ENME202 Matlab

COMPLEX VARIABLES, FUNCTIONS, and OPERATIONS

Matlab understands complex values:

```
sqrt(-4)    % ans = 0 + 2.0000i
```

```
ans = 0.0000 + 2.0000i
```

The result is shown in the standard "cartesian" or "rectangular" form for a complex number: a+b*i, where a,b are real, and i is the "imaginary" number whose square is -1.

```
i     % ans = 0 + 1.0000i
```

```
ans = 0.0000 + 1.0000i
```

```
i^2   % ans = -1
```

```
ans = -1
```

Matlab also understands complex number calculations like:

```
i^i
```

```
ans = 0.2079
```

```
sin(i)
```

```
ans = 0.0000 + 1.1752i
```

```
exp(i*pi)
```

```
ans = -1.0000 + 0.0000i
```

Engineers often use "j" instead of "i" for the "imaginary" number j^2 = -1. While Matlab will always use i to display complex values, the definition for j is also built in.

```
j     % ans = 0 + 1.0000i
```

```
ans = 0.0000 + 1.0000i
```

```
j^2   % ans = -1
```

```
ans = -1
```

Let's look at the quadratic formula again, but with coefficients that yield complex roots:

```
a = 2; b = 4; c = 10;
x1 = (-b+sqrt(b^2-4*a*c))/(2*a)     % x1 = -1.0000 + 2.0000i
```

```
x1 = -1.0000 + 2.0000i
```

```
x2 = (-b-sqrt(b^2-4*a*c))/(2*a)     % x2 = -1.0000 - 2.0000i
```

```
x2 = -1.0000 - 2.0000i
```

We can also directly define complex variables like:

```
z1 = -1 + 2*j
```

```
z1 = -1.0000 + 2.0000i
```

Note: You don't need "*" before i or j when explicitly defining a complex number (although you can, if you want to). This is the one (and only) situation where Matlab will understand implied multiplication without explicitly using "*".

```
z2 = -2 + 3.5j
```

```
z2 = -2.0000 + 3.5000i
```

However, you can only use the "implicit multiply" with i or j if the thing that precedes it is a real number (not just something that *evaluates* to a real number)

```
(2+3)*i   % This is fine, but
```

```
ans = 0.0000 + 5.0000i
```

```
% (2+3)i    % this is not
```

You also can't use the "implied multiply" shorthand with variables:

```
c = 3;
c*i       % This is fine, but
```

```
ans = 0.0000 + 3.0000i
```

```
% ci        % this is not
```

The latter is not allowed since Matlab can't know if ci is meant to be a separate variable name.

Complex functions and operations

The real() and imag() functions can be used to break a complex number into its real and imaginary parts:

```
real(z1)    % Real part of a complex number
```

```
ans = -1
```

```
imag(z1)    % Imaginary part of a complex number
```

```
ans = 2
```

```
conj(z1)    % Complex conjugate
```

```
ans = -1.0000 - 2.0000i
```

Matlab knows how to do complex arithmeticd:

```
z1 + z2
```

```
ans = -3.0000 + 5.5000i
```

```
z1 - z2
```

```
ans = 1.0000 - 1.5000i
```

```
z1 * z2
```

```
ans = -5.0000 - 7.5000i
```

```
z3 = z1 / z2
```

```
z3 = 0.5538 - 0.0308i
```

Just as an exercise/check, let's do the above division for z3 manually using complex functions, without using complex arithmetic.

To divide two complex numbers, multiply top & bottom by the complex conjugate of the denominator (z2):

```
z2c = conj(z2)                          % complex conjugate of z2
```

```
z2c = -2.0000 - 3.5000i
```

```
denom = real(z2)^2 + imag(z2)^2     % Denominator (always real)
```

```
denom = 16.2500
```

```
numReal = real(z1)*real(z2c) - imag(z1)*imag(z2c)   % numerator (real)
```

```
numReal = 9
```

```
numImag = real(z1)*imag(z2c) + imag(z1)*real(z2c)   % numerator (imag)
```

```
numImag = -0.5000
```

```
z3 = numReal/denom + j*numImag/denom    % z3 = z1/z2
```

```
z3 = 0.5538 - 0.0308i
```

The result should be the same as typing z3 = z1/z2

Complex magnitude (absolute value)

```
abs(z1)
```

```
ans = 2.2361
```

The following statements are equivalent:

```
abs(z1)
```

```
ans = 2.2361
```

```
sqrt(z1*conj(z1))
```

```
ans = 2.2361
```

```
sqrt(real(z1)^2 + imag(z1)^2)
```

```
ans = 2.2361
```

Complex exponential function -- Euler's formula

```
theta = 3*pi/4      % pick an angle
```

```
theta = 2.3562
```

```
exp(j*theta)        % A*exp(j*theta) = A*(cos(theta) + j*sin(theta))
```

```
ans = -0.7071 + 0.7071i
```

Use abs() and angle() to convert a complex number to polar form (magnitude & angle):

```
z = -1+j
```

```
z = -1.0000 + 1.0000i
```

```
r = abs(z)
```

```
r = 1.4142
```

```
theta = angle(z)    % extract the angle
```

```
theta = 2.3562
```

Visualizing z on the complex plane, it is clear that theta should be 3*pi/4 = 135 deg:

```
theta * 180/pi    % should be 135 deg
```

```
ans = 135
```

Now convert from polar to cartesian form using Euler's formula:

```
r * exp(j*theta)  % ans = -1.0000 + 1.0000i
```

```
ans = -1.0000 + 1.0000i
```

We could also do explicit rectangular conversion:

```
r*(cos(theta) + j*sin(theta))     % ans = -1.0000 + 1.0000i
```

```
ans = -1.0000 + 1.0000i
```

Careful with the polar angle! To find the angle given a complex number z, you might be tempted to use:

```
theta = atan(imag(z)/real(z))
```

```
theta = -0.7854
```

Which yields -0.7854 = -pi/4, not the (correct) 3*pi/4 value.

The above atan() trig won't always work. In this case, z is in the wrong *quadrant* of the plane for a simple atan() calculation. Looking at the planar geometry/trig for this problem, we'd actually need to use the supplement of this angle:

```
theta = atan(imag(z)/real(z)) + pi    % --> 2.3562 rad = 3*pi/4
```

```
theta = 2.3562
```

It is fairly common in planar geometry, when doing polar coordinate conversions, to encounter such a problem. Matlab, and most computer languages, have a built-in "4 quadrant" atan function to handle it.

To get the correct polar angle for a point at coordinates (x,y) in the plane, use **atan2(y,x)**

Note there are *two* separate inputs, and note their order! Applied to a complex number, the "x" coordinate is the real part, the "y" coordinate is the imag part, so:

```
theta = atan2(imag(z), real(z))
```

```
theta = 2.3562
```

gives the right polar angle.

This 4 quadrant atan2() functionality is built into the angle() function for complex numbers, without you needing to separate out real and imag parts.

Finally, note (again) that *all* Matlab's built-in functions can work with complex numbers

```
sqrt(2j)    % ans = 1.0000 + 1.0000i
```

```
ans = 1.0000 + 1.0000i
```