GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2000
Lab #2: Introduction to Complex Exponentials

Date: 24–27 Jan 2000

Next Week: A short Lab Quiz will be held during the first 15–20 minutes of Lab #3. It will cover the basics of MATLAB from Labs 1 & 2.

Lab is held in College of Computing Building, room 309.

This is *the official* Lab #2 description; it is *different* from the one in Appendix C.2 of the text.

The Warm-up section of each lab 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 initialing on the **Instructor Verification** line. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the TA or instructor.

Lab Report: It is only necessary to turn in Section 4 with explanations as this week's lab report. Staple the **Instructor Verification** sheet to the end of your lab report as evidence that the appropriate steps were witnessed by the instructor.

The report will **due during the week of 31Jan.—4Feb. at the beginning of your lab**.

---

1 Introduction

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

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

2 Overview

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.

2.1 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 *DSP 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, the following

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

function call will plot five vectors all on one graph:

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

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.

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`.

2.2 Sinusoid Addition Using Complex Exponentials

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

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

The *Phasor Addition Rule* presented in Section 2.6.2 of the text (page 33) shows how to add several sinusoids:

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

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*

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

Then the complex amplitude of the sum is

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

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 ϕ_s, so

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

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 $T_0 = 1/f_0$.

3 Warm-up

The instructor verification sheet is at the end of this lab.

3.1 Complex Numbers

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

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

Instructor Verification (separate page)

(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.

3.2 Z-Drill

Work a few problems on the complex number drill program. To start the program simply type `zdrill`. Use the buttons on the graphical user interface (GUI) to produce different problems.

3.3 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 as

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

where `vv` is an N-element row vector. Use this vectorization idea to write 2 or 3 lines of code that will perform the same task as the following MATLAB script without using a `for` loop.

Instructor Verification (separate page)

```
%--- make a plot of a weird signal
N = 200;
for k=1:N
   xk(k) = k/50;
   rk(k) = sqrt( xk(k)^2 + 2.25 );
   sig(k) = exp(j*2*pi*rk(k));
end
plot( xk, real(sig), 'o-' )
```

3.4 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 ASCII 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.5 in Appendix B for more discussion.

The following exercises are designed to help you write functions in MATLAB.

(a) Find the mistake(s) in the following function (there are at least three):

```
  [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/(20*ff):dur;  %-- gives 20 samples per period
badcos = cos(2*pi*freeq*tt);
```

(b) Write a function that will generate a single sinusoid, $x(t) = A\cos(\omega t + \phi)$, by using four input arguments: amplitude (A), frequency (ω), phase (ϕ) and duration `(dur)`. The function should return two outputs: the values of the sinusoidal signal (x) and corresponding times (t) at which the sinusoid values are known. Make sure that the function generates 20 values of the sinusoid per period.
Call this function `mycos()`. *Hint: use* `badcos()` *from part (a) as a starting point.*

Demonstrate that your `mycos()` function works by plotting the output for the following parameters: $A = 95, \omega = 200\pi$ rad/sec, $\phi = \pi/5$ radians, and `dur = 0.025` seconds. Be prepared to explain to the lab instructor features on the plot that indicate how the plot has the correct period and phase. What is the expected period in millisec?

Instructor Verification (separate page)

4 Lab Exercises: Complex Exponentials

4.1 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 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 $X_k = A_k e^{j\phi_k}$.

$$x(t) \;=\; \Re e\left\{\sum_{k=1}^{N} X_k e^{j2\pi f_k t}\right\} \tag{6}$$

$$x(t) \quad = \quad \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi_k) \tag{7}$$

4.1.1 Write the Function M-file

Write an M-file called `add_cos.m` that will synthesize a waveform in the form of (6). Although `for` loops are rather inefficient in MATLAB, *you must write the function with one loop in this lab*. The first few statements of the M-file are the comment lines—they should look like:

```
function        [xx,tt] = add_cos(fk, Xk, fs, dur, tstart)
%ADD_COS  Function to synthesize a sum of cosine waves
%  usage:
%    [xx,tt] = add_cos(fk, Xk, fs, dur, tstart)
%      fk = vector of frequencies
%              (these could be negative or positive)
%      Xk = vector of complex amplitudes: Amp*e^(j*phase)
%      fs = the number of samples per second for the time axis
%     dur = total time duration of the signal
%  tstart = starting time (default is zero, if you make this input optional)
%      xx = vector of sinusoidal values
%      tt = vector of times, for the time axis
%
%    Note: fk and Xk must be the same length.
%             Xk(1) corresponds to frequency fk(1),
%             Xk(2) corresponds to frequency fk(2), etc.
```

The MATLAB syntax `length(fk)` returns the number of elements in the vector `fk`, 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 `fk` and `Xk` are the same. See `help error`. Finally, notice that the input `fs` defines the number of samples per second for the cosine generation; in other words, we are no longer using 20 samples per period.
Include a copy of the MATLAB *code with your lab report.*

4.1.2 Default Inputs

You can make the last input argument(s) take on default values if you use the `nargin` operator in MATLAB. For example, `tstart` can be made optional by including the following line of code:

```
if nargin<5, tstart=0, end   %--default value is zero
```

4.1.3 Testing with Sounds (OPTIONAL)

Summation: Use your `add_cos()` function to generate the **sum** of three sinusoids with frequencies that correspond to notes in a piano chord: $f_1 = 440$ Hz, $f_2 = 555$ Hz, and $f_3 = 660$ Hz. If these 3 tones are played at the same time, the result should be close to an A-major chord. Make all the amplitudes equal, pick the phases to be $\frac{1}{2}\pi$, and make the length of the signal 0.7 seconds. Use a sampling rate of 11025 Hz. Play the signal with `soundsc()` to verify that it makes a pleasant sounding chord.

4.2 Representation of Sinusoids with Complex Exponentials

In MATLAB consult `help` on `exp`, `real` and `imag`. Be aware that you can also use the DSP First function `zprint` to print the polar and rectangular forms of any vector of complex numbers.

(a) Generate the signal $x(t) = \Re e\{je^{j6\pi t} - 3e^{j6\pi t} + (2-j2)e^{j6\pi t}\}$ and make a plot versus t. Use the `add_cos` function and take a range for t that will cover 2 or 3 periods. *Include the* MATLAB *code with your report.*

(b) From the plot of $x(t)$ versus t, and measure the frequency, phase and amplitude of the sinusoidal signal by hand. Show annotations on the plots to indicate how these measurements were made and what the values are. Compare to the calculation in part (c).

(c) Use the phasor addition theorem or MATLAB to determine the magnitude and phase of $x(t)$ from the complex amplitude.

4.3 Multipath Fading

In a mobile radio system (e.g., cell phones or AM radio), there is one type of degradation that can be modeled easily with sinusoids. This is the case of *multipath fading* caused by reflections of the radio waves which interfere destructively at some locations. Consider the scenario diagrammed in Fig. 1 where a vehicle traveling on the roadway receives signals from two sources: directly from the transmitter and reflections from another object such as a large building. The total received signal at the vehicle is the sum of two signals which are

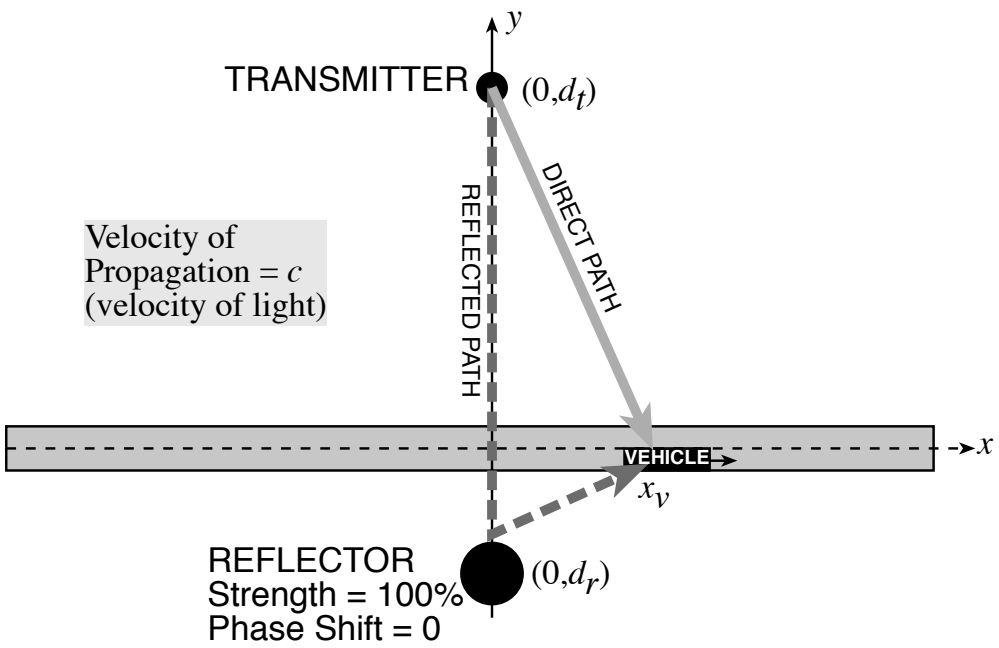

Figure 1: Scenario for multipath in mobile radio. A vehicle traveling on the roadway receives signals from two sources: the transmitter and a reflector.

themselves delayed versions of the transmitted signal, $s(t)$.

(a) The amount of the delay (in seconds) can be computed for both of the propagation paths. First of all, consider the direct path. The time delay is the distance divided by the speed of light (3×10^8 m/s). Write

a mathematical expression for the time delay in terms of the vehicle position x_v and the transmitter location $(0, d_t)$. Call this delay time t_1 and make sure that you express it as a function of x_v, i.e., $t_1(x_v)$.
Notice that the y-axis has been chosen conveniently so that both the transmitter and the reflector have an x coordinate equal to zero.

(b) Now write a mathematical formula for the time delay of the signal that travels the reflected path from the transmitter at $(0, d_t)$ to the reflector at $(0, d_r)$ and then to the vehicle at $(x_v, 0)$. Call this delay time t_2 and make sure that you also express it as a function of x_v, i.e., $t_2(x_v)$. In this case, you must add together two delays: transmitter to reflector and then reflector to vehicle.

(c) The received signal at the vehicle, $r_v(t)$, is the sum of the two delayed copies of the transmitter signal

$$r_v(t) = s(t - t_1) + s(t - t_2)$$

where $s(\cdot)$ is the transmitted signal, and the reflection is assumed to be perfect.[2]

Assume that the source signal $s(t)$ is a zero-phase sinusoid at $f_0 = 150$ MHz; and also assume that the transmitter is located at $(0, 1000)$ meters and the reflector at $(0, -500.5)$ meters. Then the received signal is the sum of two sinusoids. Derive the complex amplitudes for each, and write a MATLAB program that will generate the time delays and then add the complex amplitudes for a whole set of vehicle positions. Include this MATLAB code in your report and explain how it works.

Vectorization: It is likely that your previous programming skills would lead you to write a loop to do this implementation. The loop would run over all possible values of x_v, and would add the two complex amplitudes calculated at each x_v.
However, there is a *much more efficient way in* MATLAB, if you think in terms of vectors (which are really lists of numbers). In the vector strategy, you would make a vector containing all the vehicle positions; then do the distance and time delay calculations to generate a vector of time delays; next, a vector of complex amplitudes would be formed. In each of these calculations, only one line of code is needed and *no loops*.
You need two vectors of complex amplitudes (one for the direct path and the other for the reflected path), so the whole process is done twice and then you can perform a vector add of the two complex amplitudes.

(d) Utilize the MATLAB program from the previous part to generate a plot of signal strength versus vehicle position, x_v, over the interval from -50 meters to +150 meters. Assume that *signal strength* is defined to be the peak value of the received sinusoid, $r_v(t)$.

(e) Explain your results from the previous part. In particular, what are the largest and smallest values of received signal strength? Why do we get those values? Are there vehicle positions where we get complete signal cancellation? If so, determine those vehicle positions.

[2]For simplicity we are ignoring propagation losses: When a radio signal propagates over a distance R, its amplitude will be reduced by an amount that is inversely proportional to R^2.

Lab #2
ECE-2025
Spring-2000
INSTRUCTOR VERIFICATION SHEET

Staple this page to the end of your Lab Report.

Name: _____ Date of Lab: _____

Part 3.1(d) Show complex addition as the sum of vectors (make the plot with `zcat` and `zvect`).

Verified:_____ Date/Time:_____

Part 3.3 Replace `for` loop with only 2 or 3 lines of vectorized MATLAB code. Write the MATLAB code in the space below:

Verified:_____ Date/Time:_____

Part 3.4(c)
Write the function `mycos()` and demonstrate that it works by plotting the output for the following parameters: $A = 95$, $\omega = 200\pi$ rad/sec, $\phi = \pi/5$ radians, and `dur = 0.025` seconds:

Verified:_____ Date/Time:_____