1-Analyzing Resistive Circuits Using MATLAB

The Computer program MATLAB is a tool for making mathematical calculations. In this section MATLAB is used to solve the equations encountered when analyzing a resistive circuit. Consider the resistive circuit shown in Figure 1a. The goal is to determine the value of the input voltage, $V_s$, required to cause the current $i$ to be 1 A.

![Resistor Circuit Diagram](image)

Resistors $R_1$, $R_2$, and $R_3$ are connected in series and can be replaced by an equivalent resistor, $R_s$, given by

$$R_s = R_1 + R_2 + R_3 \quad \text{(1.1)}$$

Resistors $R_4$, $R_5$, and $R_6$ are connected in parallel and can be replaced by an equivalent resistor, $R_p$, given by

$$R_p = \frac{1}{\frac{1}{R_4} + \frac{1}{R_s} + \frac{1}{R_5}} \quad \text{(1.2)}$$

Figure 3.8-1b shows the circuit after $R_1$, $R_2$, and $R_3$ are replaced by $R_s$ and $R_4$, $R_5$, and $R_6$ are replaced by $R_p$. Applying voltage division to the circuit in Figure 1-b gives

$$V_o = \frac{R_p}{R_s + R_p} V_s \quad \text{(1.3)}$$
where $V_o$ is the voltage across $R_P$ in Figure 1-1b and is also the voltage across the parallel resistors in Figure 1-1a. Ohm's law indicates that the current in $R_6$ is given by

$$I = \frac{V}{R_5}$$

(1-4)

Figure 1-2 Plot of $I$ versus $V_s$ for the circuit shown in Figure 1-1.

Figure 1-2 shows a plot of the output current $I$ versus the input voltage $V_s$. This plot shows that $I$ will be 1 A when $V_s = 14$ V.

Figure 1-3 shows the MATLAB input file that was used to obtain Figure 1-2. The MATLAB program first causes $V_s$ to vary over a range of voltages. Next, MATLAB calculates the value of $I$ corresponding to each value of $V_s$ using Eqs. 1-1 through 1-4. Finally MATLAB plots the current $I$ versus the voltage $V_s$.

```matlab
% Analyzing Resistive Circuits Using MATLAB
%--------------------------------------------------------------
% Vary the input voltage from 8 to 16 volts in 0.1 volt steps.
%--------------------------------------------------------------
Vs = 8:0.1:16;
%--------------------------------------------------------------
% Enter values of the resistances.
%--------------------------------------------------------------
R1 = 1; R2 = 2; R3 = 3; % series resistors, ohms
R4 = 6; R5 = 3; R6 = 2; % parallel resistors, ohms
```
% Find the current, I, corresponding to each value of Vs.
%--------------------------------------------------------------
Rs = R1 + R2 + R3;                     % Equation 1-1
Rp = 1 / (1/R4 + 1/R5 + 1/R6);           % Equation 1-2
for k=1:length(Vs)
    VR(k) = Vs(k) * Rp / (Rp + Rs);    % Equation 1-3
    I(k)  = VR(k) / R6;                % Equation 1-4
end
%--------------------------------------------------------------
% Plot I versus Vs.
%--------------------------------------------------------------
plot(Vs, I)
grid
xlabel('Vs, V'), ylabel('I, A')
title('Current in R6')

Figure 1-3 MATLAB input file used to obtain the plot of I versus Vs shown in Figure 1-2.

The command by command explanation of the code to create Figure 1-2:

Command 1- Analyzing Resistive Circuits

Create a vector of values for the source v

>> Vs = 8:0.1:16;
Vs =
Columns 1 through 7
  8.0000    8.1000    8.2000    ... 
Columns 8 through 14
  8.7000    8.8000    8.9000    ... 
extra lines omitted here for space

Set the individual resistance values

>> R1 = 1, R2 = 2, R3 = 3, ...
  R4 = 6, R5 = 3, R6 = 2

R2 =
  2
extra lines omitted here for space

Determine the equivalent series resistance of the branch containing the source using the equation

>> Rs = R1 + R2 + R3
Rs =6

Determine the equivalent parallel resistance of the parallel combination using the equation

>> Rp = 1 / (1/R4 + 1/R5 + 1/R6)
Rp =1

Loop through each value of Vs to get corresponding values of Vo and I using the equation

>> for k = 1:length(Vs)
    Vo(k) = Vs(k) * Rp / (Rp + Rs);
    I(k)  = Vo(k) / R6;
end;

Note that the entire output is omitted here - running this for loop without semi-colons after the individual statements would cause MATLAB to print out the Vo vector and the I vector 41 times!
Finally, create Figure 1-2 by

plotting the data
turning the grid lines on
labeling the axes
titling the figure

```matlab
>> plot(Vs, I)
>> grid
>> xlabel('Vs, V'), ylabel('I, A');
>> title('Current in R6');
```

### 2- DC Analysis using MATLAB

Circuits that contain resistors and independent or dependent sources can be analyzed by
1. Writing a set of node or mesh equations.
2. Solving those equations.

In this section, we will use the computer program MATLAB to solve the equations.

![Diagram](image)

Figure 2-1 (a) A circuit that contains a potentiometer and (b) an equivalent circuit formed by replacing the potentiometer by a model of a potentiometer ($0 \leq a \leq 1$).

Consider the circuit shown in Figure 2-1a. This circuit contains a potentiometer. In Figure 2-1b, the
The parameter $a$ varies from 0 to 1 as the wiper of the potentiometer is moved from one end of the potentiometer to the other. The resistances $R_4$ and $R_5$ are described by the equations

$$R_4 = a \times R_p$$  
(2-1)

and

$$R_5 = (1-a) \times R_p$$  

Our objective is to analyze this circuit to determine how the output voltage changes as the position of the potentiometer wiper is changed.

The circuit in Figure 2-1b can be represented by mesh equations as

$$R_1 i_1 + R_4 i_1 + R_3 (i_1 - i_2) - v_1 - 0$$

$$R_2 i_2 + R_5 i_2 - [v_2 + R_3 (i_1 - i_2)] = 0$$  

(2-3)

These mesh equations can be rearranged as

$$\begin{align*}
(R_1 + R_4 + R_3) i_1 - R_3 i_2 &= v_1 \\
-R_3 i_1 + (R_2 + R_3 + R_5) i_2 &= v_2
\end{align*}$$  
(2-4)

Substituting Eqs. 2-1 and 2-2 into Eq. 2-4 gives

$$\begin{align*}
(R_1 + aR_p + R_3) i_1 - R_3 i_2 &= v_1 \\
-R_3 i_1 + [(1-a)R_p + R_2 + R_3] i_2 &= v_2
\end{align*}$$  
(2-5)

Equation 2-5 can be written using matrices as

$$\begin{bmatrix}
R_1 + aR_p + R_3 & -R_3 \\
-R_3 & (1-a)R_p + R_2 + R_3
\end{bmatrix}
\begin{bmatrix}
i_1 \\
i_2
\end{bmatrix}
=
\begin{bmatrix}
v_1 \\
v_2
\end{bmatrix}$$  
(2-6)

Next, $i_1$ and $i_2$ are calculated by using MATLAB to solve the mesh equation, eq. 2-6. Then the output voltage is calculated as

$$v_o = R_3 (i_1 - i_2)$$  
(2-7)

Figure 2-2 shows the MATLAB input file. The parameter $a$ varies from 0 to 1 in increments of 0.05. At each value of $a$, MATLAB solves Eq. 2-6 and then uses Eq. 2-7 to calculate the output voltage. Finally, MATLAB produces the plot of $v_o$ versus $a$ that is shown in Figure 2-3.

% mesh.m solves mesh equations
%----------------------------------------------------------
% Enter values of the parameters that describe the circuit.
%----------------------------------------------------------

% circuit parameters
R1=1000;    % ohms
R2=1000;    % ohms
R3=5000;    % ohms
V1= 15;     % volts
V2=-15;     % volts

% potentiometer parameters
Rp=20e3;    % ohms
% the parameter a varies from 0 to 1 in 0.05 increments.

a=0:0.05:1; % dimensionless
for k=1:length(a)
    R = [R1+a(k)*Rp+R3 -R3; -R3 (1-a(k))*Rp+R2+R3]; % eqn.
    V = [V1; -V2]; % eqn. 2-4

    I = V'/R;
    Vo(k) = R3*(I(1)-I(2)); % eqn. 2-5
end

plot(a, Vo)
axis([0 1 -15 15])
xlabel('a, dimensionless')
ylabel('Vo, V')

Figure 2-2 MATLAB input file
used to analyze the circuit shown in Figure 2-1.

Figure 2-3 Plot of $v_o$ versus $a$ for the circuit shown in Figure 2-1.

The command by command explanation of the code to create Figure 2-3 is:

**Command 2- DC Analysis**

Enter the values of the elements in the circuit

$$\begin{align*}
R_1 &= 1000 \\
R_2 &= 1000 \\
R_3 &= 5000 \\
V_1 &= 15 \\
V_2 &= -15 \\
R_p &= 20000
\end{align*}$$

Create a vector of $a$ values from 0 to 1 in increments of 0.05

$$\begin{align*}
a &= 0:0.05:1 \\
a &= \begin{bmatrix}
0 & 0.0500 & 0.1000 & \cdots \\
0.3500 & 0.4000 & 0.4500 & \cdots \\
0.7000 & 0.7500 & 0.8000 & \cdots
\end{bmatrix}
\end{align*}$$
Note - extra columns not shown to save space.
Use a for loop to go through all the values of a. Note that the length(a) command here is used to tell
the loop to go from k=1 until k is equal to the number of elements in a.
>> for k=1:length(a)
    Set the two matrices - one for the resistance matrix and one for the voltage vector - as given in:
(2-6)
Note that commas (or spaces) go between elements of the same row and semi-colons go between rows.
\[
R = \begin{bmatrix}
R1+a(k)*Rp+R3, & -R3; \\
-R3, & (1-a(k))*Rp+R2+R3
\end{bmatrix}
\]
\[
R = 
\begin{bmatrix}
6000 & -5000 \\
-5000 & 26000
\end{bmatrix}
\]
\[
V = \begin{bmatrix}
V1; & -V2
\end{bmatrix}
\]
\[
V=15
\]
Values here shown for a=1
Solve the mesh equation for I. Since V=IR, I=V/R. The ' on V is necessary
\[
I=V'/R
\]
Store the output voltage for this value of a from
(2-7)
\[
Vo(k) = R3*(I(1)-I(2));
\]
End the for loop
end
Note that the entire output is omitted here - running this for loop without semi-colons after the individual
statements would cause MATLAB to print out the Vo vector and the I vector 21 times!
Finally, create Figure 2-3 by
plotting the data
setting the axes
labeling the x axis
labeling the y axis
>> plot(a, Vo)
>> axis([0 1 -15 15])
>> xlabel('a, dimensionless');
>> title('Vo, V');

Figure 2-3 Plot of vo versus a for the circuit shown in Figure 2-1.

3- Using MATLAB to Determine the Thévenin Equivalent Circuit

MATLAB can be used to reduce the work required to determine the Thévenin equivalent of a circuit such
as the one shown in Figure 3-1a. First, connect a resistor, R, across the terminals of the network, as
shown in Figure 3-1b. Next, write node or mesh equations to describe the circuit with the resistor
connected across its terminals. In this case, the circuit in Figure 3-1b is represented by the mesh
equations
The current $i$ in the resistor $R$ is equal to the mesh current in the third mesh, that is, $i = i_3$.

(3-2)

The mesh equations can be written using matrices such as

\[
\begin{bmatrix}
28 & -10 & -8 \\
-10 & 28 & -8 \\
-8 & -8 & 16+R
\end{bmatrix}
\begin{bmatrix}
i_1 \\
i_2 \\
i_3
\end{bmatrix} = \begin{bmatrix} 12 \\ 12 \\ 0 \end{bmatrix}
\]

(3-3)

Notice that $i = i_3$ in Figure 3-1b.

Figure 3-1 The circuit in (b) is obtained by connecting a resistor, $R$, across the terminals of the circuit in (a).

Figure 3-2a shows a MATLAB ex. that solves Eq. 3-1. Figure 3-3 illustrates the use of this MATLAB file and shows that when $R = 6$ Ohms, then $i = 0.7164$ A and that when $R = 12$ Ohms, then $i = 0.5106$ A.

% - MATLAB input file for Section 3

\[
Z = \begin{bmatrix} 28 & -10 & -8 \\ -10 & 28 & -8 \\ -8 & -8 & 16+R \end{bmatrix} \quad \text{% Mesh Equation}
\]

\[
V = \begin{bmatrix} 12; \\ 12; \\ 0 \end{bmatrix} \quad \text{% Equation 3-3}
\]
\[ Im = Z \cdot V; \] % Calculate the mesh currents.
\[ I = Im(3) \] % Equation 3-2

Figure 3-2 MATLAB file used to solve the mesh equation representing the circuit shown in Figure 3-1b.

\begin{verbatim}
EDU>> R=6
R =
6
EDU>> ch5ex
I =
0.7164
EDU>> R=12
R =
12
EDU>> ch5ex
I =
0.5106
\end{verbatim}

Figure 3-3 Computer screen showing the use of MATLAB to analyze the circuit shown in Figure 3-1.

Next, consider Figure 3-4 which shows a resistor \( R \) connected across the terminals of a Thévenin equivalent circuit.

![Thévenin equivalent circuit](image)

The circuit in Figure 3-4 is represented by the mesh equation

\[ V_t = R_t i + R i \]  \hspace{1cm} (3-4)

As a matter of notation, let \( i = i_a \) when \( R = R_a \). Similarly, let \( i = i_b \) when \( R = R_b \). Equation 5.9-4 indicates that
Equation (3-5) can be written using matrices as

\[
\begin{bmatrix}
\bar{R}_a & \bar{i}_a \\
\bar{R}_b & \bar{i}_b
\end{bmatrix}
\begin{bmatrix}
\bar{V}_t \\
\bar{V}_t
\end{bmatrix}
= 
\begin{bmatrix}
1 & -\bar{i}_a \\
1 & -\bar{i}_b
\end{bmatrix}
\begin{bmatrix}
\bar{V}_t \\
\bar{V}_t
\end{bmatrix}
\]

Given \(i, Ra, ib, \) and \(Rb, \) this matrix equation can be solved for \(Vt, \) and \(Rt, \) the parameters of the Thévenin equivalent circuit. Figure 3-5 shows a MATLAB file that solves Eq. (3-6) using the values \(i_a = 0.7164 \, \text{A}, \) 
\(Ra = 6 \, \text{Ohms}, \) \(ib = 0.5106 \, \text{A}, \) \(Rb = 12 \, \text{Ohms}. \) The resulting values of \(Vt\) and \(Rt\) are

\[
V_t = 10.664 \, \text{V} \quad \text{and} \quad R_t = 8.8863 \, \Omega
\]

% When R=Ra then i=ia:
Ra = 12; ia = 0.5106;
% When R=Rb then i=ib
Rb = 6; ib = 0.7164;
A = [1 -ia; 1 -ib]; %
% Eqn 3-6
b = [Ra*ia; Rb*ib];%
X = A\b;
Vt = X(1) % Open Circuit Voltage
Rt = X(2) % Thevenin Resistance

Figure 3-5 MATLAB file used to calculate the open-circuit voltage and the Thevenin resistance.

Click here for a command by command explanation of the code in Figure 3-2 and Figure 3-5.

### 3- Using MATLAB to Determine the Thévenin Equivalent Circuit

First, the code from Figure 3-2

The way the code is written in the book, the code from Figure 3-2 would be in a file. To run from the command line without saving a file, you must set \(R\) before typing in the coefficient matrix.

\[
\begin{bmatrix}
28 & -10 & -8 \\
-10 & 28 & -8 \\
-8 & -8 & 16 + R
\end{bmatrix}
\begin{bmatrix}
i_1 \\
i_2 \\
i_3
\end{bmatrix}
= 
\begin{bmatrix}
12 \\
-12 \\
0
\end{bmatrix}
\]

>> \(Z = [28 \, -10 \, -8];\)
Note the use of ellipsis (…) to continue a line and the use of the transpose operator (‘) on \( V \) to write a column vector as the transpose of a row vector.

\[
\begin{bmatrix}
28 & -10 & -8 \\
-10 & 28 & -8 \\
-8 & -8 & 22 \\
\end{bmatrix}
\]

4-Analysis of Op Amp Circuits Using MATLAB

Figure 4-1 shows an inverting amplifier.

![Figure 4-1 An inverting amplifier](image)

Model the operational amplifier as an ideal op amp. Then the output voltage of the inverting amplifier is related to the input voltage by

\[
V_o(t) = -\frac{R_2}{R_1} V_i(t)
\]

(4-1)

Suppose that \( R_1 = 2 \) k Ohms, \( R_2 = 50 \) k Ohms, and \( v_s = -4 \cos (2000 \pi t) \) V. Using these values in Eq. 4-1 gives \( v_o(t) = 100 \cos (2000 \pi t) \) V. This is not a practical answer. It is likely that the operational amplifier saturates, and, therefore, the ideal op amp is not an appropriate model of the operational amplifier. When voltage saturation is included in the model of the operational amplifier, the inverting amplifier is described by
where \( \text{vsat} \) denotes the saturation voltage of the operational amplifier. Equation 4-2 is a more accurate, but more complicated, model of the inverting amplifier than Eq. 4-1. Of course, we prefer the simpler model, and we use the more complicated model only when we have reason to believe that answers based on the simpler model are not accurate.

Figures 4-2 and 4-3 illustrate the use of MATLAB to analyze the inverting amplifier when the operational amplifier model includes voltage saturation. Figure 4-2 shows the MATLAB input file, and Figure 4-3 shows the resulting plot of the input and output voltages of the inverting amplifier.

\[
V_o(t) = \begin{cases} 
V_{ex} & \text{when } - \frac{R_2}{R_1} V_o(t) > \text{vsat} \\
- \frac{R_2}{R_1} V_{sat} & \text{when } - \frac{R_2}{R_1} V_o(t) < -\text{vsat} \\
\text{vsat} & \text{when } - \frac{R_2}{R_1} V_o(t) > \text{vsat} 
\end{cases} 
\]

(4-2)

% Saturate.m simulates op amp voltage saturation
% circuit parameters
R1=10e3;                  % resistance, ohms
R2=20e3;                  % resistance, ohms
R3=20e3;                  % resistance, ohms
R4=20e3;                  % resistance, ohms
R5=20e3;                  % resistance, ohms
% op amp parameter
vsat=15;                  % saturation voltage, V
% source parameters
M=3;                      % amplitude, V
f=1000;                   % frequency, Hz
w=2*pi*f;                 % frequency, rad/s
theta=(pi/180)*45;        % phase angle, rad
tf=2/f;                   % final time
N=200;                    % number of increments
t=0:tf/N:tf;              % time, s
vs = M*cos(w*t+theta);    % input voltage
for k=1:length(vs)
    Y=[ 1/R2, 1/R2 + 1/R3 + 1/R4; 0, -1/R3];% nodal admittance matrix
    J=[-vs(k)/R1 0 ]; % input current vector
    V=J/Y; % node voltage vector
    vo(k)=V(2); % record the output voltage
    if (vo(k)>vsat) vo(k)=vsat; % check for saturation
    elseif (vo(k)<-vsat) vo(k)=-vsat;
    end
end
plot(t, vo, t, vs) % plot the transfer characteristic
axis([0 tf -20 20])
xlabel('time, s')
Figure 4-2 MATLAB input file corresponding to the circuit shown in Figure 41.

![Figure 4-9 Plots the input and output voltages of the circuit shown in Figure 4-1](image)

Command by command explanation of the code used to create Figure 4-3:

First, set the circuit parameters and the op-amp saturation voltage.

```
R1 = 2e3; R2 = 50e3; R3 = 20e3;
vsat = 15;
```

Next, set the source parameters. Note that the phase angle is 180 degrees since the source starts at -4. The \( \pi/180 \) is required since MATLAB uses radians for angle measures.

```
M = 4; f = 1000; w = 2*pi*f;
theta = (pi/180) * 180;
```

Now you can create a series of times to calculate the output voltage. There are two ways to do this. The first, shown in the book, has you set the beginning, the increment size, and the end.

```
>> tf = 2/f
tf = 0.0020
>> N = 20
```
An alternative is to use MATLAB’s `linspace` command, which takes a beginning value, an end value, and a number of points.

\[
\begin{align*}
N &= 200 \\
\text{>> } t &= 0:tf/N:tf;
\end{align*}
\]

Calculate the source voltage for all times \( t \).

\[
\begin{align*}
\text{>> } vs &= M\cos(w*t+\theta);
\end{align*}
\]

Run a `for` loop that will evaluate the output voltage at all times \( t \). The resulting op-amp output voltage is given by:

\[
V_o(i) = \begin{cases} 
V_{sat} & \text{when } -\frac{R_2}{R_1}V_j(t) > V_{sat} \\
-\frac{R_2}{R_1}V_j(t) & \text{when } -V_{sat} < -\frac{R_2}{R_1}V_j(t) < V_{sat} \\
-V_{sat} & \text{when } -\frac{R_2}{R_1}V_j(t) < -V_{sat} \\
\end{cases}
\]

\((4-2)\)

\[
\begin{align*}
\text{>> for } k=1:length(vs) \\
&\text{if } (-\frac{R_2}{R_1}*vs(k) < -\text{vsat}) \\
&\quad \text{vo(k) = -vsat;} \\
&\text{else if } (-\frac{R_2}{R_1}*vs(k) > \text{vsat}) \\
&\quad \text{vo(k) = vsat;} \\
&\text{else} \\
&\quad \text{vo(k) = } -\frac{R_2}{R_1} \times \text{vs(k);} \\
&\text{end} \\
&\text{end}
\end{align*}
\]

Finally, plot the transfer characteristic as shown in Figure 43. Note the use of print codes to change the color of the two lines (‘r’ means red and ‘b’ means blue). Also, the `text` command can be used to put words on the screen. The first two arguments are the \( x \) and \( y \) coordinates of the text and the third is the actual text. The fourth and fifth are optional arguments that specify centering the text on the location given (default is to have the location given be the left-most point). The sixth and seventh are optional arguments that specify the color of the text. The vector \([R \ G \ B]\) - when the default color scheme is used - specifies the percentage of red, green, and blue to use.

\[
\begin{align*}
\text{>> plot(t, vo, 'r', t, vs, 'b')} \\
&\text{>> axis([0 tf -20 20])} \\
&\text{>> xlabel('time, s')} \\
&\text{>> ylabel('vo(t), V')} \\
&\text{>> text(1e-3,13,'Output', ...} \\
&\quad \quad \text{'HorizontalAlignment',}... 
\end{align*}
\]
5. Using MATLAB to Plot Capacitor or Inductor Voltage and Current

Suppose that the current in a 2 F capacitor is

\[ i(t) = \begin{cases} 
4 & t < 2 \\
2 + 4t & 2 < t < 6 \\
20 - 2t & 6 < t < 14 \\
-8 & t > 14 
\end{cases} \quad (5.1) \]

where the units of current are A and the units of time are s. When the initial capacitor voltage is \( v(0) \) is approximately 5 V, the capacitor voltage can be calculated using

\[ v(t) = -\frac{1}{2} \int_0^t i(t) \, dt - 5 \quad (5.2) \]

Equation 5.1 indicates that \( i(t) = 4 \) A while \( t < 2 \) s. Using this current in Eq. 5.2 gives

\[ v(t) = -\frac{1}{2} \int_0^t 4 \, dt - 5 = 2t - 5 \quad (5.3) \]

when \( t < 2 \) s. Next, Eq. 5.1 indicates that \( i(t) = t + 2 \) A while \( 2 < t < 6 \) s. Using this current in Eq. 5.2 gives

\[ v(t) = -\frac{1}{2} \left( \int_2^t (t + 2) \, dt + \int_0^2 4 \, dt \right) - 5 - \frac{1}{2} \int_2^t (t + 2) \, dt - 1 - \frac{t^2}{4} + t - 4 \quad (5.4) \]

when \( 2 < t < 6 \) s. Continuing in this way, we calculate

\[ v(t) = \frac{1}{2} \left( \int_0^t (20 - 2t) \, dt + \int_0^2 (t + 2) \, dt + \int_0^2 4 \, dt \right) - 5 \]

\[ = \frac{1}{2} \int_0^t (20 - 2t) \, dt + 11 - \frac{t^2}{2} + 10t - 31 \quad (5.5) \]
when \( 6 < t \leq 14 \) s and
\[
V(t) = \frac{1}{2} \left( \int_{14}^{t} -8 \, dt + \int_{6}^{14} (20 - 2t) \, dt + \int_{2}^{6} (t + 2) \, dt + \int_{0}^{2} 4 \, dt \right) - 5
\]
\[
= \frac{1}{2} \int_{t}^{14} -3 \, dt + 11 = 67 - 4t
\]
when \( t > 14 \) s.

Equations (5.4) through (5.7) can be summarized as

\[
V(t) = \begin{cases} 
2t - 5 & t < 2 \\
\frac{t^2}{4} + t - 4 & 2 < t < 6 \\
-\frac{t^2}{2} + 10t - 31 & 6 < t < 14 \\
67 - 4t & t > 14
\end{cases}
\]  

(5.7) Equations (5.3) and (5.8) provide an analytic representation of the capacitor current and voltage. MATLAB provides a convenient way to obtain graphical representation of these functions. Figures 5-1a and b ow MATLAB input files that represent the capacitor current and voltage. Notice that the MATLAB input file representing the current, Figure 5-1a, is very similar to Eq. 5-1, while the MATLAB input file representing the voltage, Figure 5-1b, is very similar to Eq. 5-7. Figure 5-1c shows the MATLAB input file used to plot the capacitor current and voltage. Figure 5-2 Shows the resulting plots of the capacitor current and voltage.

function i = CapCur(t)
if t<2
i=4;
elseif t<=6
i=t+2;
elseif t<14
i=20-2*t;
else
i = -8;
end
(a)
function v = CapVol(t)
if t<2
v = 2*t - 5;
elseif t<=6
v = 0.25*t*t + t - 4;
elseif t<14
v = -0.5*t*t + 10*t - 31;
else
v = 67 - 4*t;
end
(b)
t=0:1:20;
for k=1:length(t)
i(k)=CapCur(k-1);
v(k)=CapVol(k-1);
end
Figure 5-1 MATLAB input files representing (a) the capacitor current, (b) the capacitor voltage, and (c) the MATLAB input file used to plot the capacitor current and voltage.

Figure 5-2 Plots of the voltage and current of a capacitor.

Command by command explanation of the code used to create Figure 5-2.

5- Using MATLAB to Plot Capacitor or Inductor Voltage and Current

First, make sure to have the two auxiliary files, CapCur.m and CapVol.m in the current working directory. The text for the two files is shown at right. These files can be written using any text editor and should be saved as simple text files. It is important to name them exactly as shown - the name of the file is what MATLAB uses to fund the function, not the word in the file. For example, if the first line of code at the top file at right was, instead,

<table>
<thead>
<tr>
<th>CapCur.m:</th>
</tr>
</thead>
<tbody>
<tr>
<td>function i = CapCur(t)</td>
</tr>
<tr>
<td>if t&lt;2</td>
</tr>
<tr>
<td>i=4;</td>
</tr>
<tr>
<td>elseif t&lt;6</td>
</tr>
</tbody>
</table>

function i = CapacitorCurrent(t)
the function would *still* be called by the actual filename, CapCur *not* Capacitor Current.

\[
i = \begin{cases} 
  t + 2, & \text{if } t < 14 \\
  20 - 2t, & \text{if } t < 14 \\
  -8, & \text{else}
\end{cases}
\]

CapVol.m:

function \( v = \text{CapVol}(t) \)
\[
\begin{align*}
  &\text{if } t < 2 \\
  &\quad v = 2t - 5; \\
\text{elseif } t < 6 &\quad v = 0.25t^2 + t - 4; \\
\text{elseif } t < 14 &\quad v = -0.5t^2 + 10t - 31; \\
\text{else} &\quad v = 67 - 4t;
\end{align*}
\]

Create a vector of times from 0 through 20 seconds in increments of tenths of seconds. *Note: this example has some subtle differences from the code in the book.*

\[>> \text{t} = 0:0.1:20;\]

Next, run a loop to load the current and voltage values. Pay special attention to how the values are loaded: specifically - the index to the current and voltage vector must be an integer while the time may be a decimal value.

\[>> \text{for } k=1:1:\text{length(}t\text{)}; \\
\quad \text{i}(k) = \text{CapCur(}t(k)\text{);} \\
\quad \text{v}(k) = \text{CapVol(}t(k)\text{);} \\
\text{end}\]
6. Analysis of RLC Circuits Using MATLAB

The purpose of this MATLAB example is to explore the effects of varying the resistance value in the underdamped parallel RLC circuit analyzed in example 6-1 in the textbook. Consider the natural response of the parallel RLC circuit shown in Figure 6-1:

![RLC Circuit Diagram](image)

Figure 6-1 A parallel RLC circuit.

The homogeneous second order differential equation for the voltage across all three elements is given by
Depending on the element values, the circuit will be either overdamped, critically damped, or underdamped. Suppose the inductance and capacitance values are \( L = 0.1 \, \text{H} \) and \( C = 1 \, \text{mF} \) with initial values \( v_n(0) = 10 \, \text{V} \) and \( i_L(0) = -0.6 \, \text{A} \). In order for the circuit to be underdamped, the resistance value must satisfy

\[
R > \frac{1}{2} \sqrt{\frac{L}{C}}
\]  

(6-2)

or \( R > 5 \, \text{Ohms} \). When \( R = 5 \, \text{Ohms} \), the circuit is critically damped. We will therefore examine the behavior of this circuit for resistance values greater than 5 Ohms. We will now explore the solution for \( v_n(t) \) for various values of \( R \). Given a value for \( R \), the solution to the underdamped differential equation is obtained by solving for the exponential coefficient

\[
\alpha = \frac{1}{2RC}
\]

the resonant angular frequency

\[
\omega_0 = \frac{1}{\sqrt{LC}}
\]

the damped resonant angular frequency

\[
\omega_d = (\omega_0^2 - \alpha^2)^{\frac{1}{2}}
\]

With \( R = \frac{25}{3} \, \text{Ohms} \) and the \( L \) and \( C \) values given above, the solution for the voltage \( v_n(t) \) is

\[
v_n(t) = 10 e^{-60t} \cos(80t) \, \text{V}
\]

(6-3)

as derived in the textbook. Now consider holding the values of \( L \) and \( C \) constant while varying the value of \( R \). How does the solution for \( v_n(t) \) change as the value of \( R \) is varied? Figure 6-2 shows the MATLAB output for the output voltage as a function of time for several different resistances. This figure was generated using the code in Figure 6-3.
% Exploring the effects of different R on an underdamped RLC circuit

% RLC circuit
% Set component values
L=0.1;
C=0.001;
R=25/3;

% Solve for the damping coefficient, natural frequency, and damped resonance frequency
a=1/(2*R*C);
w0=1/sqrt(L*C);
wd=sqrt(w0*w0 - a*a);

% Set coefficients based on initial conditions
B1=10;
B2=(a/wd)*B1 - 10/(wd*R*C) + 0.6/(wd*C);

% Create a time base and calculate the response at those times
t=0:0.001:0.12;
\[ v = B_1\exp(-a\cdot t)\cdot\cos(wd\cdot t) + B_2\exp(-a\cdot t)\cdot\sin(wd\cdot t); \]

hold off
plot(1000\cdot t, v, 'b+-')
hold on

\% The next three segments go through the process of setting
\% a resistance, solving for the characteristics of the
\% response, calculating the voltage, and then
\% plotting the voltage
\%
R=20;
a=1/(2\cdot R \cdot C);
wd=sqrt(w0\cdot w0 - a\cdot a);
B2=(a/wd)\cdot B1 - 10/(wd\cdot R \cdot C) + 0.6/(wd\cdot C);
v=B1\cdot\exp(-a\cdot t)\cdot\cos(wd\cdot t) + B2\cdot\exp(-a\cdot t)\cdot\sin(wd\cdot t);
plot(1000\cdot t, v, 'mo-');

R=50;
a=1/(2\cdot R \cdot C);
wd=sqrt(w0\cdot w0 - a\cdot a);
B2=(a/wd)\cdot B1 - 10/(wd\cdot R \cdot C) + 0.6/(wd\cdot C);
v=B1\cdot\exp(-a\cdot t)\cdot\cos(wd\cdot t) + B2\cdot\exp(-a\cdot t)\cdot\sin(wd\cdot t);
plot(1000\cdot t, v, 'kx-');

R=100;
a=1/(2\cdot R \cdot C);
wd=sqrt(w0\cdot w0 - a\cdot a);
B2=(a/wd)\cdot B1 - 10/(wd\cdot R \cdot C) + 0.6/(wd\cdot C);
v=B1\cdot\exp(-a\cdot t)\cdot\cos(wd\cdot t) + B2\cdot\exp(-a\cdot t)\cdot\sin(wd\cdot t);
plot(1000\cdot t, v, 'rd-');

\%
\% Finally, add some information to the graph to make it
\% clearer and explain the axes
\%
legend('R=25/3','R=20','R=50','R=100')
ylabel('v_n(t), V');
xlabel('t, ms');
title('Natural Response of an Underdamped Parallel RLC Circuit');

Figure 6-3 MATLAB input file used to obtain the responses shown in Figure 9.m-2.

6-1 Analysis of RLC Circuits Using MATLAB

First, set component values

\[
\begin{align*}
L &= 0.1; \\
\Rightarrow & \quad C = 0.001; \\
\Rightarrow & \quad R = 25/3;
\end{align*}
\]

Next, calculate the damping coefficient, natural frequency, and the damped resonance frequency based on:

\[
a = \frac{1}{2\cdot R \cdot C}
\]
\[
\alpha = \frac{1}{2RC} \\
\omega_0 = \frac{1}{\sqrt{LC}} \\
\omega_d = \left( \omega_0^2 - \alpha^2 \right)^{\frac{1}{2}}
\]

Next, calculate the values of B1 and B2 based on the fact that B1 will always be 10 and B2 can be calculated as shown in the book.

\[
B1 = 10; \\
B2 = \left( \frac{a}{\omega_d} \right)B1 - \frac{10}{(\omega_d \cdot R \cdot C)} + 0.6/(\omega_d \cdot C)
\]

To start the analysis, calculate the voltage for these coefficients and plot them. Note that the *hold on* command allows you to use multiple plot commands on the same graph while the *hold off* command makes sure you create a new plot with the next graphing command. The graph right now looks like Figure 9.m-m1

\[
t = 0:0.001:0.12; \\
v = B1 \cdot \exp(-a \cdot t) \cdot \cos(\omega_d \cdot t) + B2 \cdot \exp(-a \cdot t) \cdot \sin(\omega_d \cdot t); \\
hold off \\
plot(1000 \cdot t, v, 'b+-') \\
hold on
\]

Now you can calculate three more solution sets with differing resistances. Since the *hold on* command was used, each of the curves will be placed on the same graph. Pay special attention to the plot commands used - specifically which line style refers to a particular resistance.

Also, you should note that the initial value for all four
curves is the same - 10V. This is because the initial condition specified in the problem is that the voltage at time 0 is 10V.

\[
0.6/(wd*C);
\]
\[
>> v = B1*exp(-a*t).*cos(wd*t) + B2*exp(-a*t).*sin(wd*t);
\]
\[
>> plot(1000*t,v,'mo-');
\]
\[
>> R=50;
\]
\[
>> a=1/(2*R*C);
\]
\[
>> wd=sqrt(w0*w0 - a*a);
\]
\[
>> B2=(a/wd)*B1 - 10/(wd*R*C) + 0.6/(wd*C);
\]
\[
>> v = B1*exp(-a*t).*cos(wd*t) + B2*exp(-a*t).*sin(wd*t);
\]
\[
>> plot(1000*t,v,'kx-');
\]
\[
>> R=100;
\]
\[
>> a=1/(2*R*C);
\]
\[
>> wd=sqrt(w0*w0 - a*a);
\]
\[
>> B2=(a/wd)*B1 - 10/(wd*R*C) + 0.6/(wd*C);
\]
\[
>> v = B1*exp(-a*t).*cos(wd*t) + B2*exp(-a*t).*sin(wd*t);
\]
\[
>> plot(1000*t,v,'rd-');
\]

Finally, add some labels and a legend. The legend command is used by specifying labels for each of the curves in the order they were created. The final graph is shown in Figure 9.m-2

\[
>> legend('R=25/3','R=20','R=50','R=100')
\]
\[
>> ylabel('v_n(t), V');
\]
\[
>> xlabel('t, ms');
\]
\[
>> title('Natural Response of an Underdamped Parallel RLC Circuit');
\]
7. Using MATLAB for Analysis of Steady-State Circuits with Sinusoidal Inputs

Analysis of steady-state linear circuits with sinusoidal inputs using phasors and impedances requires complex arithmetic. MATLAB can be used to reduce the effort required to do this complex arithmetic. Consider the circuit shown in Figure 7-1a. The input to this circuit, $v_i(t)$, is a sinusoidal voltage. At steady state, the output, $v_o(t)$, will also be a sinusoidal voltage as shown in Figure 10.15-1a. This circuit can be represented in the frequency domain using phasors and impedances as shown in Figure 10.15-1b. Analysis of this circuit proceeds as follows. Let $Z_1$ denote the impedance of the series combination of $R_1$ and $j\omega L$. That is,

$$Z_1 = R_1 + j\omega L \quad (7-1)$$

Next, let $Y_2$ denote the admittance of the parallel combination of $R_2$ and $1/j\omega C$. That is,

$$Y_2 = \frac{1}{R_2} + j\omega C \quad (7-2)$$

Let $Z_2$ denote the corresponding impedance, that is,

$$Z_2 = \frac{1}{Y_2} \quad (7-3)$$

Finally, $V_o$ is calculated from $V_s$ using voltage division. That is,
Figure 7-2 shows a MATLAB input file that uses equations 7-1 through 7-3 to find the steady-state response of the circuit shown in Figure 7-1. Equation 7-4 is used to calculate $V_o$. Next $B = \frac{|V_o|}{2}$ and $\phi = \angle V_o$ are calculated and used to determine the magnitude and phase angle of the sinusoidal output voltage. Notice that MATLAB, not the user, does the complex arithmetic needed to solve these equations. Finally, MATLAB produces the plot shown in Figure 7-3, which displays the sinusoidal input and output voltages in the time domain.

\[
V_o = \frac{Z_2}{Z_1 + Z_2} V_x \quad (7-4)
\]

```matlab
%-----------------------------------------------
%        Describe the input voltage source.
%-----------------------------------------------
w = 2;
A = 12;
theta = (pi/180)*60;
Vs = A*exp(j*theta)
%------------------------------------------------
% Describe the resistors, inductor and capacitor.
%------------------------------------------------
R1 = 6; 
L = 4; 
R2 = 12; 
C = 1/24; 
%------------------------------------------------
% Calculate the equivalent impedances of the
% series resistor and inductor and of the
% parallel resistor and capacitor.
%------------------------------------------------
Z1 = R1 + j*w*L         % Eqn 10.15-1
Y2 = 1/R2 + j*w*C;      % Eqn 10.15-2
Z2 = 1 / Y2             % Eqn 10.15-3
%------------------------------------------------
% Calculate the phasor corresponding to the
% output voltage.
%------------------------------------------------
Vo = Vs * Z2/(Z1 + Z2)  % Eqn 10.15-4
B = abs(Vo);
phi = angle(Vo);
%------------------------------------------------
%------------------------------------------------
T = 2*pi/w;
tf = 2*T; N = 100; dt = tf/N;
t = 0 : dt : tf;
%------------------------------------------------
% Plot the input and output voltages.
%------------------------------------------------
for k = 1 : 101
   vs(k) = A * cos(w * t(k) + theta);
   vo(k) = B * cos(w * t(k) + phi);
end
plot (t, vs, t, vo)
```
7. Using MATLAB for Analysis of Steady-State Circuits with Sinusoidal Inputs

First, set the variables associated with the voltage source

\[
\begin{align*}
& \text{\texttt{w = 2}} \\
& \text{w} = 2 \\
& \text{\texttt{A = 12}} \\
& \text{A} = 12 \\
& \text{\texttt{theta = (pi/180) * 60}} \\
& \text{theta} = 1.0472
\end{align*}
\]
\[ V_s = A \exp(j \theta) \]

\[ V_s = 6.0000 + 10.3923i \]

Next, set the variables associated with the circuit elements

\[ R_1 = 6 \]
\[ R_1 = 6 \]

\[ L = 4 \]
\[ L = 4 \]

\[ R_2 = 12 \]
\[ R_2 = 12 \]

\[ C = \frac{1}{24} \]
\[ C = 0.0417 \]

Now calculate the impedances as given in

\[ Z_1 = R_1 + j \omega L \] (7-1)

\[ Z_2 = \frac{1}{Y_2} \] (7-3)

using

\[ Y_2 = \frac{1}{R_2} + j \omega C \] (7-2)

Calculate the output phasor voltage using:

\[ V_o = \frac{Z_2}{Z_1 + Z_2} V_s \] (7-4)

then extract an amplitude (B) and an angle (phi). Note that the angles MATLAB uses are in radians

\[ V_o = 8.3308 + 0.8077i \]

\[ B = \text{abs}(V_o) \]
\[ B = 8.3698 \]

\[ \phi = \text{angle}(V_o) \]
\[ \phi = 0.0967 \]

Set up the time variable for the plot. The book suggests plotting for two periods of the input voltage using 100 points. As in the OpAmp Circuits

\[ T = 2 \pi \omega \]

\[ V_o = \frac{V_s \cdot Z_2}{Z_1 + Z_2} \]

\[ V_o = 8.3308 + 0.8077i \]

\[ B = \text{abs}(V_o) \]
\[ B = 8.3698 \]

\[ \phi = \text{angle}(V_o) \]
\[ \phi = 0.0967 \]
The code, the `linspace` command can be used to accomplish this task.

\[
T = 3.1416
\]
\[
t = \text{linspace}(0, 2*\pi, 100);
\]

Now set the `vs` and `vo` variables. The book shows how to use a `for` loop to set the variables one index at a time. MATLAB is also capable of *vectorizing*, that is, acting on an entire vector. Some of the operations MATLAB can perform on an entire array are addition and subtraction between two arrays of the same size, addition and subtraction a scalar (single value) and an array, and multiplication and division - by element - of two arrays. See MATLAB's help page for `arith` for more information.

\[
\gg \text{vs} = A \times \cos(w \times t + \theta);
\]
\[
\gg \text{vo} = B \times \cos(w \times t + \phi);
\]

Finally, make the plot shown in Figure 3. The values returned to `my_handles` are *handles* to the two lines plotted. Handles can be used to manipulate properties of graphs. The properties can be shown using the `set` command with a handle as shown. The value in {} is the current value, while the other options are possible values for that option. To set an option, use the set command with a handle, the name of the option, and the new value.

\[
\gg \text{my_handles} = \text{plot}(t, \text{vs}, 'k', t, \text{vo}, 'b')
\]
\[
\text{my_handles} =
\begin{array}{c}
7.0002 \\
9.0001
\end{array}
\]
\[
\gg \text{set}(\text{my_handles}(1))
\]
\[
\begin{align*}
\text{Color} & : \text{normal} | \text{background} | \text{xor} | \text{none} \\
\text{EraseMode} & : \begin{cases} \text{normal} | \text{background} | \text{xor} | \text{none} \\
\text{LineStyle} & : \begin{cases} \text{-} | \text{--} | \text{:} | \text{-.} | \text{none} \\
\text{LineWidth} & : \begin{cases} \text{normal} | \text{background} | \text{xor} | \text{none} \\
\text{SelectionHighlight} & : \begin{cases} \text{on} | \text{off} \\
\text{Tag} & : \text{on} | \text{off} \\
\text{UserData} & : \text{on} | \text{off} \\
\end{cases}
\end{cases}
\end{cases}
\end{align*}
\]

The direct code for getting Figure 3 is shown here. The two `set` commands are used to thicken the individual lines. Note on the second text command that a variable - `text2` - is given a value. This is a handle to that piece of text. You can use the `set` command on text, plots, labels, and the title. Use set with the handle alone to see what you can change, then use set with the handle, a property name, and a value to change that property. The `\text{it v}` items allow MATLAB to use italic font. The most recent version of MATLAB allows you to use simple LaTeX macros within labels. This includes things such as italics, bold face, fractions, subscripts, superscripts, and Greek letters.

\[
\gg \text{my_handles} = \text{plot}(t, \text{vs}, 'k', ...
\]
\[
t, \text{vo}, 'b');
\]
\[
\gg \text{set}(\text{my_handles}(1),
\begin{array}{c}
\text{'LineWidth', 3}
\end{array}
\]
\[
\gg \text{set}(\text{my_handles}(2),
\begin{array}{c}
\text{'LineWidth', 3}
\end{array}
\]
\[
\gg \text{text}(3, 11, '\text{it v}_s(t)', ...
8. Design Verification of a Parallel Resonant Circuit Using MATLAB

Consider the frequency response of the parallel resonant circuit shown in Figure 8-1.

\[ \frac{1}{\sqrt{\left(\frac{1}{R}\right)^2 + \left(\omega C - \frac{1}{\omega L}\right)^2}} \]  

(8-1)

This expression is obtained by recognizing that the frequency response is the impedance, combining the three impedances in parallel and then taking the magnitude. To verify that the design values for R, L, and C are correct, we will evaluate the magnitude of the frequency response function using these values and examine the resulting maximum value (k), quality factor (Q, ratio center frequency to bandwidth), and the center angular frequency.
Figure 13.m-2 shows a plot of the amplitude of the output voltage as a function of the input angular frequency as generated by the code in Figure 13.m-3.

Figure 8-2 Response of a parallel resonant circuit to different input frequencies.

```matlab
%--------------------------------------------------------------
% Set component values as given in the problem statement
%--------------------------------------------------------------
R=4000;
L=20e-3;
C=0.25e-6;

%--------------------------------------------------------------
% Calculate the amp. of the response and plot vs. frequency
% over a narrow range of frequencies
%--------------------------------------------------------------
w=12000:10:16000;
H=1./sqrt(1/(R^2) + (w*C - 1./(w*L)).^2);
subplot(2,2,1)
plot(w,H);
grid
xlabel('omega_0');
ylabel('Magnitude');
title('Output Mag. vs. Input Freq.);
%--------------------------------------------------------------
% Calculate the amp. of the response and plot vs. frequency
```
8-1 Design Verification of a Parallel Resonant Circuit Using MATLAB

First, set the component values

```matlab
>> R=4000;
>> L=20e-3;
>> C=0.25e-6;
```
The magnitude curve is plotted as a function of angular frequency. The BW curve is created and plotted to show the location of the half-power frequencies. From this, it is clear that the center frequency is approximately 14130 rad/s and the bandwidth is 1000 rad/s.

\[ H = \frac{1}{\sqrt{1 + \left(\frac{\omega}{L}\right)^2}} \]

MATLAB can be used to display the Bode plot or frequency response plot corresponding to a network function. As an example, consider the network function

\[ H(\omega) = \frac{K\left(1 + \frac{j\omega}{2}\right)}{\left(1 + \frac{j\omega}{P_1}\right)\left(1 + \frac{j\omega}{P_2}\right)} \]

Figure 9-1 shows a MATLAB input file that can be used to obtain the Bode plot corresponding to this network function. This MATLAB file consists of four parts.

% nf.m - plot the Bode plot of a network function
% Note - this is a corrected version from the one which appears in the Fourth Edition, First Printing
%---------------------------------------------------------------
% Create a list of logarithmically spaced frequencies.
%---------------------------------------------------------------

wmin=1; % starting frequency, rad/s
wmax=10000; % ending frequency, rad/s
w = logspace(log10(wmin),log10(wmax));

%---------------------------------------------------------------
%       Enter values of the parameters that describe the
%                        network function.
%---------------------------------------------------------------
K= 10;            % constant
z= 100;           % zero
p1=10;  p2=1000;  % poles
%---------------------------------------------------------------
% Calculate the value of the network function at each frequency.
% Calculate the magnitude and angle of the network function.
%---------------------------------------------------------------
for k=1:length(w)
    H(k) = K*(1+j*w(k)/z) / ( (1+j*w(k)/p1) * (1+j*w(k)/p2) );
    mag(k) = abs(H(k));
    phase(k) = angle(H(k));
end
%---------------------------------------------------------------
%                   Plot the Bode plot.
%---------------------------------------------------------------
subplot(2,1,1), semilogx(w/(2*pi), 20*log10(mag))
xlabel('Frequency, Hz'), ylabel('Gain, dB')
title('Bode plot')
subplot(2,1,2), semilogx(w/(2*pi), phase)
xlabel('Frequency, Hz'), ylabel('Phase, deg')

Figure 9-1 MATLAB input file used to plot the Bode plots corresponding to a network function

In the first part, the MATLAB command logspace is used to specify the frequency range for the Bode plot. The command logspace also provides a list of frequencies that are evenly spaced (on a log scale) over this frequency range.

The given network has four parameters -- the gain $K$, the zero $z$, and two pole $p_1$ and $p_2$. The second part of the MATLAB input file specifies values for these four parameters.
The third part of the MATLAB input file is for a loop that evaluates $H(\omega)$, $|H(\omega)|$, and $\angle H(\omega)$ at each frequency in the list of frequencies produced by the command `logspace`.

The fourth part of the MATLAB input file does the plotting. The command

```matlab
semilogx(w/(2*pi), 20*log10(mag))
```

does several things. The command `semilogx` indicates that the plot is to be made using a logarithmic scale for the first variable and a linear scale for the second variable. The first variable, frequency, is divided by $2\pi$ to convert to Hz. The second variable, $|H(\omega)|$, is converted to dB.

The Bode plots produced using this MATLAB input file are shown in Figure 9-2.

The second and third parts of the MATLAB input file can be modified to plot the Bode plots for a different network function.

---

**Command by command explanation of the code used to create Figure 9-2.**

### 9- Plotting Bode Plots Using MATLAB

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&gt;&gt; wmin=1;</code></td>
<td>Set up values for the angular frequency using the <code>logspace</code> command. Note that logspace will create an array with 50 entries if you do not specify a number.</td>
</tr>
<tr>
<td><code>&gt;&gt; wmax=100000;</code></td>
<td></td>
</tr>
</tbody>
</table>

---

The second and third parts of the MATLAB input file can be modified to plot the Bode plots for a different network function.
Set up network parameters, then calculate the values for the network function, the magnitude of the network function, and the phase of the network function. The code here uses MATLAB’s built-in vectorizing ability while the book uses a loop. The `.*` and `./` operators are element multiplication and division. This means that each element in the solution vector is made by multiplying or dividing corresponding elements in the first and second vector. For example, if `a=[1 2 3]` and `b=[4 5 6]`, `a.*b` will give `[4 10 18]` while `a*b` will give an error. `a*b` represents regular matrix multiplication, and for that the number of columns in the first array must equal the number of rows in the second.

Finally, make the plot as seen in Figure 9-2.

| >> w = logspace(log10(wmin), log10(wmax)); |
| >> K = 10; |
| >> z = 100; |
| >> p1 = 10; |
| >> p2 = 1000; |
| >> H = K*(1+j*w/z)./((1+j*w/p1).*(1+j*w/p2)); |
| >> mag = abs(H); |
| >> phase = angle(H); |
| >> subplot(2,1,1) |
| >> semilogx(w/(2*pi), 20*log10(mag)) |
| >> set(gca, 'TickLength',[.03 .02]) |
| >> xlabel('Frequency, Hz') |
| >> ylabel('Gain, dB') |
| >> title('Bode Plot') |
| >> axis([10^0, 10^5, -20, 20]) |
| >> subplot(2,1,2) |
| >> set(gca, 'TickLength',[.03 .02]) |
| >> semilogx(w/(2*pi), phase) |
| >> xlabel('Frequency, Hz') |
| >> ylabel('Phase, rad') |

10- Making Plots Using MATLAB
The following information is for making plots such as those seen in the MATLAB examples throughout the book and within this supplement. After reading this tutorial, you should be able to create 2-D plots including multiple traces, automatic and manually assigned axes, axis labels, titles, grids, and legends.

**Basic Input Syntax**

The basic command used in MATLAB to create a plot is `plot`. This command can take any number of arguments, as will be described below. The first part of the tutorial will explain how to use the different forms of the `plot` command.

**Single Input**

The most basic `plot` command has a single vector of numbers for an argument. For example:

```matlab
>> x=10:20;
>> y=sin(2*pi*x/10);
>> plot(y)
```

In this case, the x-axis indicates the index of the vector passed in (where *index* refers to what row or column the entry is stored) while the y axis indicates the value stored.

**Two Inputs (two vectors)**

Generally, you will want set both the x and y values. For this, you will use the two argument version of the `plot` command where the first argument stores the x values and the second argument stores the y values. Both x and y must have at least one dimension in common - rows or columns. The example below illustrates the difference between the code above and the two-argument code:

```matlab
>> x=10:20;
>> y=sin(2*pi*x/10);
>> plot(x, y)
```

In this case, the x-axis indicates the index of the vector passed in (where *index* refers to what row or column the entry is stored) while the y axis indicates the value stored.
The two main differences are that the abscissa is numbered according to the x values and that the axes automatically resize themselves to include only the values of x in the first vector.

**Two Inputs (one matrix, one vector)**

Another example of the two-argument syntax is when y is a matrix. In this case, each row or column in y is plotted against x depending upon which dimensions are the same. In the following example, both x and y have the same number of columns (200), so each line represents the data in a particular row of y:

```matlab
>> x = linspace(0, 2*pi, 100);
>> y(1, :) = cos(x);
>> y(2, :) = sin(x);
>> plot(x, y);
```

When both x and y are matrices, the results are not necessarily intuitive.

**Multiple Input**

A better way to plot data that has multiple abscissas is to use the `plot` command with several input pairs. You can give `plot` as many x-y pairs as you want and it will plot each y against its corresponding x:

```matlab
>> x1 = linspace(0,2,10);    
>> x2 = linspace(5,7,10);    
>> y1 = x1.^2;               
>> y2 = 9-x2;               
>> plot(x1, y1, x2, y2);    
```
Hold Command

Finally, multiple plots can be combined using the `hold on` and `hold off` commands. The `hold on` command instructs MATLAB to keep all traces currently on the plot and add additional traces to it. The `hold off` command instructs MATLAB to clear the plot each time a new plotting command is issued. The graph above can be produced using these commands:

```matlab
>> x1 = linspace(0,2,10);
>> x2 = linspace(5,7,10);
>> y1 = x1.^2;
>> y2 = 9-x2;
>> plot(x1, y1);
>> hold on;
>> plot(x2, y2);
>> hold off;
```

Notice that in this case both traces are the same color. This is because the multi-argument version of `plot` automatically rotates the colors of the traces while, for the `hold` command, each trace starts with the typical blue color. Note that it is good practice to use the `hold off` command to close out the `hold on` command.

Trace Options

There are three main options for each trace: color, symbol, and line style. These options are
passed as a string (i.e. surrounded by single quotes) after the data pair for the trace. For example, the trace below is of a red line with an o at every point and a solid line connecting the o's:

```matlab
>> x = linspace(0,5,15);
>> y = x.*exp(-x);
>> plot(x, y, 'ro-');
```

Trace Colors

Each trace can have a single color code associated with it. The 8 color codes are (b)lue, (g)reen, (r)ed, (c)yan, (m)agenta, (y)ellow, and (w)hite. If you use the multiple argument form of the `plot` command and do not specify colors, the different traces will be colored, in order, from blue through yellow and then start over at blue; white is not used since the background is generally white. To specify a color, simply put the color code in single quotes after the data set. Here are two pieces of code that generate the graph on the right. One uses the multiple argument form and one uses the `hold` command:

**code 1:**

```matlab
>> x = logspace(-1,0,10);
>> y1 = x.^2-x+1;
>> y2 = x.^3-x.^2+x;
>> y3 = x.^4-x.^3+x.^2;
>> plot(x, y1, 'r', x, y2, 'c', x, y3, 'y');
```

**code 2:**

```matlab
>> x = logspace(-1,0,10);
>> y1 = x.^2-x+1;
>> y2 = x.^3-x.^2+x;
>> y3 = x.^4-x.^3+x.^2;
>> plot(x, y1, 'r');
>> hold on;
>> plot(x, y2, 'c');
>> plot(x, y3, 'y');
>> hold off;
```
Trace Symbols

The current version of MATLAB has 13 different symbols that can be placed at each data point. They include a point (.), circle (o), x-mark (x), plus (+), star (*), square (s), diamond (d), pentagram (p), hexagram (h), and triangles that point in four directions (^, >, v, and <). The graph below demonstrates each of these in order:

```matlab
>> plot(1.3,'.',2.3,'o',3.3,'x',4.3,'+');
>> hold on
>> plot(0.5,2,'*',1.5,2,'s')
>> plot(2.5,2,'d',3.5,2,'p', 4.5, 2, 'h');
>> plot(1.1,'^',2.1,'>',3.1,'v',4.1,'<');
>> axis([0 5 0 4])
```

Line Styles

MATLAB also has 4 different line styles. They are solid(-), dotted (:), dashdot (-.), and dashed(--) . The graph below demonstrates each of these in order:

```matlab
>> x = [0 1];
>> y = [1 1];
>> plot(x, 4*y, '-', x, 3*y, ':');
>> hold on;
>> plot(x, 2*y, '-.', x, 1*y, '--');
>> hold off;
>> axis([0 1 0 5]);
```
Combinations

With MATLAB, you can have any combination of color, symbol, and line for each trace. Simply put all the codes for a single trace together inside single quotes. You do not have to specify every part for every data set, though it is good practice to do so for consistency. See the example below for various combinations with missing parts:

```matlab
>> x = linspace(0, 5, 50);
>> y1 = cos(2*pi*x).*exp(-x/2);
>> y2 = sin(2*pi*x).*exp(-x/2);
>> y3 = cos(2*pi*x+pi/4).*exp(-x/2);
>> plot(x,y1,'r-',x,y2,'>--',x,y3,'bs');
```

For the first trace, the absence of a symbol specification means no symbols are drawn. For the second line, the lack of a color specification makes that line default to the first color in the rotation - blue. For the third line, the lack of a line specification means that no line is drawn. If neither a symbol nor a line specification is given, a solid line will be drawn.

Labeling Commands

Now that you can control what each trace looks like, you should be able to add the finishing touches.

Axis Labels and Titles
You should begin by labeling your axes and titling your plots. You may also want to add a grid or change the axes. The labeling commands are very easy. The `xlabel` command sets the label for the x-axis, the `ylabel` command sets the label for the y-axis, and the `title` command sets the title of the plot. The `grid on` and `grid off` commands alternately place or remove a grid from the figure. Finally, the `axis` command allows you to specify either the x limits or the x and y limits of the figure. For example:

```matlab
>> TC = -40:1:100;
>> TF = (TC+40)*9/5-40;
>> plot(TC, TF, 'k-');
>> xlabel('Temperature (Celsius)');
>> ylabel('Temperature (Fahrenheit)');
>> title('Temperature Conversion Chart');
>> axis([-40 100 min(TF) max(TF)]);
```

There are several options for the xlabel, ylabel, and title commands. Most of these are beyond the scope of this work, but the two that are not are LaTeX-style character formatting and font sizing. Using LaTeX style commands, Greek letters, subscripts, superscripts, and other limited constructions. Using the 'FontSize' property, you can change the size of your text. The example below shows a figure using these commands:

```matlab
>> omega_n=1;
>> zeta = linspace(0, 1, 100);
>> omega = omega_n*sqrt(1-zeta.^2);
>> plot(zeta, omega);
>> xlabel('\zeta', 'FontSize', 24);
>> ylabel('\omega', 'FontSize', 24);
>> title('Plot of \omega=\omega_n(1-\zeta^2)^{1/2}');
```
Generally, Greek letters can be obtained by typing a slash followed by the name of the Greek letter. Upper and lower case Greek letters are available, though letters which look the same in Greek and Latin do not have special symbols (e.g. capital theta is \Theta but capital eta is H, not \Eta; neither upper nor lower case omicron has a special symbol). Super- and subscripts are created by placing a ^ or _ after the character to which the script is attached. For multiple-letter scripts, the text must be enclosed in {}.

Legends and Text

The final steps of creating a plot are putting a legend and extra text on the graph. The legend command allows you to specify titles for each of the traces on a graph. It will then produce a legend within the plot. You can move the legend in the figure simply by putting the cursor over the legend, holding the mouse button down, and dragging the legend. The text command allows you to specify a location for a string to appear and what that string will be. The gtext command allows you to simply specify a string and then use the cursor to place it. Note that the 'FontSize' property discussed above is also applicable to text and gtext. The example below shows how to create a plot with labels, a legend, and text:

```matlab
>> t = linspace(0, 4*pi, 200);
>> v1 = cos(t);
>> v2 = cos(t+2*pi/3);
>> v3 = cos(t+4*pi/3);
>> plot(t, v1, t, v2, t, v3);
>> legend('\phi=0', '\phi=2\pi/3', '\phi=4\pi/3');
>> xlabel('Time, {\bf s}');
>> ylabel('Voltage {\it v(t)}, {\bf mV}');
>> text(2, 0.5,'Note the {\it italics}');
>> text(2, 0, 'Note the big text', 'FontSize', 24);
>> text(2, -0.5, 'Note the {\bf bold}');
```
Note that the legend will remain on - even through different plot commands - until you explicitly turn it off by typing `legend` with no arguments.

### Conclusion

Given the commands above, you should be able to duplicate any plot in the book. The plotting tools in MATLAB are actually much more versatile than this tutorial shows (through the use of attributes, handles, the `get` command, and the `set` command), but the tools explained here go a long way towards making clear, sensible plots.

---

## 11- Saving and Loading Data Using MATLAB

The following information should help you load and save data using MATLAB. Specifically, this tutorial will go over saving a data set in MATLAB format, saving a data set in text format, and loading data from MATLAB or text files.

### Saving Data

The basic command used in MATLAB to save data is `save`. MATLAB can also save data by using the `fscanf` and `fwrite` functions, but these are beyond the scope of this tutorial. The save function is versatile enough for most uses, especially those in this book.

#### Saving Everything

The `save` command entered by itself saves all variables and inline functions to a file named `matlab.mat`. The `.mat` extension is used for files that are in MATLAB's proprietary format. Text editors cannot read `.mat` files properly as the data set is stored using a compression algorithm. The best part about using the `save` command and MATLAB's special format is that not only is the data set saved, the variable names are saved as well. The following code, for example, creates a vector, saves it to the `matlab.mat` file, clears all the variables out, then loads the variable back through the `load` command described below. The `whos` command is
used to list what variables are present. Note that entries in **bold** are those entered by the user:

```matlab
>> whos
>> t=1:5
   t =
   1  2  3  4  5
>> whos
   Name      Size         Bytes  Class      Attributes
   t         1x5             40  double array

Grand total is 5 elements using 40 bytes

>> save
   Saving to: matlab.mat

>> whos
   Name      Size         Bytes  Class      Attributes
   t         1x5             40  double array

Grand total is 5 elements using 40 bytes

>> clear
>> whos
>> load matlab
>> whos
   Name      Size         Bytes  Class      Attributes
   t         1x5             40  double array

Grand total is 5 elements using 40 bytes

If the `save` command is used with a name after it, then all the variables are stored in a file named `name.mat`. For example,

`save MyData`

will save all the variables in `MyData.mat`. Avoid putting periods in the file name - this will confuse MATLAB.

### Saving Parts

Sometimes you will only want to save some of the variables you have created. If this is the case, you use the `save` command followed by the name of the file you want to save and then by the names of the variables you wish to save. For example,

```matlab
>> clear;
>> t=linspace(1,2,10);
>> u=linspace(3,4,10);
>> save only_t t u
```
will save the contents of t into the file only_t.mat. The values in u remain until the MATLAB session is cleared or ended. Loading only_t will only load (or re-load) t.

Saving as Text

The three options above all save the data into files that MATLAB alone can read. Sometimes you will use MATLAB to produce data that you want to be read by another program. In these cases, you need to append the flag -ascii to your save command. There are two additional flags that may be used with the -ascii flag: -double will cause the data to be saved as 16-bit precision numbers (vs. 8 bit), and -tabs will put a tab between columns of data.

Saving as text has the same three formats as shown for saving in MATLAB's format: saving everything to a default name, saving everything to a specific file, and saving certain variables to a specific file. If you simply type save -ascii the contents of all the variables are saved, in alphabetical order, to a file called matlab. Inline functions, however, cannot be saved in text files. Furthermore, the variable names themselves do not show up in the file, merely the data. One problem this causes is if two or more of the variables you save have an unequal number of columns: the vectors are saved one after the other in rows with no spaces between rows, so the output file may have data with unequal columns. This means that MATLAB will not be able to re-read that data.

Using the save command with a filename, such as save MyTextData -ascii, will save the data into a file with the given name. No extension is added in this case. Finally, using the save command with a file name and a list of variables will save only those variables sequentially in the text file. Note again that variables of different widths can be saved, but that the data set will become invalid for MATLAB to load because of the varying width.

Loading Data

The basic command MATLAB uses to load data is load. MATLAB can also read data in using the fscanf and fread functions, but these are beyond the scope of this tutorial.

Load Everything (.mat)

The load command by itself will look for a file called matlab.mat and load all the variables from it. If a variable in the file shares the name of a variable in the session, the variable in the session is overwritten by the one in the file. There is no interactive override for this, so be careful when loading data sets.

Using the load command with a filename FNAME that has no dots in it will attempt to load a file called FNAME.mat. This version of the command will again load all the variables from the file. Because .mat files are in MATLAB's proprietary format, the variable names will be loaded with the data.

Selective Loading (.mat)
If you want to load only particular variables from a .mat file, use the `load` command followed by the file name and the names of the variables you want to load. For example:

```matlab
>> load LotsOData a b d
```

will load only variables a, b, and d from `LotsOData.mat`. If one of the names you give does not exist in the file, MATLAB will not warn you; it will simply not load anything for that variable.

### Loading Text Files

Text files can be loaded in two ways. Note that if a text file is loaded, the entire file is loaded into a single variable with a name that matches or is close to the name of the text file. The main way to load data from a file is to type `load` followed by the name of the file. If the file name has an extension other than `.mat`, MATLAB expects a text file. Typical extensions used are `.dat` and `.txt`.

The second way is by using the `-ascii` flag. The `-ascii` flag forces MATLAB to load the file as text. This is useful if you have a text file with data in it that has no extension; remember, typing `load` with a file name that does not have an extension usually makes MATLAB look for a file with the same name and a `.mat` extension.

The main difficulties with loading text files are that MATLAB loads the entire set into a single variable and that MATLAB cannot handle data sets with varying column widths. There are workarounds for this using the `fscanf` command, which is similar in use to its C counterpart.

### Conclusion

Given the commands above, you should be able to save and load data sets several different ways. Once you have the ability to make plots, scripts, and functions, you should be able to plot any data set given to you and save the results of any calculations performed at the command line, in a function, or in a script.
\[ i = 20 - 2t; \]

```
if t < 2
  v = 2*t - 5;
elseif t < 6
  v = 0.25*t*t + t - 4;
elseif t < 14

(a) function v = CapVol(t)
```
\[ v = -0.5t^2 + 10t - 31; \]

else
\[ v = 67 - 4t; \]
end

(b)
\[ t = 0:1:20; \]
for \( k = 1:1:length(t) \)
\[ i(k) = \text{CapCur}(k-1); \]
\[ v(k) = \text{CapVol}(k-1); \]
end
plot(t, i, t, v)
text(12, 10, 'v(t), V')
Figure 7.11-1 MATLAB input files representing (a) the capacitor current, (b) the capacitor voltage, and (c) the MATLAB input file used to plot the capacitor current and voltage.