

Working with Phasors and Using Complex Polar Notation in MATLAB/Octave

Tony Richardson
University of Evansville

By default, MATLAB/Octave accepts complex numbers only in rectangular form. Use **i** or **j** to represent the imaginary number $\sqrt{-1}$.

```
> 5+4i
ans = 5 + 4i
```

A number in polar form, such as $(2\angle 45^\circ)$, can be entered using complex exponential notation. The angle must be converted to radians when entering numbers in complex exponential form:

```
>> x = 2*exp(j*45*pi/180)
x = 1.4142 + 1.4142i
```

You can save yourself some typing by defining the following functions:

```
>> d2r = @(x) (x*pi/180);
>> r2d = @(x) (x*180/pi);
>> cis = @(x) exp(j*x);
>> cisd = @(x) cis(d2r(x));
>> angled = @(x) r2d(angle(x));
```

The first two functions convert degrees to radians (**d2r**) and radians to degrees (**r2d**). The **cis** functions are complex exponential functions ($\text{cis}(x) = e^{jx} = \cos(x) + j\sin(x)$). The argument to the **cis** function is in radians, while the argument to the **cisd** function is in degrees. The **angled** function returns the polar angle of a complex number in degrees.

The previous expression can now be entered as:

```
>> x = 2*cisd(45)
x = 1.4142 + 1.4142i
```

The **abs** and **angled** functions can be used to find the polar form components of a complex number:

```
> [abs(x)    angled(x)]
ans = 2.0000e+00    45.0000e+00
```

Here are some additional examples:

```
> z = (7+8i+20*cisd(-30))/(5*cisd(45))
z = 3.1566e+00 - 3.7223e+00i
> [abs(z) angled(z)]
ans = 4.8805e+00    -49.7011e+00
> volts = [10*cisd(45); 20*cisd(-30); 100]
volts =
    7.0711e+00 + 7.0711e+00i
   17.3205e+00 - 10.0000e+00i
  100.0000e+00 + 0.0000e+00i
```

The first two results indicate that z is equal to $3.1566 - 3.7223i$ or $(4.8805 \angle -49.7011^\circ)$. The last example illustrates how a voltage column array can be defined. An alternative method for entering a complex array in polar form separates the magnitude and phase vectors:

```
> volts = [10; 20; 100] .* cisd([45; -30; 0])
```

```

volts =
    7.0711e+00 + 7.0711e+00i
    17.3205e+00 - 10.0000e+00i
    100.0000e+00 + 0.0000e+00i

```

Note that the element-by-element multiplication operator “.*” is used instead of the ordinary (matrix) multiplication operator “*”. Also note that the magnitude and phase vectors must be of the same dimension.

We can easily recover the polar components of the **volts** array using:

```

> [abs(volts) angled(volts)]
ans =
    10.0000e+00    45.0000e+00
    20.0000e+00   -30.0000e+00
    100.0000e+00    0.0000e+00

```

Here's a final example that illustrates how to use MATLAB/Octave to solve a sinusoidal steady-state circuit problem. Suppose we want to find the branch phasor currents in the phasor domain circuit shown in Figure 1.

We will use mesh analysis to solve this problem. We can write the following set of mesh equations by inspection:

$$\begin{aligned}
 100 \angle -90 &= j12(I_A - I_C) + 80(I_A - I_B) \\
 -500 \angle 0 &= 80(I_B - I_A) + 20(I_B - I_C) \\
 0 &= -j20I_C + 20(I_C - I_B) + j12(I_C - I_A)
 \end{aligned}$$

Instead of simplifying these equations by hand, let's go directly to MATLAB/Octave. We can define the voltage array as:

```

> V = [100*cisd(-90); -500*cisd(0); 0];

```

The elements of Z are formed directly from the mesh equations above (let MATLAB/Octave do the work of combining the terms for you):

```

> Z = [12i+80, -80, -12i
      -80,    80+20, -20
      -12i,  -20,   -20i+20+12i];

```

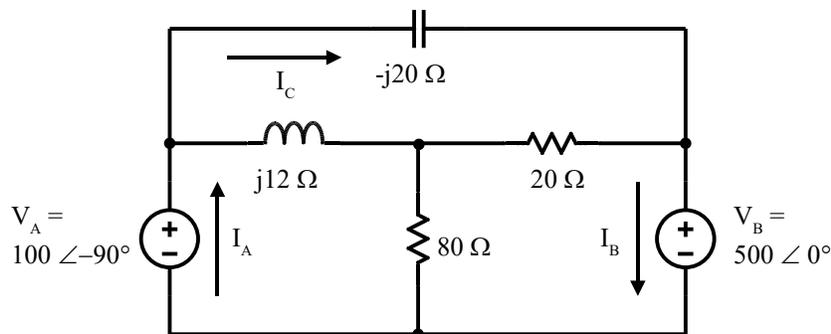


Figure 1: Example Frequency Domain Circuit

Since $V = Z I$, then $I = Z^{-1} V$. Using MATLAB/Octave to compute the currents gives:

```

> I = Z \ V;
> [abs(I) angled(I)]

```

```
ans =
    22.0227e+00   -129.4725e+00
    24.0208e+00   -129.2558e+00
    25.4951e+00   -78.6901e+00
```

The currents are therefore equal to $I_A = (22.0 \angle -129.5^\circ)$, $I_B = (24.0 \angle -129.3^\circ)$, and $I_C = (25.5 \angle -78.7^\circ)$. Note: Although the current vector I could be computed using $I = \text{inv}(Z)*V$, it is more accurate (and faster) to do left division using the left division operator `\` as shown.

Addendum

Two additional functions, **to_rd** (polar to rectangular with angle in degrees) and **to_pd** (rectangular to polar with angle in degrees) can be defined to make it even easier to enter and display numbers in polar form. Define the functions as follows:

```
> to_rd = @(m, d) m .* cisd(d);
> to_pd = @(x) [abs(x) angled(x)];
```

To enter a complex number in polar form ($2 \angle 45^\circ$) and convert it to the standard rectangular form use the **to_rd** function:

```
> x = to_rd(2,45)
x = 1.4142 + 1.4142i
```

To display the polar form components of a complex number use the **to_pd** function:

```
> to_pd(x)
ans = 2.0    45.0
```

Note that this returns a hybrid vector with the magnitude as the first element and the phase angle as the second element.

You can use the **to_rd** function to enter a complex number in polar form at any point you might use a number in rectangular form. Here are some additional examples:

```
> z = (7+8*i + to_rd(20, -30))/to_rd(5, 45)
z = 3.1566 - 3.7223i
> to_pd(z)
ans = 4.8805   -49.7011
> volts = [to_rd(10, 45); to_rd(20, -30); to_rd(100, 0)]
volts =
    7.0711 + 7.0711i
   17.3205 - 10.0000i
  100.0000 + 0.0000i
```

The first two results indicate that z is equal to $3.1566 - 3.72228i$ or $(4.8805 \angle -49.701^\circ)$. The last example illustrates how a voltage column array can be defined.

The function **to_rd** will also accept matrix arguments. The first argument should contain the magnitudes and the second argument should contain the phase angles (in degrees), the **volts** array in the previous example could be defined (with less typing) as:

```
> volts = to_rd([10 20 100]', [45 -30 0]')

volts =
    7.0711 + 7.0711i
   17.3205 - 10.0000i
  100.0000 + 0.0000i
```

The function **to_pd** will also accept an array of complex numbers:

```
> to_pd(volts)
ans =
    10.00000    45.00000
    20.00000   -30.00000
    100.00000    0.00000
```

Repeating the circuit analysis using these functions, we can define the voltage array as:

```
> V = [to_rd(100, -90); -to_rd(500,0); to_rd(0, 0)];
```

The **to_rd** function was used to allow us to easily enter in the voltage array using magnitude and phase angle values.

The Z matrix is:

```
> Z = [12i+80,    -80,    -12i
>       -80, 80+20,    -20
>       -12i,   -20, -20i+20+12i];
```

Although not necessary here, the **to_rd** function could have been used to allow us to easily enter in any values that were in polar form.

Finally solving for the currents and displaying the result in polar form yields:

```
> I = Z\V;
> to_pd(I)
    22.023   -129.472
    24.021   -129.256
    25.495    -78.690
```

The currents are therefore equal to $I_A = (22.0 \angle -129.5^\circ)$, $I_B = (24.0 \angle -129.3^\circ)$, and $I_C = (25.5 \angle -78.7^\circ)$ as before.

Note: When using the **to_pd** function with a complex matrix argument, the magnitudes will be shown in the columns that make up the first half of the result while the phase angles are shown in the second half:

```
> Z
Z =
    80.0 + 12.0i    -80.0    -12.0i
   -80.0           100.0    -20.0
   -12.0i          -20.0     20.0 - 8.0i

> to_pd(Z)
ans =
    80.8950    80.0000    12.0000     8.5308    180.0000   -90.0000
    80.0000   100.0000    20.0000    180.0000     0.0000   180.0000
    12.0000    20.0000    21.5407   -90.0000    180.0000   -21.8014
```

The (1, 1) element of the Z matrix is $80 + 12i$ or $(80.8950 \angle 8.5308^\circ)$. (The magnitude is in the (1, 1) position of the result returned from **to_pd**, while the phase angle is in the (1, 4) position.)

MATLAB/Octave Functions for Polar/Rectangular Conversion

The functions used in this tutorial can be defined directly on the command line as shown. Unfortunately these definitions can be erased by the **clear** function. You may want to define them in function files as follows:

```
function r = d2r(a_d)
% -- d2r(ang)
%   Converts argument from degrees to radians
  r = a_d*pi/180;
end

function r = r2d(a_r)
% -- r2d(ang)
%   Converts argument from radians to degrees
  r = a_r*180/pi;
end

function r = cis(a_r)
% -- cis(ang)
%   Returns exp(j*ang)
  r = exp(j*a_r);
end

function r = cisd(a_d)
% -- cis(ang)
%   Returns exp(j*d2r(ang))
  r = exp(j*d2r(a_d));
end

function r = angled(c)
% -- angled(c)
%   Returns the polar angle (in degrees) of a complex number
  r = r2d(angle(c));
end

function r = to_rd(m,d)
% -- to_rd(m, d)
%   Converts arrays containing the mag (m) and phase (d, in degrees) of
%   a complex number to standard complex form
  r = m .* cisd(d);
end

function r = to_pd(c)
% -- to_pd(c)
%   Returns an array containing the mag and phase (in degrees) of
%   a complex number.
  r = [abs(c) angled(c)];
end
```