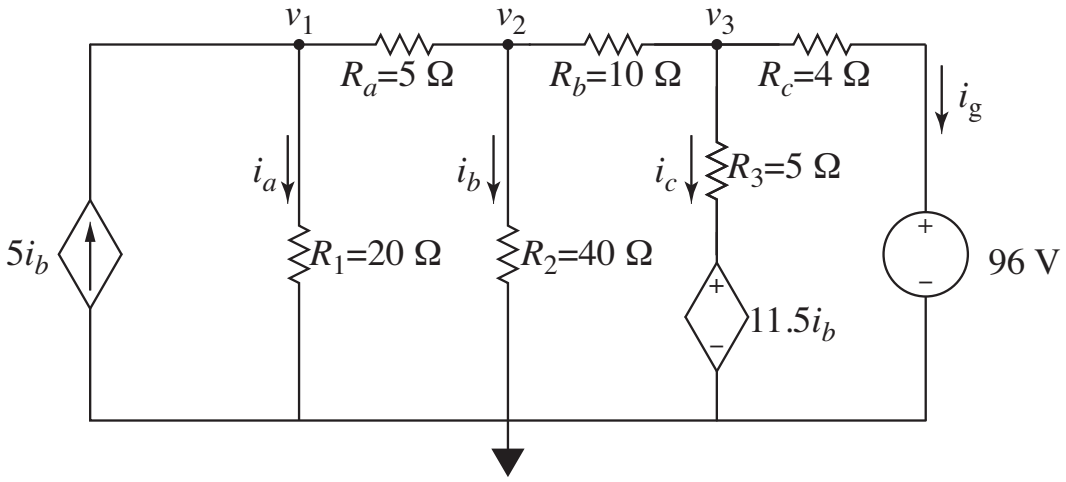
MATLAB and Systems of Equations

1 Problem Statement

Consider the following circuit where our goal is to determine the power being delivered by each of the sources. This entails finding the voltage across and the current through each device. In the case of the 96 V independent source, we will need to determine i_g; for the dependent current source, we need to determine i_b and v_1; for the dependent voltage source we need to find i_b and i_c (since the voltage across the source is given by $11.5i_b$).



Let's use the node-voltage technique to determine the voltages v_1, v_2, and v_3 and from those we can easily determine all the other circuit parameters. Solving for these voltages requires solving a 3×3 systems of equations. This can be done by hand, of course, but let's consider how *MATLAB* can aid in this.

First, in terms of the currents, we know that

$$i_b = \frac{v_2}{R_2}. \tag{1}$$

Thus, the current through the dependent current source and the voltage across the dependent voltage source (both of which depend on i_b) are

$$5i_b = 5\frac{v_2}{R_2} \quad \text{(A)}, \tag{2}$$

$$11.5i_b = 11.5\frac{v_2}{R_2} \quad \text{(V)}. \tag{3}$$

Now let's write the node-voltage equations for v_1, v_2, and v_3 where, for the moment, we will hold

File: `matlab-n-systems-of-eqs.tex`

off on using numeric quantities.

$$-5\frac{v_2}{R_2} + \frac{v_1}{R_1} + \frac{v_1 - v_2}{R_a} = 0, \tag{4}$$

$$\frac{v_2 - v_1}{R_a} + \frac{v_2}{R_2} + \frac{v_2 - v_3}{R_b} = 0, \tag{5}$$

$$\frac{v_3 - v_2}{R_b} + \frac{v_3 - 11.5\frac{v_2}{R_2}}{R_3} + \frac{v_3 - 96}{R_c} = 0. \tag{6}$$

Regrouping in terms of v_1, v_2, and v_3 yields

$$\left(\frac{1}{R_1} + \frac{1}{R_a}\right) v_1 - \left(\frac{5}{R_2} + \frac{1}{R_a}\right) v_2 = 0, \tag{7}$$

$$-\frac{1}{R_a} v_1 + \left(\frac{1}{R_2} + \frac{1}{R_a} + \frac{1}{R_b}\right) v_2 - \frac{1}{R_b} v_3 = 0, \tag{8}$$

$$-\left(\frac{11.5}{R_2 R_3} + \frac{1}{R_b}\right) v_2 + \left(\frac{1}{R_3} + \frac{1}{R_b} + \frac{1}{R_c}\right) v_3 = \frac{96}{R_c}. \tag{9}$$

If we were solving this by hand, at this point we would probably enter numeric values and try to rearrange things to give us "nice" integer coefficients. For example, considering (7), the node-voltage equation associated with v_1, we have

$$\left(\frac{1}{20} + \frac{1}{5}\right) v_1 - \left(\frac{5}{40} + \frac{1}{5}\right) v_2 = 0. \tag{10}$$

Multiplying through by 40 yields

$$(2 + 8)\, v_1 - (5 + 8)\, v_2 = 0, \tag{11}$$

which simplifies to

$$10 v_1 - 13 v_2 = 0. \tag{12}$$

We could perform similar operations for the other node-voltage equations. This is perfectly reasonable and does make the equations more manageable, but let's stick with the equations as they are expressed in (7)–(9) and think about them a bit more.

In (7) the coefficient of v_1 is the sum of two conductances (the reciprocal of resistance is conductance). With node-voltage equations we can express things in terms of conductances times voltages that are equal to currents (e.g., the right-hand side of (9) is 96 V divided by a resistance which yields a current). In general, this boils down to a system of equations that looks like this

$$g_{11} v_1 + g_{12} v_2 + g_{13} v_3 = i_1, \tag{13}$$

$$g_{21} v_1 + g_{22} v_2 + g_{23} v_3 = i_2, \tag{14}$$

$$g_{31} v_1 + g_{32} v_2 + g_{33} v_3 = i_3. \tag{15}$$

Comparing (10) and (13), we would have $g_{11} = (1/20 + 1/5) = 0.25$, $g_{12} = -(5/40 + 1/5) = -0.325$, $g_{13} = 0$, and $i_1 = 0$.

We can express these node-voltage equations in the form of a matrix equation such as

$$\begin{bmatrix} g_{11} & g_{12} & g_{13} \\ g_{21} & g_{22} & g_{23} \\ g_{31} & g_{32} & g_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} i_1 \\ i_2 \\ i_3 \end{bmatrix}, \tag{16}$$

or, symbolically,

$$\bar{\bar{G}}\bar{v} = \bar{\imath}, \tag{17}$$

where $\bar{v}$ is a column vector of unknown voltages, $\bar{\imath}$ is a column vector of known currents, and $\bar{\bar{G}}$ is square matrix of known conductances. To solve for the voltages we multiply both sides by the inverse of $\bar{\bar{G}}$, i.e.,

$$\bar{v} = \bar{\bar{G}}^{-1}\bar{\imath}. \tag{18}$$

Let us now consider how *MATLAB* can do this for us.

2 *MATLAB* Solution

In *MATLAB* the elements of a matrix or vector are enclosed in square brackets. When entering matrices, elements are entered row by row with the elements of each column separated by commas or spaces. Rows are separated by a semicolon or by a new line (i.e., the <enter> or <return> character). Assume we want to create the matrix $\bar{\bar{A}}$ that is given by

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

The following are all equivalent ways of creating this 2×3 matrix:

```
1 >> A=[1 2 3; 4 5 6]; % columns separated by spaces, rows by semicolon
2 >> A=[1,2,3; 4,5,6]; % columns separated by commas, rows by semicolon
3 >> A=[1 2 3          % columns separated by spaces, rows by newline
4       4 5 6];
5 >> A % Enter A by itself to see how MATLAB displays A.
6 A =
7      1      2      3
8      4      5      6
```

(Everything to the right of a percent sign is a comment and is ignored by MATLAB.) Although *MATLAB* allows you to use spaces to separate elements in a row, I encourage you to stick with commas as it can reduce ambiguity in certain instances.

In the listing above, note that the statements on lines 1 through 3 do not produce output but the statement on 5 does. That is not because the statements in lines 1 through 3 have an equal sign[1] in them. Instead, it is because these lines are terminated with a semicolon. Putting a semicolon at the end of a line suppresses output. If didn't have semicolons at the ends of those earlier statements, they would have produced the same output as we see in lines 6 through 8.

To pick out the element of a vector or matrix, we specify the indices enclosed in parentheses. Thus the value in the second row and third column is specified as follows:

[1] As with many program languages, in *MATLAB* the equal sign is actually serving as the "assignment operator" where the value on the right is assigned to the variable on the left.

```
>> A(2, 3)
ans =
     6
```

The elements of a row or column vector are specified with a single index. You can think of a row or column vector as a one-dimensional matrix. (Unlike in some programming languages, such as C and C++, in *MATLAB* the first element of "an array" has an index of 1.)

Assume we want to create a three-element column vector corresponding to the values on the right-hand sides of (7)–(9). This could be created with the following commands where the statement in line 1 merely sets the variable `rc` to 4.

```
1  >> rc = 4
2  rc =
3       4
4  >> currents = [0; 0; 96/rc] % Note use of semicolons.
5  currents =
6       0
7       0
8      24
```

The statement in line 4 creates the desired column vector `currents`. Despite `currents` being *written* on a single line in line 4, owing to the use of semicolons to separate the elements, each succeeding element actually correspond to a separate row (i.e., there are three rows here, which you could think of as "three lines"). As shown in line 4, we can use an expression for the elements of a matrix or vector (the value of the element in the third row is given by 96 divided by `rc`).

There are a couple other ways we could specify this column vector as shown in the following.

```
1  >> rc = 4;
2  >> % Use newlines to separate rows.
3  >> currents = [0
4  0
5  96/rc]
6  currents =
7       0
8       0
9      24
10 >> % Create a row vector and then transpose it.
11 >> currents = [0, 0, 96/rc]'
12 currents =
13      0
14      0
15     24
```

In lines 3 through 5, each value of `current` is provided on a new line. This serves to place each value in a new row, thus creating the column vector. In line 11, the values are given as a row vector. However that row vector is then transposed into a column vector. This is accomplished with the transpose operator which is simply an apostrophe (`'`) that appears at the end of the row vector.[2]

[2]The transpose operator actually performs the *conjugate* transpose, i.e., it changes a row to a column vector (or

Provided matrices are "conformal" (i.e., they have the proper dimensions for a given operation), we can add, subtract, and multiply them in the usual way. These operations are performed using the usual keyboard symbols: +, −, and ⋆. We can also calculate the inverse of a matrix using the `inv()` function. So, assuming we have a 3×3 matrix of conductances called `G` and the currents defined in a three-element column vector as shown in the previous code, we could obtain the corresponding voltages via "`inv(G)*currents`". However, the operation of inverting a matrix and multiplying it by a vector is so common in *MATLAB*, that there is a special symbol for this operation: backslash (`\`).[3] The following demonstrates this (using a $\bar{\bar{G}}$ matrix that is rather arbitrary).

```
>> G = [2, 3, 0; 3, -2, 15; 0, 3, 1]
G =
     2     3     0
     3    -2    15
     0     3     1
>> inv(G) * currents
ans =
  -10.4854
    6.9903
    3.0291
>> G \ currents
ans =
  -10.4854
    6.9903
    3.0291
```

We can, of course, store the results of such operations to a variable.[4]

Now let's return to the circuit problem we described earlier. We want to obtain the node voltages via the known conductance matrix and the known current vector. The following shows one way to obtain the voltages using *MATLAB* where we start by setting the resistances (lines 1 and 2) and then calculate all the terms of the conductance matrix using the expressions from (7)–(9) (lines 5–7).

```
1  >> r1 = 20; r2 = 40; r3 = 5;  % Define the resistances.
2  >> ra = 5; rb = 10; rc = 4;
3  >> currents = [0; 0; 96/rc];  % Define currents.
4  >> % Create the conductance matrix.
5  >> G = [1/r1 + 1/ra,       -(5/r2 + 1/ra),                0;
6                -1/ra,   1/r2 + 1/ra + 1/rb,            -1/rb;
7                    0, -(11.5/r2/r3 + 1/rb), 1/r3 + 1/rb + 1/rc]
8  G =
9      0.2500   -0.3250         0
10    -0.2000    0.3250   -0.1000
11         0   -0.1575    0.5500
```

vice versa), and it takes the complex conjugate of the values. Since we are only considering real values here, the conjugation is not relevant to us.

[3] There is a computational difference between the backslash operator and the `inv()` function. Although the results are the same, the backslash operator is more efficient so it is best to stick with that if you do not otherwise need to calculate the full inverse.

[4] Although we are only working with real numbers here, were we working in the frequency domain and had complex numbers (phasors and impedance), *MATLAB* handles those perfectly well.

```
>> voltages = G \ currents  % Solve for voltages.
voltages =
  156.0000
  120.0000
   78.0000
```

In general, you will find that when working with problems such as this, your life will be much easier if you work in the "Live Editor" and clicking "Run" when desired, rather than working directly in the "Command Window." Note that all the results that are generated via the "Live Editor" are subsequently available in the "Command Window." So, do the bulk of your entry in the "Live Editor" and then perform whatever addition calculations may be necessary in the "Command Window."

At this point, *MATLAB* has informed us, in lines 13 through 16 in the previous listing, that the voltages are $v_1 = 156$ V, $v_2 = 120$ V, and $v_3 = 78$ V. These voltages are stored in *MATLAB* as `voltages(1)`, `voltages(2)`, nad `voltages(3)`, respectively.

However, our interest was not in these voltages. Instead, we were asked for the power associated with each of the sources. We now have all the information we need to obtain these values. For the independent source, i_g is $(v_3 - 96)/R_c$ and the power is $96i_g$. For the dependent current source, the current is $5i_b = 5v_2/R_2$ so the power is given by $-v_1 5v_2/R_2$. The current associated with the dependent voltage source is $i_c = (v_3 - 11.5v_2/R_2)/R_3$ while the voltage itself is $11.5v_2/R_2$. Thus, the power for this source is $(v_3 - 11.5v_2/R_2)11.5v_2/(R_2R_3)$. We can have *MATLAB* perform these calculations as follows:

```
>> % Power of 96 V voltage source.
>> p96 = 96 * (voltages(3) - 96) / rc
p96 =
  -432
>> % Power of dependent current source.
>> p5i = -voltages(1) * 5 * voltages(2) / r2
p5i =
  -2.3400e+03
>> % Power of dependent voltage source.
>> p115i = (voltages(3) - 11.5 * voltages(2) / r2) / r3 ...
           * 11.5 * voltages(2) / r2
p115i =
  300.1500
```

From line 4 we see that the independent voltage source is delivering 432 W. Line 8 shows the dependent current source is delivering $2,340$ W (2.34 kW). And line 13 shows that the dependent voltage source is consuming 300.15 W.

Notice that line 7 is terminated with "`...`" This is the way to indicate line continuation in *MATLAB*. So, the complete expression is the combination of lines 7 and 8.