GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

**ECE 2026 Fall 2013**

**Lab #2: Introduction to Complex Exponentials & Phasor Addition**

Date: 9-12 September 2013

---

Each Lab assignment in ECE2026 consists of three parts: Pre-Lab, In-lab Tasks, and Take-home Questions. It requires you to come into lab prepared. Be sure to read the entire lab carefully before arriving.

**Pre-Lab:** You should read the Pre-Lab section of the lab and go over all exercises in this section before going to your assigned lab session. Although you do not need to turn in results from the Pre-Lab, doing the exercises therein will help make your in-lab experience more rewarding and go more smoothly with less frustration and panicking.

**In-lab Tasks and Verification:** There are a number of designated tasks for each student to accomplish during the lab session. Students are encouraged to read, prepare for and even attempt at these tasks beforehand. These tasks must be completed **during your assigned lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply put a plastic cup on top of your PC and demonstrate the step to one of the TAs or the professor. (You can also use the plastic cups to indicate if you have a more general question, i.e. you can use it to get our attention even if you don't have an Instructor Verification ready.)

**Take-home Questions:** At the end of each lab sheet below all the verification steps, several questions are to be answered by the student, who can choose to complete the answers while in the lab or after leaving the lab.

The lab sheet with all verification signatures and answers to the questions needs to be turned in to the Lab-grading TA at the beginning of the next lab session.

***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 and you are allowed to consult old lab reports, but you cannot give or receive written material or electronic files. Your submitted work should be original and it should be your own work.***

---

**PRINTING BUDGET:** For the printers in the ECE labs, you have a quota. Please limit your printing to essential items for the labs. If you need to print lecture slides and other large documents, use the central (OIT) printing facilities.

---

# 1 Introduction & 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 $z(t) = Ae^{j\omega t}e^{j\varphi}$. The key is to use the complex amplitude, $X = Ae^{j\varphi}$, and then the real part operator applied to Euler's formula:

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

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 on the *SP First*

*CD-ROM*. 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:


*SP-First* MATLAB *Toolbox*

```
zvect( [ 1-j, j, 3+4*j, exp(j*0.3*pi), exp(2j*pi/5) ] )
```

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()`** are no longer in use in MATLAB.


*Zdrill*

Finally, there is a complex numbers drill program called:

```
zdrill
```

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:

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

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 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 equation (2) are $A_s$ and $\varphi_s$, so

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

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

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 $T_0 = 1/f_0$.

### 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.,

$$f_k = k f_0 \qquad \textbf{(HARMONIC FREQUENCIES)}$$

then the sum of $N$ cosine waves given by (2) becomes

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

This particular 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, we **do not and cannot** use 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 $z_1 = 5e^{-j\pi/4}$ 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. For example, execute the statement **`zcat([3+1j,-1+2j,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 $z_1/z_2$ 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 (What is a subplot in MATLAB?) 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 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.,

```
cos(vv) = [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 = 100;
for k=1:M
      x(k) = k;
      yy(k) = cos( 0.02*pi*x(k)*x(k) );
end
plot( x(1:M), yy(1:M) )
```

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

```
M = 100;
yy = cos( 0.02*pi*(1:M).*(1:M) );
plot( 1:M, yy )
```

Run these two programs to see that they give identical results. Note that the vectorized version runs much faster. In order to make the speed difference a bit more obvious, you may need to use a large **M**.

### 2.4 Functions

Functions are a special type of M-file that can accept inputs (matrices and vectors) and also return outputs. The keyword function must appear as the first word in the M-file that defines the function, and the first 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/(120*ff):dur; %-- gives 120 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/(120*ff):dur; %-- gives 120 samples per period
xx = real(exp(2i*pi*ff*tt));
```

**Explanation:** Notice the word function at the beginning of the first line. Also, the exponential needs to have an imaginary exponent, and the variable **freeq** has not been 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.

# 3 Exercise: Complex Exponentials

In the Pre-Lab part of this lab, you learned how to write function M-files. In this section, you will write two functions that can generate sinusoids, or sums of sinusoids.

### 3.1 Vectorization

Use the vectorization idea to write two or three lines of code that will perform the same task as the following MATLAB script without using a **`for`** loop.

```
%--- make a plot of a weird signal
N = 180;
for k=1:N
xk(k) = k/60;
rk(k) = sqrt( xk(k)*xk(k) - 0.5 )+ 0.25;
sig(k) = exp(j*2*pi*rk(k));
end
plot( xk, real(sig), 'mo-', xk, imag(sig), 'go-' )
```

***Note:*** there is a difference between the two multiply operations **`rr*rr`** and **`rr.*rr`** when **`rr`** is a vector.

**Instructor Verification** (separate page)

### 3.2 M-file to Generate One Sinusoid

Write a function that will generate a **single** sinusoid, $x(t) = A\cos(2\pi f t + \varphi)$, by using input arguments for frequency ($f$) in Hz (**`freq`**), complex amplitude $X = Ae^{j\varphi}$ (**`camp`**), duration in second (**`dur`**) and starting time (**`tstart`**). Keep in mind that while we are generating a sinusoid, as a function of time, the result is in fact a sequence of numbers, which are the values of the function sampled at certain time instances. Therefore, we need to specify how many such values we'll generate on a unit-time basis, i.e., the so-called sampling rate (**`fs`**). We'll call this function **`onecos()`**; i.e., the function call should look like **`onecos(freq, camp, fs, dur, tstart).`** The function should return two output vectors: the values of the sinusoidal signal ($x$) and corresponding times ($t$) at which the sinusoid values are known. (*Hint: use* **`goodcos()`** *from the Pre-Lab part as a starting point.* Figure out how to define the sequence for the "time instances" at which the signal values will be generated. The incremental value between successive time instances is called the **sampling interval**, which is equal to the reciprocal of the sampling rate, **`fs`**.) Plot the result from the following call to test your function.

```
[xx0,tt0] = onecos(4, 2.5*exp(-j*pi/6), 100, 0.8, -0.15); % (freq in Hz)
```

**Instructor Verification** (separate page)

### 3.3 Sinusoidal Synthesis with an M-file: Summing Sinusoids of 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 will 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 $X_k = A_k e^{j\varphi_k}$ .

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

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

#### 3.3.1 Write A Sum-of-Sinusoids Synthesizer M-file

Write an M-file called **add_sines.m** that will synthesize a waveform in the form of (7) using $X_k$ defined in (3). Although **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 [xx,tt] = add_sines(freqs, Camps, fs, dur, tstart)
%ADD_SINES Synthesize a signal from sum of complex exponentials
% usage:
% [xx,tt] = add_sines(freqs, Camps, dur, tstart)
% freqs = vector of frequencies (usually none are negative)
% Camps = vector of COMPLEX amplitudes
% dur = total time duration of the signal
% tstart = starting time
% xx = vector of sinusoidal values
% tt = vector of times, for the time axis
%
% Note: freqs and Camps must be the same length.
% Camps(1) corresponds to frequency freqs(1),
% Camps(2) corresponds to frequency freqs(2), etc.
% The tt vector should be generated with a small time increment define by
% sampling rate, fs.
%
% Make use of onecos
```

The MATLAB syntax **length(freqs)** returns the number of elements in the vector **freqs**, so we do not need a separate input argument for the number of frequencies. On the other hand, the programmer (that's you) should provide error checking to make sure that the lengths of **freqs** and **Camps** are all the same. See **help error**.

### 3.3.2 Testing

In order to verify that this M-file can synthesize sinusoids, try the following test and plot the result:

```
[xx0,tt0] = add_sines([6,3,18],[2*exp(j*pi/4),3-j,1.2],99,1.5,-0.2); %-Period = ?
```

Measure the DC value (which is also the average value over one period of the signal) and the period of **xx0** on the plot. The period is to be expressed in second (s); so you need to inspect the plot, find the period, and relate it back to time in second. Then write an explanation on the verification sheet of why the measured values are correct. One way to offer your explanation for the period is to relate the measured period to the frequencies of the component sinusoids (how many cycles of the first sinusoid will appear in this measured period, how many cycles of the second sinusoid, and so on) and then infer why your result must be correct. (As you will learn soon on Friday, when this M-file is used to synthesize harmonic waveforms, you must choose the entries in the frequency vector to be integer multiples of the fundamental frequency.)

**Instructor Verification** (separate page)

### 3.3.3 Periodicity of A Periodic Signal

Now, use **add_sines** of the sum-of-sinusoids synthesizer to produce a signal of form (7) that contains frequencies 40Hz, 64Hz and 80Hz with complex amplitudes, $4e^{j0.2}$ , $2.2e^{-j\pi/2}$, and 1.1, respectively, using a sampling rate of 164 samples per second for a duration of 10 seconds, i.e., from $t$ = 0 to $t$ = 10. In other words, generate 10-second long of $x(t)$:

$$x(t) = \mathrm{Re}\left\{ 4e^{j(80\pi t+0.2)} + 2.2e^{j(128\pi t-\pi/2)} + 1.1e^{j160\pi t} \right\} \tag{9}$$

Show a "stem" plot of the synthesized signal, called **xx**, from $t$ = 4 to $t$ = 4.5 to the instructor for verification. (If you don't know anything about stem plot, do **help stem**.)

**Instructor Verification** (separate page)

An interesting observation related to the period of the signal can be made. There are two different ideas of "period" to speak of here. One is the period in $x(t)$ and the other the "period" that exists in the generated, so-called discrete-time, sequence, **xx**. You can figure out the period of $x(t)$ from the expression of (9); give your answer in the Lab Sheet. For the period in **xx,** use the stem plot to find out the number of time increments, i.e., the number of sampling intervals, between repeated patterns of waveform. Report your result in the Lab Sheet as well. Can you explain what you have observed?

A more detailed discussion on the so-called periodic signal or harmonic signal will come on Friday (9/13). Question 3.3 at the end can wait until then.

# Lab #2
# ECE-2026 Fall-2013
# LAB SHEET

Turn this page in to your grading TA at the beginning of your next lab period

Name: _____ Date of Lab:_____

Part 3.1 Replace the inner for loop with only 1 or 2 lines of vectorized MATLAB code. Write the MATLAB code in the space below:

Verified: _____ Date/Time:_____

Part 3.2 Generated one sinusoid.

Verified: _____ Date/Time:_____

Part 3.3.2 Show that your **`add_sines.m`** function is correct by running the test in Section 3.3.2 and plotting the result. Measure the DC value and the period of **`xx0`** and explain why the measured values are correct. Write your explanations in the space below.

DC = _____; Period (s) = _____; Fundamental Frequency (Hz) = _____
Explanation:

Verified: _____ Date/Time:_____

Part 3.3.3 Show a stem plot of the generated signal within the specified time

Verified: _____ Date/Time:_____

**Questions:**

Question 3-1: Related to Part 3.3.3

a) What is the fundamental period of $x(t)$? _____
b) What is the period in **xx**? (i.e., number **N0** such that **xx[n]= xx[n+N0]**)? _____ sampling intervals.
c) What is the sampling interval in second?
d) Is there any relationship between answers a and b, considering c? Explain.

Question 3-2: The signal **xx** below obviously contains three sinusoids. Suppose **A**, **B**, and **C** have identical magnitudes, say 5, and you are allowed to adjust the phase values of **A**, **B**, and **C** in the code:

```
[xx,tt] = add_sines([18,36,90], [A,B,C], 500, 3, 0);
```

a) Give the values of **A**, **B**, and **C**, and the achieved maximum value in the resultant array **xx.**
b) What if **A**, **B**, and **C** do not have identical magnitudes?

Question 3-3: Fill in possible values of **f1**,**f2** and **f3**, and **A**,**B**,and **C** below

```
[xx,tt] = add_sines([f1,f2,f3], [A,B,C], 8000, 1000, 0);
```

such that the resultant array **xx** is aperiodic (not periodic).