 2; 3 4]
e =
   1  2
   3  4
>> f = det(e)
f =
  -2
>> g = inv(e)
g =
  -2.0000   1.0000
   1.5000  -0.5000
```

TABLE E.3

Special matrices, variables, and constants.

Matrix, Variable, Constant	Remark
`eye`	Identity matrix
`ones`	An array of 1s
`zeros`	An array of 0s
`i or j`	Imaginary unit or `sqrt(-1)`
`pi`	3.142
`NaN`	Not a number
`inf`	Infinity
`eps`	A very small number, `2.2e - 16`
`rand`	Random element

```
>> H = eig(g)
H =
   -2.6861
    0.1861
```

Note that not all matrices can be inverted. A matrix can be inverted if and only if its determinant is nonzero. Special matrices, variables, and constants are listed in Table E.3. For example, type

```
>> eye(3)
ans =
     1  0  0
     0  1  0
     0  0  1
```

to get a 3×3 identity matrix.

Plotting

To plot using *MATLAB* is easy. For a two-dimensional plot, use the plot command with two arguments as follows:

```
>> plot(xdata,ydata)
```

where `xdata` and `ydata` are vectors of the same length containing the data to be plotted.

For example, suppose we want to plot `y = 10*sin(2*pi*x)` from `0` to `5*pi`. We will proceed with the following commands:

```
% x is a vector, 0 <= x <= 5*pi, increments of pi/100
% creates a vector y
% creates the plot
```

```
>> x = 0:pi/100:5*pi;
>> y = 10*sin(2*pi*x);
>> plot(x,y);
```

With this, *MATLAB* responds with the plot in Fig. E.1.

MATLAB will let you graph multiple plots together and distinguish them with different colors. This is obtained with the format `plot(xdata, ydata, 'color')`, where the color is indicated by using a character string from the options listed in Table E.4.

For example,

```
>> plot (x1,y1, 'r', x2,y2, 'b', x3,y3, '--');
```

will graph data `(x1, y1)` in red, data `(x2, y2)` in blue, and data `(x3, y3)` in dashed line all on the same plot.

TABLE E.4

Various color and line types.

y	Yellow	.	Point
m	Magenta	o	Circle
c	Cyan	x	x mark
r	Red	+	Plus
g	Green	-	Solid
b	Blue	*	Star
w	White	:	Dotted
k	Black	-.	Dashdot
		--	Dashed

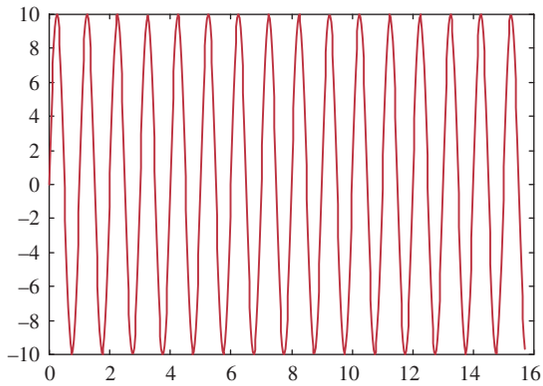

Figure E.1
MATLAB plot of `y = 10*sin(2*pi*x)`.

MATLAB also allows for logarithm scaling. Rather than using the plot command, we use

`loglog log(y)` versus `log(x)`

`semilogx y` versus `log(x)`

`semilogy log(y)` versus `x`

Three-dimensional plots are drawn using the functions mesh and meshdom (mesh domain). For example, to draw the graph of `z = x*exp( - x^2 - y^2)` over the domain `-1 < x, y < 1`, we type the following commands:

```
>> xx = -1:.1:1;
>> yy = xx;
>> [x,y] = meshgrid(xx,yy);
>> z = x.*exp(-x.^2 -y.^2);
>> mesh(z);
```

(The dot symbol used in `x.` and `y.` allows element-by-element multiplication.) The result is shown in Fig. E.2.

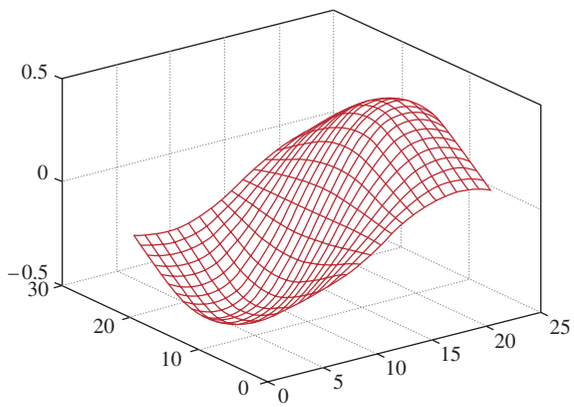

Figure E.2
A three-dimensional plot.

Programming *MATLAB*

So far we have used *MATLAB* as a calculator. You can also use *MATLAB* to create your own program. The command line editing in *MATLAB* can be inconvenient if one has several lines to execute. To avoid this problem, you can create a program that is a sequence of statements to be executed. If you are in the Command window, click **File/New/M**-files to open a new file in the *MATLAB* Editor/Debugger or simple text editor. Type the program and save it in a file with an extension `.m`, say `filename.m`; it is for this reason that it is called an M-file. Once the program is saved as an M-file, exit the Debugger window. You are now back in the Command window. Type the file without the extension `.m` to get results. For example, the plot that was made in Fig. E.2 can be improved by adding title and labels and being typed as an M-file called `example1.m`.

```
% x is a vector, 0 <= x <= 5*pi, increments of pi/100
% creates a vector y
% create the plot
% label the x axis
% label the y axis
% title the plot
% add grid
x = 0:pi/100:5*pi;
y = 10*sin(2*pi*x);
plot(x,y);
xlabel('x (in radians)');
ylabel('10*sin(2*pi*x)');
title('A sine functions');
grid
```

Once the file is saved as `example1.m` and you exit the text editor, type

```
>> example1
```

in the Command window and hit **Enter** to obtain the result shown in Fig. E.3.

To allow flow control in a program, certain relational and logical operators are necessary. They are shown in Table E.5. Perhaps the most commonly used flow control statements are `for` and `if`. The `for`

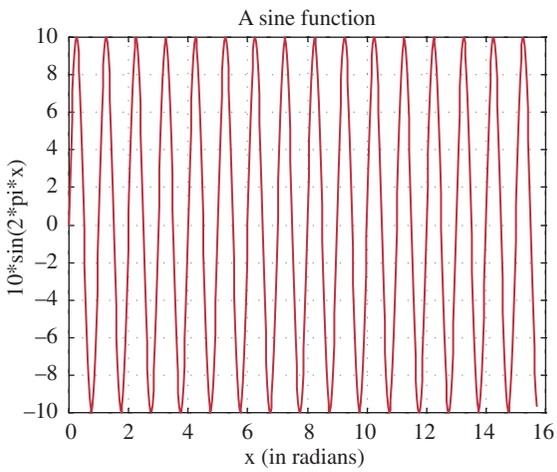

Figure E.3
MATLAB plot of `y = 10*sin(2*pi*x)` with title and labels.

TABLE E.5
Relational and logical operators.

Operator	Remark
<	less than
<=	less than or equal
>	greater than
>=	greater than or equal
==	equal
~=	not equal
&	and
|	or
~	not

statement is used to create a loop or a repetitive procedure and has the general form

```
for x = array
   [commands]
end
```

The `if` statement is used when certain conditions need to be met before an expression is executed. It has the general form

```
if expression
   [commands if expression is True]
else
   [commands if expression is False]
end
```

For example, suppose we have an array `y(x)` and we want to determine the minimum value of `y` and its corresponding index `x`. This can be done by creating an M-file as shown here.

```
% example2.m
% This program finds the minimum y value and
  its corresponding x index
x = [1 2 3 4 5 6 7 8 9 10]; %the nth term in y
y = [3 9 15 8 1 0 -2 4 12 5];
min1 = y(1); for k = 1:10
   min2 = y(k);
   if(min2 < min1)
      min1 = min2;
      xo = x(k);
   else
      min1 = min1;
   end
end
diary
min1, xo
diary off
```

Note the use of the `for` and `if` statements. When this program is saved as `example2.m`, we execute it in the Command window and obtain the minimum value of `y` as `-2` and the corresponding value of `x` as `7`, as expected.

```
>> example2
min1 =
  -2
xo =
   7
```

If we are not interested in the corresponding index, we could do the same thing using the command

```
>> min(y)
```

The following tips are helpful in working effectively with *MATLAB*:

- Comment your M-file by adding lines beginning with a % character.
- To suppress output, end each command with a semicolon (;); you may remove the semicolon when debugging the file.
- Press the up and down arrow keys to retrieve previously executed commands.
- If your expression does not fit on one line, use an ellipse (. . .) at the end of the line and continue on the next line. For example, *MATLAB* considers

```
y = sin(x + log10(2x + 3)) + cos(x + ...
log10(2x + 3));
```

  as one line of expression.
- Keep in mind that variable and function names are case sensitive.

Solving Equations

Consider the general system of n simultaneous equations:

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n &= b_2 \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= b_n \end{aligned}$$

or in matrix form

$$\mathbf{AX} = \mathbf{B}$$

where

$$\mathbf{A} = \begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \cdots & \cdots & \cdots & \cdots \\ a_{n1} & a_{n2} & a_{n3} & a_{n4} \end{bmatrix} \quad \mathbf{X} = \begin{bmatrix} x_1 \\ x_2 \\ \cdots \\ x_n \end{bmatrix} \quad \mathbf{B} = \begin{bmatrix} b_1 \\ b_2 \\ \cdots \\ b_n \end{bmatrix}$$

A is a square matrix and is known as the coefficient matrix, while **X** and **B** are vectors. **X** is the solution vector we are seeking to get. There are two ways to solve for **X** in *MATLAB*. First, we can use the back-slash operator(\) so that

```
X = A\B
```

Second, we can solve for **X** as

$$\mathbf{X} = \mathbf{A}^{-1}\mathbf{B}$$

which in *MATLAB* is the same as

```
X = inv(A)*B
```

Example E.1

Use *MATLAB* to solve Example A.2.

Solution:
From Example A.2, we obtain matrix **A** and vector **B** and enter them in *MATLAB* as follows.

```
>> A = [25 -5 -20; -5 10 -4; -5 -4 9]
A =
   25  -5  -20
   -5  10   -4
   -5  -4    9
>> B = [50 0 0]'
B =
   50
    0
    0
>> X = inv(A)*B
X =
   29.6000
   26.0000
   28.0000
>> X = A\B
X =
   29.6000
   26.0000
   28.0000
```

Thus, $x_1 = 29.6$, $x_2 = 26$, and $x_3 = 28$.

Practice Problem E.1

Solve the problem in Practice Prob. A.2 using *MATLAB*.

Answer: $x_1 = 3 = x_3$, $x_2 = 2$.

E.2 DC Circuit Analysis

There is nothing special in applying *MATLAB* to resistive dc circuits. We apply mesh and nodal analysis as usual and solve the resulting simultaneous equations using *MATLAB* as is described in Section E.1. Examples E.2 to E.5 illustrate.

Example E.2

Use nodal analysis to solve for the nodal voltages in the circuit of Fig. E.4.

Solution:
At node 1,

$$2 = \frac{V_1 - V_2}{4} + \frac{V_1 - 0}{8} \rightarrow 16 = 3V_1 - 2V_2 \qquad \textbf{(E.2.1)}$$

At node 2,

$$3I_x = \frac{V_2 - V_1}{4} + \frac{V_2 - V_3}{2} + \frac{V_2 - V_4}{2}$$

But

$$I_x = \frac{V_4 - V_3}{4}$$

so that

$$3\left(\frac{V_4 - V_3}{4}\right) = \frac{V_2 - V_1}{4} + \frac{V_2 - V_3}{2} + \frac{V_2 - V_4}{2} \rightarrow \quad \textbf{(E.2.2)}$$

$$0 = -V_1 + 5V_2 + V_3 - 5V_4$$

At node 3,

$$3 = \frac{V_3 - V_2}{2} + \frac{V_3 - V_4}{4} \rightarrow 12 = -2V_2 + 3V_3 - V_4 \quad \textbf{(E.2.3)}$$

At node 4,

$$0 = 2 + \frac{V_4 - V_2}{2} + \frac{V_4 - V_3}{4} \rightarrow -8 = -2V_2 - V_3 + 3V_4 \quad \textbf{(E.2.4)}$$

Combining Eqs. (E.2.1) to (E.2.4) gives

$$\begin{bmatrix} 3 & -2 & 0 & 0 \\ -1 & 5 & 1 & -5 \\ 0 & -2 & 3 & -1 \\ 0 & -2 & -1 & 3 \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \end{bmatrix} = \begin{bmatrix} 16 \\ 0 \\ 12 \\ -8 \end{bmatrix}$$

or

$$\mathbf{AV} = \mathbf{B}$$

We now use *MATLAB* to determine the nodal voltages contained in vector **V**.

```
>> A = [ 3 -2  0  0;
        -1  5  1 -5;
         0 -2  3 -1;
         0 -2 -1  3];
>> B = [16  0 12 -8]';
>> V = inv(A)*B
V =
    -6.0000
   -17.0000
   -13.5000
   -18.5000
```

Hence $V_1 = -6.0$, $V_2 = -17$, $V_3 = -13.5$, and $V_4 = -18.5$ V.

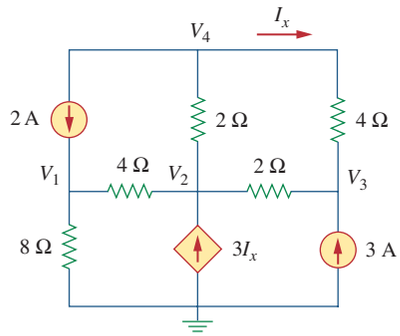

Figure E.4
For Example E.2.

Practice Problem E.2

Find the nodal voltages in the circuit in Fig. E.5 using *MATLAB*.

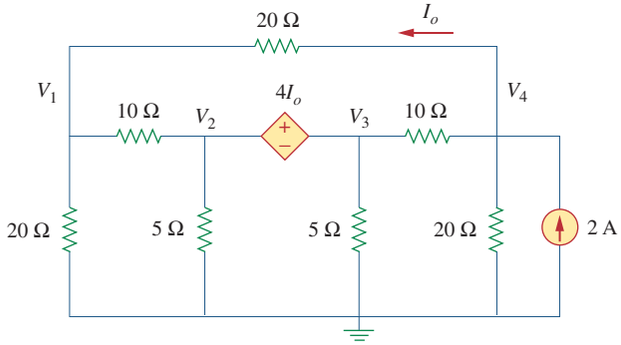

Figure E.5
For Practice Prob. E.2.

Answer:
$V_1 = 14.55$, $V_2 = 38.18$, $V_3 = -34.55$, and $V_4 = -3.636$ V.

Example E.3

Use *MATLAB* to solve for the mesh currents in the circuit in Fig. E.6.

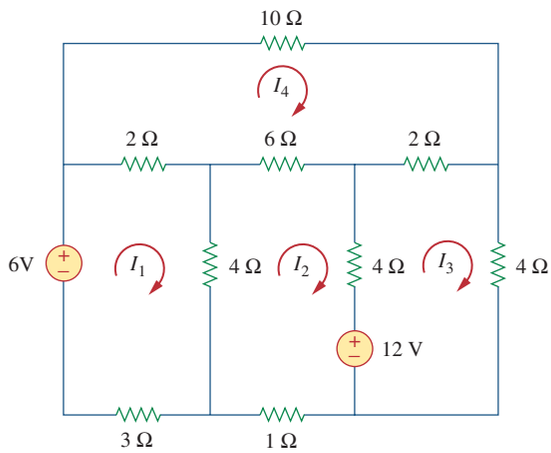

Figure E.6
For Example E.3.

Solution:
For the four meshes,

$$-6 + 9I_1 - 4I_2 - 2I_4 = 0 \longrightarrow 6 = 9I_1 - 4I_2 - 2I_4 \quad \textbf{(E.3.1)}$$

$$12 + 15I_2 - 4I_1 - 4I_3 - 6I_4 = 0 \longrightarrow$$

$$-12 = -4I_1 + 15I_2 - 4I_3 - 6I_4 \quad \textbf{(E.3.2)}$$

$$-12 + 10I_3 - 4I_2 - 2I_4 = 0 \longrightarrow 12 = -4I_2 + 10I_3 - 2I_4 \quad \textbf{(E.3.3)}$$

$$20I_4 - 2I_1 - 6I_2 - 2I_3 = 0 \longrightarrow 0 = -2I_1 - 6I_2 - 2I_3 + 20I_4 \quad \textbf{(E.3.4)}$$

Putting Eqs. (E.3.1) to (E.3.4) together in matrix form, we have

$$\begin{bmatrix} 9 & -4 & 0 & -2 \\ -4 & 15 & -4 & -6 \\ 0 & -4 & 10 & -2 \\ -2 & -6 & -2 & 20 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \\ I_4 \end{bmatrix} = \begin{bmatrix} 6 \\ -12 \\ 12 \\ 0 \end{bmatrix}$$

or $\mathbf{AI} = \mathbf{B}$, where the vector $\mathbf{I}$ contains the unknown mesh currents. We now use *MATLAB* to determine $\mathbf{I}$ as follows:

```
>> A = [9 -4  0 -2; -4 15 -4 -6;
        0 -4 10 -2;  -2 -6 -2 20]
A =
     9  -4   0  -2
    -4  15  -4  -6
     0  -4  10  -2
    -2  -6  -2  20
>> B = [6 -12 12 0]'
B =
     6
   -12
    12
     0
>> I = inv(A)*B
I =
    0.5203
   -0.3555
    1.0682
    0.0522
```

Thus, $I_1 = 0.5203, I_2 = -0.3555, I_3 = 1.0682$, and $I_4 = 0.0522$ A.

Practice Problem E.3

Find the mesh currents in the circuit in Fig. E.7 using *MATLAB*.

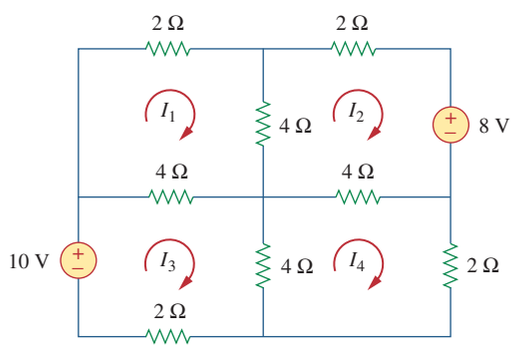

Figure E.7
For Practice Prob. E.3.

Answer: $I_1 = 0.2222, I_2 = -0.6222, I_3 = 1.1778$, and $I_4 = 0.2222$ A.

E.3 AC Circuit Analysis

Using *MATLAB* in ac circuit analysis is similar to how *MATLAB* is used for dc circuit analysis. We must first apply nodal or mesh analysis to the circuit and then use *MATLAB* to solve the resulting system of equations. However, the circuit is in the frequency domain, and we are dealing with phasors or complex numbers. So in addition to what we learned in Section E.2, we need to understand how *MATLAB* handles complex numbers.

MATLAB expresses complex numbers in the usual manner, except that the imaginary part can be either j or i representing $\sqrt{-1}$. Thus, $3 - j4$ can be written in *MATLAB* as `3 - j4`, `3 - j*4`, `3 - i4`, or `3 - I*4`. Here are the other complex functions:

`abs(A)`	Absolute value of magnitude of `A`
`angle(A)`	Angle of `A` in radians
`conj(A)`	Complex conjugate of `A`
`imag(A)`	Imaginary part of `A`
`real(A)`	Real part of `A`

Keep in mind that an angle in radians must be multiplied by $180/\pi$ to convert it to degrees, and vice versa. Also, the transpose operator (`'`) gives the complex conjugate transpose, whereas the dot-transpose (`.'`) transposes an array without conjugating it.

Example E. 4

In the circuit of Fig. E.8, let $v = 4\cos(5t - 30°)$ V and $i = 0.8\cos 5t$ A. Find v_1 and v_2.

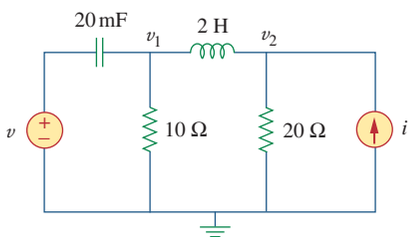

Figure E.8
For Example E.4.

Solution:

As usual, we convert the circuit in the time-domain to its frequency-domain equivalent.

$$v = 4\cos(5t - 30°) \longrightarrow \mathbf{V} = 4\angle -30°, \quad \omega = 5$$
$$i = 0.8\cos 5t \longrightarrow \mathbf{I} = 8\angle 0°$$
$$2\text{ H} \longrightarrow j\omega L = j5 \times 2 = j10$$
$$20\text{ mF} \longrightarrow \frac{1}{j\omega C} = \frac{1}{\text{j10 }\Omega \times 10^{-3}} = -j10$$

Thus, the frequency-domain equivalent circuit is shown in Fig. E.9. We now apply nodal analysis to this.

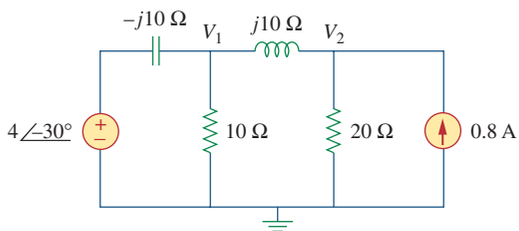

Figure E.9
The frequency-domain equivalent circuit of the circuit in Fig. E.8.

At node 1,

$$\frac{4\angle -30^\circ - V_1}{-j10} = \frac{V_1}{10} + \frac{V_1 - V_2}{j10} \longrightarrow 4\angle -30^\circ = 3.468 - j2$$
$$= -jV_1 + V_2 \quad \textbf{(E.4.1)}$$

At node 2,

$$0.8 = \frac{V_2}{20} + \frac{V_2 - V_1}{j10} \longrightarrow j16 = -2V_1 + (2 + j)V_2 \quad \textbf{(E.4.2)}$$

Equations (E.4.1) and (E.4.2) can be cast in matrix form as

$$\begin{bmatrix} -j & 1 \\ -2 & (2+j) \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} 3.468 - j2 \\ j16 \end{bmatrix}$$

or $\mathbf{AV} = \mathbf{B}$. We use *MATLAB* to invert $\mathbf{A}$ and multiply the inverse by $\mathbf{B}$ to get $\mathbf{V}$.

```
>> A = [-j 1; -2 (2+j)]
A =
   0 - 1.0000i  1.000
   -2.0000  2.0000 + 1.000 i
>> B = [(3.468 - 2j) 16j].' %note the dot-transpose
B =
   3.4680 - 2.0000i
   0 + 16.0000i
>> V = inv(A)*B
V =
   4.6055 - 2.4403i
   5.9083 + 2.6055i
>> abs(V(1))
ans =
   5.2121
>> angle(V(1))*180/pi  %converts  angle  from
   radians  to  degrees
ans  =
  -27.9175
>> abs(V(2))
ans  =
   6.4573
>> angle(V(2))*180/pi
ans  =
   23.7973
```

Thus,

$$V_1 = 4.6055 - j2.4403 = 5.212\angle -27.92^\circ$$
$$V_2 = 5.908 + j2.605 = 6.457\angle 23.8^\circ$$

In the time domain,

$$v_1 = 4.605\cos(5t - 27.92°)\text{ V}, \qquad v_2 = 6.457\cos(5t + 23.8°)\text{ V}$$

Practice Problem E.4

Calculate v_1 and v_2 in the circuit in Fig. E.10 given $i = 4\cos(10t + 40°)$ A and $v = 12\cos 10t$ V.

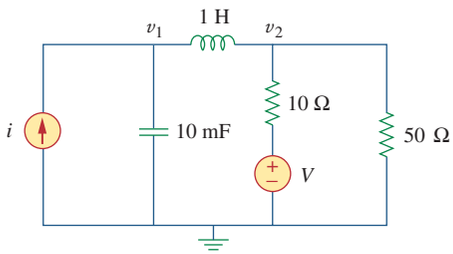

Figure E.10
For Practice Prob. E.4.

Answer: $63.58\cos(10t - 10.68°)$ V, $40\cos(10t - 50°)$ V.

Example E.5

In the unbalanced three-phase system shown in Fig. E.11, find currents I_1, I_2, I_3, and I_{Bb}. Let

$$Z_A = 12 + j10\ \Omega, \qquad Z_B = 10 - j8\ \Omega, \qquad Z_C = 15 + j6\ \Omega$$

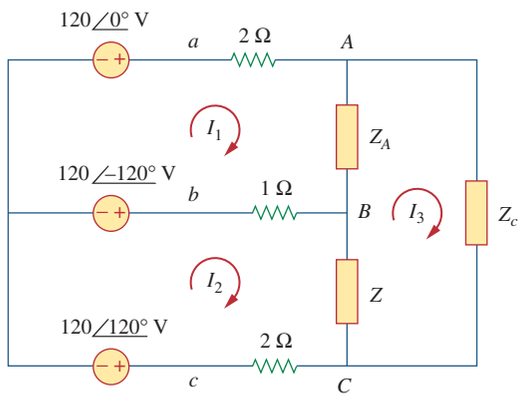

Figure E.11
For Example E.5.

Solution:
For mesh 1,

$$120\angle -120° - 120\angle 0° + I_1(2 + 1 + 12 + j10) - I_2 - I_3(12 + j10) = 0$$

or

$$I_1(15 + j10) - I_2 - I_3(12 + j10) = 120\angle 0° - 120\angle -120° \qquad \textbf{(E.5.1)}$$

For mesh 2,

$$120\angle 120^\circ - 120\angle -120^\circ + I_2(2 + 1 + 10 - j8) - I_1 - I_3(10 - j8) = 0$$

or

$$-I_1 + I_2(13 - j8) - I_3(10 - j8) = 120\angle -120^\circ - 120\angle 120^\circ \qquad \textbf{(E.5.2)}$$

For mesh 3,

$$I_3(12 + j10 + 10 - j8 + 15 + j6) - I_1(12 + j10) - I_2(10 - j8) = 0$$

or

$$-I_1(12 + j10) - I_2(10 - j8) - I_3(37 + j8) = 0 \qquad \textbf{(E.5.3)}$$

In matrix form, we can express Eqs. (E.5.1) to (E.5.3) as

$$\begin{bmatrix} 15 + j10 & -1 & -12 - j10 \\ -1 & 13 - j8 & -10 + j8 \\ -12 - j10 & -10 + j8 & 37 + j8 \end{bmatrix} \begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} 120\angle 0^\circ - 120\angle -120^\circ \\ 120\angle -120^\circ - 120\angle 120^\circ \\ 0 \end{bmatrix}$$

or

$$\mathbf{ZI} = \mathbf{V}$$

We input matrices **Z** and **V** into *MATLAB* to get I.

```
>> z = [(15 + 10j) -1 (-12 - 10j);
        -1 (13 - 8j) (-10 + 8j);
        (-12 - 10j) (-10 + 8j) (37 + 8j)];
>> c1=120*exp(j*pi*(-120)/180);
>> c2=120*exp(j*pi*(-120)/180);
>> a1=120 - c1; a2=c1 - c2;
>> V = [a1; a2; 0]
>> I = inv(z)*V
I=
   16.9910 - 6.5953i
   12.4023 - 16.9993i
    5.6621 - 6.0471i
>> IbB = I(2) - I(1)
IbB =
   -4.5887 - 10.4039i
>> abs(I(1))
ans =
    18.2261
>> angle(I(1))*180/pi
ans =
   -21.2146
```

```
>> abs (I(2))
ans =
    21.0426
>> angle(I(2))*180/pi
ans =
   -53.8864
>> abs(I(3))
ans =
    8.2841
>> angle(I(3))*180/pi
ans =
   -46.8833
>> abs(IbB)
ans =
   11.3709
>> angle(IbB)*180/pi
ans =
   -113.8001
```

Thus, $I_1 = 18.23\angle -21.21°$, $I_2 = 21.04\angle -58.89°$, $I_3 = 8.284\angle -46.88°$, and $I_{bB} = 11.37\angle -113.8°$A.

Practice Problem E.5

In the unbalanced wye-wye three-phase system in Fig. E.12, find the line currents I_1, I_2, and I_3 and the phase voltage V_{CN}.

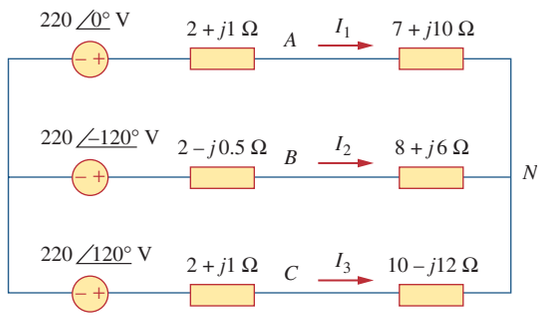

Figure E.12
For Practice Prob. E.5.

Answer: $22.66\angle -26.54°\ A$, $6.036\angle -150.48°\ A$, $19.93\angle 138.9°\ A$, $94.29\angle 159.3°\ V$.

E.4 Frequency Response

Frequency response involves plotting the magnitude and phase of the transfer function $H(s) = D(s)/N(s)$ or obtaining the Bode magnitude and phase plots of $H(s)$. One hard way to obtain the plots is to generate

data using the `for` loop for each value of $s = j\omega$ for a given range of ω and then plot the data as we did in Section E.1. However, there is an easy way that allows us to use one of two *MATLAB* commands: `freqs` and `bode`. For each command, we must first specify `H(s)` as `num` and `den`, where `num` and `den` are the vectors of coefficients of the numerator `N(s)` and denominator `D(s)` in descending powers of `s`, i.e., from the highest power to the constant term. The general form of the `bode` function is

```
bode(num, den, range);
```

where `range` is a specified frequency interval for the plot. If `range` is omitted, *MATLAB* automatically selects the frequency range. The `range` could be linear or logarithmic. For example, for $1 < \omega < 1000$ rad/s with 50 plot points, we can specify a linear `range` as

```
range = linspace(1,1000,50);
```

For a logarithmic `range` with $10^{-2} < \omega < 10^{4}$ rad/s and 100 plot points in between, we specify `range` as

```
range = logspace(-2,4,100);
```

For the `freqs` function, the general form is

```
hs = freqs(num, den, range);
```

where `hs` is the frequency response (generally complex). We still need to calculate the magnitude in decibels as

```
mag = 20*log10(abs(hs))
```

and phase in degrees as

```
phase = angle(hs)*180/pi
```

and plot them, whereas the `bode` function does it all at once. We illustrate with an example.

Example E.6

Use *MATLAB* to obtain the Bode plots of

$$G(s) = \frac{s^3}{s^3 + 14.8s^2 + 38.1s + 2554}$$

Solution:

With the explanation previously given, we develop the *MATLAB* code as shown here.

```
% for example e.6
num=[1 0 0 0];
den = [1 14.8 38.1 2554];
w = logspace(-1,3);
bode(num, den, w);
title('Bode plot for a highpass filter')
```

Running the program produces the Bode plots in Fig. E.13. It is evident from the magnitude plot that $G(s)$ represents a highpass filter.

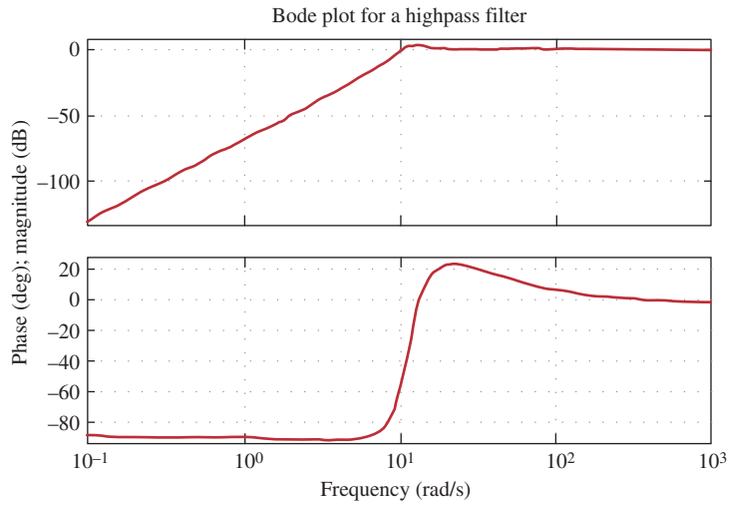

Figure E.13
For Example E.6.

Practice Problem E.6

Use *MATLAB* to determine the frequency response of

$$H(s) = \frac{10(s + 1)}{s^2 + 6s + 100}$$

Answer: See Fig. E.14.

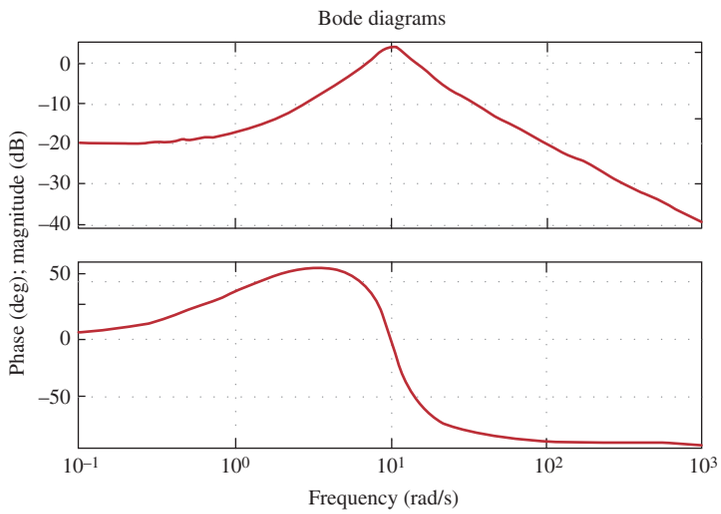

Figure E.14
For Practice Prob. E.6.