

## Working with Complex Numbers and Matrices in Scilab

Tony Richardson  
University of Evansville

By default, *Scilab* accepts complex numbers only in rectangular form. Use %i to represent the complex number  $i$ .

```
-->5+4*%i
ans =

    5. + 4.i
```

Two additional functions `to_r()` and `to_p()` make it easy to enter and display numbers in polar form. The code for these functions is included on the last page of this tutorial. You should also be able to find the code in a file on the same web site where you found this document. If you add the function definitions into a file named `.scilab` in your *Scilab* start-up directory, they will be automatically available whenever you start *Scilab*. Alternatively, you can put the function definitions into a file (named `polar.sci` for example) and then use the command `exec("polar.sci")` to load the definitions and make them available for use.

To enter the complex number ( $1\angle 45^\circ$ ) using the `to_r()` function, just enter:

```
-->x = to_r(1,45)
x =

    0.7071068 + 0.7071068i
```

To convert a complex number to polar form use the `to_p()` function:

```
-->p = to_p(x)
p =

!   1.   45. !
```

Note that the magnitude and phase angle are returned as two elements in a matrix and that the phase angle is in degrees.

You can use the `to_r()` 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_r(20, -30))/to_r(5, 45)
z =

    3.1565965 - 3.7222819i

-->p = to_p(z)
p =

!   4.8805209  - 49.701147 !

-->volts = [to_r(10,45); to_r(20,-30); to_r(100,0)]
volts =

!   7.0710678 + 7.0710678i !
!   17.320508 -10.i       !
!   100.                !
```

The first two results indicate that  $z$  is equal to  $3.1566 + 3.72228i$  or  $(4.8805 \angle -49.70^\circ)$ . The last example illustrates how a voltage column array can be defined. `to_r()` will also accept a matrix of complex numbers in polar form with the magnitude and phase as adjacent elements in the matrix, the `volts` array in the previous example could be defined as:

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

! 7.0710678 + 7.0710678i !
! 17.320508 -10.i      !
! 100.              !
```

`to_p()` will accept an array of complex numbers. It returns an array of magnitude and phase pairs:

```
-->p = to_p(volts)
p =

! 10.    45. !
! 20.   -30. !
! 100.    0. !
```

It should be noted that `to_r()` and `to_p()` are inverse functions, the output from one can be used as input to the other to obtain the original number or array.

There is a built-in *Scilab* operator that is useful when working with complex arrays. The apostrophe (') can be used to form the complex-conjugate transpose of a complex array. If you want the transpose of an array be sure to use the dot-apostrophe (.) operator. For a real array the apostrophe and dot-apostrophe operators give identical results. When applied to a single complex number (instead of an array) the apostrophe operator gives the complex-conjugate of the number.

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

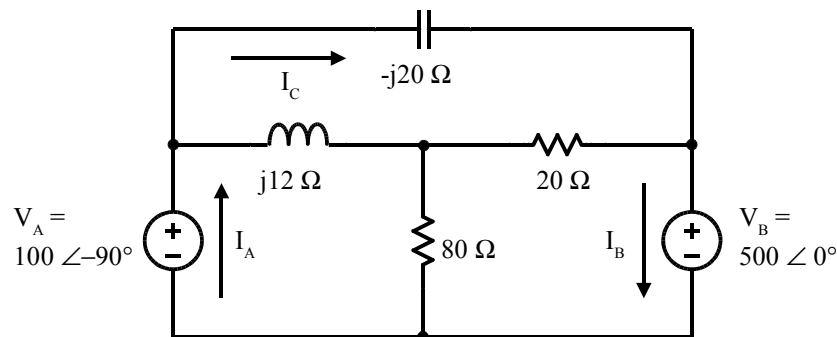


Figure 1: Example Frequency Domain Circuit

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

$$\begin{aligned} V_A &= j12(I_A - I_C) + 80(I_A - I_B) \\ -V_B &= 80(I_B - I_A) + 20(I_B - I_C) \\ 0 &= -j20I_C + 20(I_C - I_B) + j12(I_C - I_A) \end{aligned}$$

Instead of simplifying these equations, let's go directly to *Scilab*. We can define a voltage array as:

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

The `t_o_r()` function is used to allow us to easily enter in the voltage array using magnitude and phase angle pairs.

Now I want to enter in the impedance matrix  $Z$ . The elements of  $Z$  are formed directly from the mesh equations above (let *Scilab* do the work of combining the terms for you):

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

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

Since  $V = Z I$ , then  $I = Z^{-1} V$ . Although *Scilab* can easily do a matrix inverse, the left division operator (`\`) gives a more accurate result:

```
-->I = Z\V;
-->Ip = t_o_p(I)
Ip =

!   22.022716   - 129.47246 !
!   24.020824   - 129.25584 !
!   25.495098   - 78.690068 !
```

Combining the results from the magnitude and phase arrays we have,  $I_A = (22.0 \angle -129.5^\circ)$ ,  $I_B = (24.0 \angle -129.3^\circ)$ , and  $I_C = (25.5 \angle -78.7^\circ)$ . The corresponding time functions are:

$$\begin{aligned}i_A &= 22.0 \cos(\omega t - 129.5^\circ) \\i_B &= 24.0 \cos(\omega t - 129.3^\circ) \\i_C &= 25.5 \cos(\omega t - 78.7^\circ)\end{aligned}$$

## Scilab Complex Number Routines

*Note: The versions of these routines that are available for download have more comments, better error checking, and support additional options. Only the core code is shown here to limit the size of this document.*

```
// to_r() - Convert from polar to rectangular form.
function [c] = to_r(m,d)
    nc = size(m,2);
    if argn(2) == 1 then
        // If there is a single argument, the mag and phase are
        // in adjacent columns.
        c = zeros(size(m,1),nc/2);
        c = m(:,1:2:$).*cos(%pi*m(:,2:2:$)/180.) + ...
            %i*m(:,1:2:$).*sin(%pi*m(:,2:2:$)/180.);
    elseif argn(2) == 2 then
        // With two arguments, the mag and phase are in separate matrices
        c = zeros(m);
        c = m.*cos(%pi*d/180.) + %i*m.*sin(%pi*d/180.);
    end
endfunction
```

```
// to_p() - Convert from rectangular to polar form.
function [m, d] = to_p(c)
    if argn(1) == 1 then
        // If there is a single return argument, return the mag and phase
        // in adjacent columns.
        m = zeros(size(c,1),2*size(c,2));
        d = zeros(c);
        m(:,1:2:$) = abs(c);
        // Protect against 0 entries
        i = (c <> 0);
        d(i) = atan(imag(c(i)),real(c(i)))*180./%pi;
        m(:,2:2:$) = d;
    elseif argn(1) == 2 then
        // With two return arguments, the mag and phase are in stored in
        // separate matrices
        // Protect against 0 entries
        i = (c <> 0);
        d = zeros(c);
        d(i) = atan(imag(c(i)),real(c(i)))*180./%pi;
        m = abs(c);
    else
        error("invalid number of return arguments");
    end
endfunction
```