GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Summer 2022
Lab #2: Using Complex Exponentials

Date: June 2, 2022

Pre-Lab: You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section ***before your assigned lab time.***

Verification: The Exercise section of each lab must be completed **before your assigned Lab time** and the steps marked *Instructor Verification* must be demonstrated **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on each step of the **Instructor Verification** worksheet. Demonstrate each step to the TA or instructor.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students, but you cannot give or receive any written material or electronic files. In addition, you are not allowed to use or copy material from old lab reports from previous semesters. Your submitted work must be your own original work.

1 Introduction and Overview

The goal of this laboratory is to gain familiarity with complex numbers and their use in representing sinusoidal signals such as $x(t) = A\cos(\omega t + \varphi)$ as complex exponentials $z(t) = Ae^{j\varphi}e^{j\omega t}$. The key is to use the complex amplitude, $X = Ae^{j\varphi}$, and then the real part operator applied to Euler's formula:

$$x(t) = \Re\{Xe^{j\omega t}\} = \Re\{Ae^{j\varphi}e^{j\omega t}\} = A\cos(\omega t + \varphi)$$

Manipulating sinusoidal functions using complex exponentials turns trigonometric problems into simple arithmetic and algebra. In this lab, we first review the complex exponential signal and the phasor addition property needed for adding cosine waves. Then we will use MATLAB to make plots of phasor diagrams that show the vector addition needed when combining sinusoids.

1.1 Complex Numbers in MATLAB

MATLAB can be used to compute complex-valued formulas and also to display the results as vector or "phasor" diagrams. For this purpose several new MATLAB functions have been written and are available as part of the SP-First toolbox: `http://dspfirst.gatech.edu/matlab/toolbox/`. Make sure that this toolbox has been installed[1] by doing `help` on the new M-files: `zvect`, `zcat`, `ucplot`, `zcoords`, and `zprint`. Each of these functions can plot (or print) several complex numbers at once, when the input is formed into a vector of complex numbers. For example, try the following function call and observe that it will plot five vectors all on one graph:

```
zvect( [ 1+j, j, 3-4*j, exp(j*pi), exp(2j*pi/3) ] )
```

[1]Correct installation means that the `spfirst` directory will be on the MATLAB path. Try `help path` if you need more information.

Here are some of MATLAB's complex number operators:

`conj`	Complex conjugate
`abs`	Magnitude
`angle`	Angle (or phase) in radians
`real`	Real part
`imag`	Imaginary part
`i,j`	pre-defined as $\sqrt{-1}$
`x = 3 + 4i`	`i` suffix defines imaginary constant (same for `j` suffix)
`exp(j*theta)`	Function for the complex exponential $e^{j\theta}$

Each of these functions takes a vector (or matrix) as its input argument and operates on each element of the vector. Notice that the function names `mag()` and `phase()` do not exist in MATLAB.[2]

Finally, there is a complex numbers drill program called: which uses a GUI to generate complex number problems and check your answers. *Please spend some time with this drill since it is very useful in helping you to get a feel for complex arithmetic.*

When unsure about a command, use `help`.

1.2 Sinusoid Addition Using Complex Exponentials

Recall that sinusoids may be expressed as the real part of a complex exponential:

$$x(t) = A\cos(2\pi f_0 t + \varphi) = \Re\left\{Ae^{j\varphi}e^{j2\pi f_0 t}\right\} \tag{1}$$

The *Phasor Addition Rule* presented in Section 2.6.2 of the text shows how to add several sinusoids:

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_0 t + \varphi_k) \tag{2}$$

assuming that each sinusoid in the sum has the *same* frequency, f_0. This sum is difficult to simplify using trigonometric identities, but it reduces to an algebraic sum of complex numbers when solved using complex exponentials. If we represent each sinusoid with its *complex amplitude*

$$X_k = A_k e^{j\varphi_k} \tag{3}$$

Then the complex amplitude of the sum X_s is

$$X_s = \sum_{k=1}^{N} X_k = A_s e^{j\varphi_s} \tag{4}$$

Based on this complex number manipulation, the *Phasor Addition Rule* implies that the amplitude and phase of $x(t)$ in (2) are A_s and φ_s, so

$$x(t) = A_s \cos(2\pi f_0 t + \varphi_s) \tag{5}$$

We see that the sum signal $x(t)$ in (2) and (5) is a single sinusoid that still has the same frequency, f_0, and it is periodic with period $T_0 = 1/f_0$.

[2]In the latest release of MATLAB a function called `phase()` is defined in a seldom used toolbox; it does more or less the same thing as `angle()` but also attempts to add multiples of 2π when processing a vector.

1.3 Harmonic Sinusoids

There is an important extension where $x(t)$ is the sum of N cosine waves whose frequencies (f_k) are *different.* If we concentrate on the case where the frequencies (f_k) are all multiples of one basic frequency f_0, i.e.,

$$f_k = kf_0 \qquad \text{(HARMONIC FREQUENCIES)}$$

then the sum of N cosine waves becomes

$$x_h(t) = \sum_{k=1}^{N} A_k \cos(2\pi k f_0 t + \varphi_k) = \Re\left\{\sum_{k=1}^{N} X_k \, e^{j2\pi k f_0 t}\right\} \tag{6}$$

This signal $x_h(t)$ has the property that it is also periodic with period $T_0 = 1/f_0$, because each of the cosines in the sum repeats with period T_0. The frequency f_0 is called the *fundamental frequency*, and T_0 is called the *fundamental period.* (Unlike the single frequency case, there is no phasor addition theorem to combine the harmonic sinusoids.)

2 Pre-Lab

Please do the exercises in this section prior to coming to lab.

2.1 Complex Numbers

This section will test your understanding of complex numbers when plotted as vectors. Use $z_1 = 2e^{-j\pi/4}$ and $z_2 = \sqrt{3} + j$ for all parts of this section.

(a) Enter the complex numbers z_1 and z_2 in MATLAB, then plot them with `zvect()`, and also print them with `zprint()`.

When unsure about a command, use `help`.

Whenever you make a plot with `zvect()` or `zcat()`, it is helpful to provide axes for reference. An x-y axis and the unit circle can be superimposed on your `zvect()` plot by doing the following:

```
hold on, zcoords, ucplot, hold off
```

(b) Compute the conjugate z^* and the inverse $1/z$ for both z_1 and z_2 and plot the results as vectors. In MATLAB, see `help conj`. Display the results numerically with `zprint`.

(c) The function `zcat()` can be used to plot vectors in a "head-to-tail" format. Execute the statement `zcat([1+j,-2+j,1-2j]);` to see how `zcat()` works when its input is a vector of complex numbers.

(d) Compute $z_1 + z_2$ and plot the sum using `zvect()`. Then use `zcat()` to plot z_1 and z_2 as 2 vectors head-to-tail, thus illustrating the vector sum. Use `hold on` to put all 3 vectors on the same plot. If you want to see the numerical value of the sum, use `zprint()` to display it.

(e) Compute $z_1 z_2$ and z_2/z_1 and plot the answers using `zvect()` to show how the angles of z_1 and z_2 determine the angles of the product and quotient. Use `zprint()` to display the results numerically.

(f) Make a 2×2 subplot that displays four plots in one window, similar to the four operations done previously: (i) z_1, z_2, and the sum $z_1 + z_2$ on a single plot; (ii) z_2 and z_2^* on the same plot; (iii) z_1 and $1/z_1$ on the same plot; and (iv) $z_1 z_2$. Add a unit circle and x-y axis to each plot for reference.

2.2 Z-Drill

Work a few problems generated by the complex number drill program. To start the program simply type `zdrill`; if necessary, install the GUI and add `zdrill` to MATLAB's path. Use the buttons on the graphical user interface (GUI) to produce different problems.

2.3 Cell Mode in MATLAB

MATLAB has a formatting syntax that allows you to produce documentation at the same time that you make an M-file. A quick summary is that double percent signs followed by a space (`%%␣`) are interpreted as sections in cell mode so that your code is broken into natural blocks that can be run individually. In addition, the M-file can be "published" to an HTML file and then viewed as a nicely formatted web page. There are several sources with information about cell mode:

1. Videos:
   `https://www.youtube.com/watch?v=CWgl5Yl1txk`
   `https://www.youtube.com/watch?v=_TWBG95mCQU`
   and `https://www.mathworks.com/help/matlab/matlab_prog/run-sections-of-programs.html`

Please watch the videos, especially the first one, and read some of the documentation.

The MATLAB code in the following section uses cell mode. It can be published to HTML, and displayed in a web browser (the default is MATLAB's internal browser).

2.4 Vectorization

The power of MATLAB comes from its matrix-vector syntax. In most cases, loops can be replaced with vector operations because functions such as `exp()` and `cos()` are defined for vector inputs, e.g.,

$$\texttt{cos(vv)} = [\texttt{cos(vv(1)), cos(vv(2)), cos(vv(3)), ... cos(vv(N))}]$$

where `vv` is an N-element row vector. Vectorization can be used to simplify your code. If you have the following code that plots the signal in the vector `yy`,

```
M = 200;
for k=1:M
   x(k) = k;
   yy(k) = cos( 0.001*pi*x(k)*x(k) );
end
plot( x, yy, 'ro-' )
```

then you can replace the `for` loop with one line and get the same result with four lines of code:

```
M = 200;
x = 1:M;
yy = cos( 0.001*pi*x.*x );
plot( x, yy, 'ro-' )
```

Run these two programs to see that they give identical results, but note that the vectorized version runs much faster.

2.5 Vectorizing a 2-D Evaluation

You can also vectorize 2D functions. Suppose that you want to plot $f(u,v) = u^2 + v^2$ versus (u,v) over the domain $[-20, 20] \times [-20, 20]$. The result should be a parabolic surface. To avoid having nested `for` loops, we can use `meshgrid` instead:

```
u = -20:0.5:20;
v = -20:0.5:20;
[uu,vv] = meshgrid(u,v);
mesh(u,v,uu.*uu + vv.*vv)
```

The `meshgrid` function generates all the pairs (u, v) for the domain.

2.6 Functions

Functions are a special type of M-file that can accept inputs (matrices, vectors, structures, etc.) and also return outputs. The keyword `function` must appear as the first non-comment word in the M-file that defines the function, and that line of the M-file defines how the function will pass input and output arguments. The file extension must be lower case "m" as in `my_func.m`. See Section B-6 in Appendix B of the text for more discussion.

The following function has several mistakes (there are at least four). Before looking at the correct one below, try to find these mistake(s):

```
matlab mfile  [xx,tt] = badcos(ff,dur)
%BADCOS  Function to generate a cosine wave
%  usage:
%       xx = badcos(ff,dur)
%       ff = desired frequency
%      dur = duration of the waveform in seconds
%
tt = 0:1/(100*ff):dur;   %-- gives 100 samples per period
badcos = real(exp(2*pi*freeq*tt));
```

The corrected function should look something like:

```
function [xx,tt] = goodcos(ff,dur)
tt = 0:1/(100*ff):dur;   %-- gives 100 samples per period
xx = real(exp(2i*pi*ff*tt));
```

Notice that the word `function` must be at the beginning of the first line. Also, the exponential needs to have an imaginary exponent, and the variable `freeq` must be defined before being used. Finally, the function has `xx` as an output, so the variable `xx` must appear on the left-hand side of at least one assignment line within the function body. In other words, the function name is *not* used to hold values produced in the function.

2.7 Structures in MATLAB

MATLAB can do structures. Structures are convenient for grouping information together. For example, we can group all the information about a sinusoid into a single structure with fields for amplitude, frequency and phase. We could also add fields for other attributes such as a signal name, the signal values, and so on. To see how a structure might be used, run the following program which plots a sinusoid:

```
x.Amp = 7;
x.phase = -pi/2;
x.freq = 100;
x.fs = 11025;   %-- sampling rate controls the spacing of values on the time grid
x.times = 0:(1/x.fs):0.05;
x.values = x.Amp*cos(2*pi*(x.freq)*(x.times) + x.phase);
x.name = 'My Signal';
x           %---- echo the contents of the structure "x"
plot( x.times, x.values )
title( x.name )
```

Notice that the fields in the structure can contain different types of variables: scalars, vectors or strings.

You can also have arrays of structures. For example, if xx is array of sinusoid-structures with the same fields as above, you would reference one of the sinusoids via:

```
 xx(3).name, xx(3).Amp, xx(3).freq, xx(3).phase
%
 plot( xx(3).times, xx(3).values )
 title( [xx(3).name,' Amp=',num2str(xx(3).Amp),' Phase=',num2str(xx(3).phase)] )
```

Notice that the array name is xx, so the array index, 3, is associated with xx, e.g., xx(3).

3 In-lab Exercise: Complex Exponentials

In the Pre-Lab section of this lab, you saw examples of function M-files. In this section, you will write functions that can generate sinusoids, or sums of sinusoids.

For the instructor verification, you will have to demonstrate that you understand everything in a given subsection. It is not necessary to do everything in the subsections; skip parts that you already know. The Instructor Verification is usually placed close to the most important item, i.e., the most likely one to generate questions from the TAs.

3.1 Vectorization

Use the vectorization idea to write 1 or 2 lines of code that will perform the *same task as the inner loop* of the following MATLAB script without using a `for` loop. If you are ambitious, try to replace both loops with some vectorized code.

```
 %--- make a plot of sum of cosines
 % Create a single stream and designate it as the current global stream:
thisClass = RandStream('mt19937ar','Seed',1);
dt = 1/800;
XX = rand(thisClass,1,3).*exp(2i*pi*rand(thisClass,1,3)); %--Random amplitude and phases
freq = 20;
 ccsum = zeros(1,500);
 for kx = 1:length(XX)
     for kt = 1:500
         t = kt*dt;
         Ak = abs(XX(kx));
         phik = angle(XX(kx));
         ccsum(kt) = ccsum(kt) + Ak*cos(2*pi*freq*t + phik);
         tt(kt) = t;
     end
 end
 plot(tt,ccsum)   %-- Plot the sum sinusoid
 grid on, zoom on, shg
```

Instructor Verification (separate page)

3.2 M-file to Generate One Sinusoid

Write a function that will generate a **single** sinusoid, $x(t) = A\cos(2\pi f t + \varphi)$. The function should have the following input arguments: a sinusoid-structure with two fields for the frequency (f) in Hz, the complex amplitude $(X = Ae^{j\varphi})$, and then three other arguments: a duration argument `(dur)`, followed by an argument for the starting time `(tstart)`, and then a final argument which is the spacing between times, Δt. The function should return a structure having both of the fields of the input structure plus two new fields: the vector of values of the sinusoidal signal (x) along with the corresponding vector of times (t) at which the sinusoid values are known. The spacing between times in the time-vector, Δt, is a constant, but make sure that it is small enough so that there are at least 32 time points per period of the sinusoid. Call this function `makeCosVals()`. *Hint: use* `goodcos()` *from the Pre-Lab part as a starting point.* Here is a suggested template that needs to be completed for the M-file:

```
function sigOut = makeCosVals(sigIn, dur, tstart, dt )
%
freq = sigIn.freq;
X = sigIn.complexAmp;
%
%...(Fill in several lines of code)...
%
tt = tstart: dt : ???;    %-- Create the vector of times
xx = A*cos(...???;        %-- Vectorize the cosine evaluation
sigOut.times = ???;       %-- Put vector of times into the output structure
sigOut.values = ???;      %-- Put values into the output structure
```

Plot the result from the following call to test your function.

```
mySig.freq = 2;    %-- (in hertz)
mySig.complexAmp = 5*exp(j*pi/4);
dur = 2;
start = -1;
dt = 1/(32*mySig.freq);
mySigWithVals = makeCosVals( mySig, dur, start, dt );
%- Plot the values in sigWithVals
```

Instructor Verification (separate page)

3.3 Sinusoidal Synthesis with an M-file: Different Frequencies

Since we will generate many functions that are a *sum of sinusoids,* it will be convenient to have a MATLAB function for this operation. To be general, we should allow the frequency of each component (f_k) to be different. The following expressions are equivalent if we define the complex amplitude X_k as $X_k = A_k e^{j\varphi_k}$.

$$x(t) = \Re\left\{\sum_{k=1}^{N} X_k e^{j2\pi f_k t}\right\} = \Re\left\{\sum_{k=1}^{N} (A_k e^{j\varphi_k}) e^{j2\pi f_k t}\right\} \tag{7}$$

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \varphi_k) \tag{8}$$

3.3.1 Write a Sum of Sinusoids M-file

Write an M-file called `addCosVals.m` that will synthesize a waveform in the form of (7) using X_k defined in (3). The result is not a sinusoid unless all the frequencies are the same, so the output signal has to be represented by its values over some (finite) time interval.

Even though `for` loops are rather inefficient in MATLAB, *you must write the function with one outer loop in this lab.* The inner loop should be vectorized. The first few statements of the M-file are the comment lines—they should look like:

```
function    sigOut = addCosVals( cosIn, dur, tstart, dt )
%ADDCOSVALS  Synthesize a signal from sum of sinusoids
%                (do not assume all the frequencies are the same)
%
%  usage:   sigOut = addCosVals( cosIn, dur, tstart, dt )
%
%   cosIn = vector of structures; each one has the following fields:
%      cosIn.freq = frequency (in Hz), usually none should be negative
%      cosIn.complexAmp = COMPLEX amplitude of the cosine
%
%   dur = total time duration of all the cosines
%   start = starting time of all the cosines
%   dt = time increment for the time vector
% The output structure has only signal values because it is not necessarily a sinusoid
%      sigOut.values = vector of signal values at t = sigOut.times
%      sigOut.times  = vector of times, for the time axis
%
%    The sigOut.times vector should be generated with a small time increment that
%         creates 32 samples for the shortest period, i.e., use the period
%         corresponding to the highest frequency cosine in the input array of structures.
```

In order to verify that this M-file can synthesize ***harmonic*** sinusoids, try the following test:

```
ss(1).freq = 21;  ss(1).complexAmp = exp(j*pi/4);
ss(2).freq = 15;  ss(2).complexAmp = 2i;
ss(3).freq =  9;  ss(3).complexAmp = -4;
%
dur = 1;
tstart = -0.5;
dt = 1/(21*32);   %-- use the highest frequency to define delta_t
%
ssOut = addCosVals( ss, dur, tstart, dt );
%
plot( ssOut.????, ssOut.???? )
%
```

Use MATLAB to make a plot of `ssOut`. Notice that the waveform is periodic. Measure its period and state how the period is related to the fundamental frequency which is 3 Hz in this case.

Lab #2
ECE-2026 Summer-2022
INSTRUCTOR VERIFICATION LIST

Part 3.1 Replace the inner `for` loop with only one or two lines of vectorized MATLAB code. Show the sinusoids generated by both the codes - with and without for loop - to the instructors.

Part 3.2 and other places: Learn to publish MATLAB's `script` capability to create and save the plots when doing the Instructor Verifications.

Part 3.3.1 Show that your `addCosVals.m` function is correct by running the test in Section 3.3.1 and plotting the result. Measure the period of signal in the structure `ssOut`, and explain its relationship to the fundamental frequency.