GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2004
Lab #2: Introduction to Complex Exponentials

Date: 20–26 Jan. 2004

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. You **MUST** complete the online Pre-lab exercise on Web-CT at the beginning of your lab session. You can use MATLAB or any notes you might have, but you cannot discuss the exercises with any other students. You will have approximately 20 minutes at the beginning of your lab session to complete the online Pre-Lab exercise. The Pre-Lab exercise for this this lab also includes some questions about concepts from the *previous* report.

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 signing 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. After completing the warm-up section, turn in the verification sheet to your TA.

It is only necessary to turn in Section 4 as the lab report for this lab. More information on the lab report format can be found on Web-CT under the 'Information" link. You are asked to **label** the axes of your plots and include a title for every plot. In order to reduce missing plots, include your plot as a figure *embedded* within your report. For more information on how to include figures and plots from MATLAB in your report file, consult the "information" link on Web-CT. If you still do not know how to do so, ask your TA.

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 the submitted work should be original and it should be your own work.

The lab report and verification will be graded out of 100 points. The Pre-Post-Lab questions will be graded separately.

The lab report will be **due during the period 27-Jan. through 2-Feb. at the start of your lab.**

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 and Overview

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

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

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:

```
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.[2]

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:

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

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

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

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

[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.

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 $T_0 = 1/f_0$.

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.,

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

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

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

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, there is no phasor addition theorem here to combine the harmonic sinusoids.)

2 Pre-Lab

You should do all the exercises in this section to prepare for the on-line pre-lab questions.

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 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 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 The Notebook Option

The purpose of this section is to introduce you to the notebook capability that links MATLAB and Microsoft Word under Windows. This option allows you to execute MATLAB commands, scripts and functions from inside a Microsoft Word document and then have results (including graphs) displayed automatically inside that same MS-Word document. This should make it easy to create lab reports by reducing the problems (and pain) associated with importing figures and results into MS-Word. You are not required to use this feature to write your lab reports, but many students find it useful.

Note that GUI items such as `zdrill` will not work from inside MS-Word. Also note that MATLAB functions cannot be defined from inside a MS-Word document.

2.4 Notebook setup

Depending on whether or not you have "Administrator privileges" on your Windows machine, there are two ways to set up the `notebook` functionality (which works in versions 5.x and 6.x of MATLAB).

1. This *easy* setup works if your Windows account has administrator privileges: From inside MATLAB, type `notebook -setup`, and follow the instructions given.[3]

2 This *harder* setup that works even for accounts with non-administrative privileges (i.e., it works in ECE Labs such as BH-216).

   (a) Open MS-Word and then go to the menu `Tools->Macro->Security` and select `medium` instead of `high`. Click the `OK` button.

[3] Once the the setup is complete, click on the start menu and find the `New Office Document` option. Under the `General tab` you should find `m-book.dot`. Click on it to start MS-Word. This might also start up MATLAB if it is not already running.

(b) Now go to the `File` menu in MS-Word, and open the file `m-book.dot` which is in the folder `C:\Appl\MatlabR13\Notebook\PC\`.

(c) A dialog box comes up where you type the path to the `matlab.exe` file.
Type in `C:\Appl\MatlabR13\Bin\win32\`.

(d) MATLAB will start up and once it has loaded up, return to MS-Word and select the `File` menu and pick `New M-Book`.

- A new MS-Word window will come up and the setup is complete. MS-Word will have a new menu called `Notebook` which contains a number of new commands for communicating with MATLAB.

2.4.1 Using MATLAB from MS-Word

Once the `notebook` capability has been installed you can start it by either: (1) typing `notebook` at the MATLAB command prompt, or (2) by selecting `New M-Book` in the `File` menu of MS-Word.

- To verify that the `notebook` capability works, from inside MS-Word, type `var_A=1+1i-6`, and then press the *Ctrl-Enter* keys together. This should give you the answer in the MS-Word document.
- To create a plot inside MS-Word, type the the line below (followed by *Ctrl-Enter*)
`t=0:0.1:pi; y=sin(3*t+0.05); plot(t,y); title('My First Plot')`
- To call one of your functions or MATLAB script files, just type in the name followed by *Ctrl-Enter*. Built-in MATLAB functions are called in the same way, for example
`[V,lam] = eig([1 2 3; 3 3 3; -1 2 3])` followed by *Ctrl-Enter*
- Take some time to investigate the other items and more advanced features under the `Notebook` menu that was installed in MS-Word. Additional help can be found in MATLAB by typing `helpdesk` and searching for Notebook.
- If you dislike the way that figures get inserted inside MS-Word with a grey background, then you can change to a white background by resetting one of MATLAB's defaults. One way is to execute the following at the beginning of the MS-Word document
`set(0,'DefaultFigureColor',[1 1 1]);set(0,'DefaultAxesColor',[1 1 1]);`

2.5 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 a certain signal,

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

then you can replace the `for` loop with one line and get the same result with 3 lines of code:

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

Run these two programs to see that they give identical results. Use the `notebook` capability to put the plots into a MS-Word document.

2.6 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-6 in Appendix B of the text for more discussion.

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

```
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 the word `function` in the first line, and the exponential needs to have an imaginary exponent. Also, the variable `freeq` has not been defined before being used. Finally, the function has `xx` as an output and hence the variable `xx` should appear in 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 Warm-Up: 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. Use MATLAB's `notebook` capability to put the plots into a MS-Word document when doing the Instructor Verifications.

3.1 Vectorization

Use the 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.

```
%--- make a plot of a weird signal
N = 200;
for k=1:N
   xk(k) = k/60;
   rk(k) = sqrt( xk(k)*xk(k) - 1 );
   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 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, $x(t) = A\cos(\omega t + \phi)$, by using input arguments for complex amplitude ($X = Ae^{j\phi}$), frequency (ω), duration (`dur`) and starting time (`tstart`). 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 exactly 32 values of the sinusoid per period. Call this function `onecos()`. *Hint: use* `goodcos()` *from the Pre-Lab part as a starting point.* Plot the result from the following call to test your function.

```
[xx0,tt0] = onecos( [5*exp(j*pi/4i)], [2], 2, -1);
```

Use the `notebook` feature to make this plot inside a MS-Word document.

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

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

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

3.3.1 Write the Sum of Sinusoids M-file

Write an M-file called `addCexp.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] = addCexp(Xk, fk, dur, tstart)
%ADDCEXP  Synthesize sinusoid from sum of complex exponentials
%  usage:
%    [xx,tt] = addCexp(Xk, fk, dur, tstart)
%      Xk = vector of COMPLEX amplitudes
%      fk = vector of frequencies   (all positive)
%     dur = total time duration of the signal
%  tstart = starting time
%      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 tt vector should be generated with a small time increment that
%         creates 32 samples per period. Use the period corresponding to
%         the highest frequency in the fk vector.
```

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 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 in MS-Word via the `notebook` capability.

```
[xx0,tt0] = addCexp( [5*exp(pi/3j),3i,-4], [6,4,0], 2, -1); %-Period = ?
```

Measure the DC value (which is also the average value) and also the period of `xx0` on the plot. Then write an explanation on the verification sheet of why the measured values are correct. Notice that 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)

4 Lab Exercise: Source Localization

There are many applications where is it important to be able to locate the precise position of an object. The Global Positioning System (GPS) provides this capability by using multiple satellites that emit signals that can be processed in a single receiver to calculate 3-D position anywhere on the earth. Another application would be locating the position of a cell phone caller in order to provide a **911** service that could automatically determine the source location of a distress call. This could be done through a process called *triangulation*, if multiple cell towers were receiving the call. Finally, a related comment is that humans have two ears so that the brain can process acoustic signals received at the two ears and determine the direction to the source of a sound and then use perceived amplitude to estimate the distance to the source.

In this lab we will use sinusoids to describe and analyze a simple scenario that demonstrates *acoustic source localization* in terms of phase differences and simple geometry. This same principle is used in many other applications including radars that locate and track airplanes. The source localization process consists of two distinct steps:

1. *Direction Finding* which is done using a pair of receivers to determine the angle of arrival of the acoustic signal (Fig. 1).

2. *Triangulation* which uses two angle of arrival measurements along with two distinct receiver-pair locations to calculate the position of the source (Fig. 2).

4.1 Direction Finding with Microphone Sensors

Consider a simple measurement system that consists of two microphones that can both hear the same source signal as in Fig. 1. If the microphones are placed some distance apart, then the sound must travel different

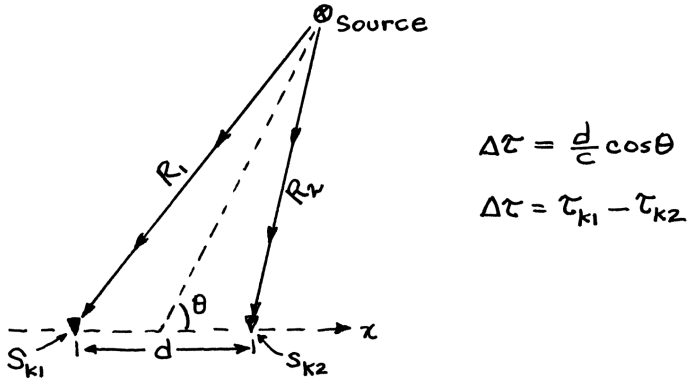

Figure 1: Estimating the angle of arrival (θ) using a pair of receivers. As drawn here the signal will arrive at sensor $\mathcal{S}_{k_2}$ first, so the difference between time-of-arrivals, $\tau_{k_1} - \tau_{k_2}$, would be a positive number.

paths from the source to the receivers. When the travel paths have different lengths, the two signals will arrive at different times. This time difference can be used to calculate the direction (θ) from the sensor pair to the source. A convenient reference points for measuring the angle is the midpoint between the sensors. Furthermore, if the sensors are placed along the x-axis then it would be convenient to measure the angle θ with respect to the x-axis.

The received signal at a sensor, e.g., $r(t)$, is a delayed copy of the transmitter signal $x(t)$. Thus we could write

$$r(t) = x(t - \tau_d)$$

where τ_d is the amount of time it takes for the signal to travel from the source location to the sensor and $x(\cdot)$ is the transmitted (sinusoidal) signal.[4] The travel time τ_d can be computed easily once we know the speed of sound (c) and the locations of the source and the sensors.

If we assume that the source signal is a sinusoid with frequency f_0, the received signals at the sensors are also sinusoids at f_0, but with different amplitudes and phases. Consider the case of one source transmitting the signal $x(t)$ to two sensors. The received signals will be labeled as in Fig. 1:

$$\begin{aligned} \text{Sensor } \mathcal{S}_{k_1}: &\quad r_{k_1}(t) = x(t - \tau_{k_1}) \\ \text{Sensor } \mathcal{S}_{k_2}: &\quad r_{k_2}(t) = x(t - \tau_{k_2}) \end{aligned}$$

where τ_{k_1} is the propagation time from the source to Sensor $\mathcal{S}_{k_1}$, and τ_2 the propagation time from the source to Sensor $\mathcal{S}_{k_2}$. The index k denotes different sensor pairs, because we need two sensor pairs in order to perform triangulation (see Section 4.2).

[4]For simplicity we will ignore propagation losses. Usually, the amplitude of an acoustic signal that propagates over a distance R will be reduced by an amount that is inversely proportional to R.

4.1.1 Direction Finding Approximation

The angle of arrival (θ) in Fig. 1 can be computed by measuring the time difference between the two sinusoids arriving at the two sensors. If $\Delta\tau = \tau_{k_1} - \tau_{k_2}$ is the time difference between the arrivals at sensors S_{k_1} and S_{k_2}, the formula for calculating the angle is

$$\Delta\tau = \frac{d}{c}\cos\theta \tag{9}$$

where c is the velocity of sound, and d is the inter-sensor spacing in meters, both of which should be known values. The formula in (9) involves two important constraints:

1. It is *approximate* so the value calculated for θ might not be exact. The approximation improves as the distance between the source and sensors increases. When R_1 (or R_2) is more than 10 times greater than the inter-sensor distance d, the formula (9) will have very little error; when the ratio is 50 or more the approximation is almost perfect.

2. The formula (9) cannot tell "up from down." Therefore, we must restrict our attention to targets in a half-space. For this lab exercise, we will assume that any target of interest lies above the sensors, i.e., $y_T > y_k$ in Fig. 2.

4.1.2 Data Generation M-file

It would be relatively easy to write a MATLAB program to generate data that simulates the propagating signal for the source localization and direction finding scenario of this lab, but we will provide this capability in the function **`sensordatagen`**. As the comments below describe, the inputs to **`sensordatagen.m`** must be the sensor locations in (x, y) space and the target location. Since complex numbers can be used to encode an (x, y) pair, the inputs must be given as $x + jy$ for each sensor. Note that multiple sensors can be done by giving a vector of sensor positions.

```
function Camps = sensordatagen( z_Target, z_Sensors )
%SENSORDATAGEN  generate data for a sinusoid emitted from a single
%               source location and received at multiple sensors
%
%  z_Target = complex number giving the (x,y) position of source
% z_Sensors = matrix of complex numbers for (x,y) positions of sensors
%     Camps = matrix of COMPLEX amplitudes, one for each sensor
%               size(Camps) is the same as size(z_Sensors)
%         NOTE: the Amps are all equal, but the value is random
%         NOTE: the initial phase is random
c = 340;    %-- sound velocity
freq = 85;  %-- frequency in Hz
```

The output from the **`sensordatagen.m`** program is a vector of complex amplitudes, one for each sensor. Since the signal arriving at any one sensor is a sinusoid, we only need to know its amplitude, frequency and phase. The frequency is fixed at $f_0 = 85$ Hz for this lab, and the complex amplitude gives the amplitude and phase of each sensor's sinusoidal signal.

Note: The **`sensordatagen.m`** program assigns a random amplitude and a random initial phase to the source sinusoid. Therefore, all the signals that are going to be used together must be generated by one call to **`sensordatagen.m`**. This provides a single snapshot of the source signal propagating to all the sensors.

4.2 Triangulation

If we know the positions of two sensor pairs and we measure the angle of arrivals, then it is a simple problem in geometry to find the source location. In Fig. 2 we would draw rays emanating from the sensors and find where the rays intersect. Writing equations for this situation can be done several ways, but the best approach

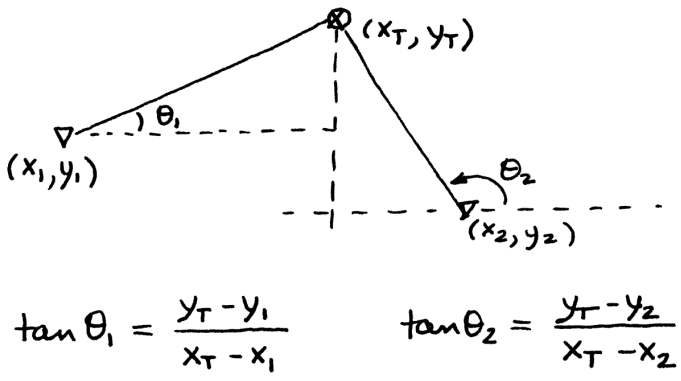

Figure 2: Compute the source position by intersecting two rays that emanate from the known receiver-pair locations at angles θ_1 and θ_2. The coordinates (x_k, y_k) are the midpoints of the sensor-pairs that would do direction finding to estimate θ_k.

is to generate two linear equations in the two unknowns (x_T, y_T) by using the tangent definition in Fig. 2. The linear equations are then solved via matrix inverse or with MATLAB's backslash operator \. Remember the constraint that the target will always be above the sensors which means that the angles will lie in the first or second quadrants.

4.3 Find Targets

This section outlines a number of processing steps to be implemented to show that you can produce some MATLAB functions that perform direction finding and source localization. Use the P-code version of the M-file **`sensordatagen`** to create test data when needed; recall that the frequency is $f_0 = 85$ Hz and sound speed is $c = 340$ m/s.

(a) Write a MATLAB function will take the complex amplitudes at two sensors along with the sensor positions and will produce an estimate of the angle of arrival (θ). Call your M-file **`myCalcAngle.m`** In your lab report, write an explanation of the mathematical steps used to calculate the angle.

(b) Show that your **`myCalcAngle.m`** function works by doing the following test. Place the sensors at the coordinates $(-0.3, 0)$ and $(0.3, 0)$. Then pick a value for your angle by using the sum of 110 plus 0.4 times the last two digits in your `gtnnnn` number as the angle in degrees. For example, `gte308b` would give $110 + (0.4 \times 8) = 113.2°$. Then generate a target position at that angle and at a distance of 50 meters.

(c) Repeat the previous part for all distances between 50 and 3 meters. Make a plot of the estimated angle-of-arrival versus distance for your known target angle. Comment on where the angle estimate seems to be good (i.e., has very little error) and where it is bad.

(d) Write a MATLAB function that will perform triangulation; its inputs should be two sensor-pair positions and two angles. Call this M-file **`myTriangulate.m`** and show that it works by using the

following inputs: one sensor-pair midpoint at $(-10, 1)$ and the other at $(10, -1)$, with angles $\theta_1 = 0$ and $\theta_2 = 3\pi/4$, respectively. The answer should be very easy to calculate (in your head), so you can use it (and other simple cases) to verify your M-file. A second test that will reveal a problem that must be treated as a special case is $\theta_1 = +\pi/4$ and $\theta_2 = \pi/2$.

(e) Apply your **`myTriangulate.m`** function to the data **`triTest`** found in the test file **`Lab02data.mat`**.[5] The sensor positions are the same as in the previous part. Plot of all the source locations found using **`plot(xT,yT,'rh')`** to plot a red hexagram at each source location.

(f) Now create a master M-file called **`findTarget.m`** that uses the previous M-files to figure out a source location from the sinusoidal data at two sensor-pairs. The inputs to **`findTarget.m`** will be a vector containing the sensor coordinates (as complex numbers) and a vector of four complex amplitudes that represent the received signals at the four sensors. The coordinates of the sensor pairs should be:

Sensor $\mathcal{S}_{1_1}$ at $(-10.3, 1)$	Sensor $\mathcal{S}_{2_1}$ at $(9.7, -1)$
Sensor $\mathcal{S}_{1_2}$ at $(-9.7, 1)$	Sensor $\mathcal{S}_{2_2}$ at $(10.3, -1)$

Notice that the first sensor pair is centered at $(-10, 1)$, while the second pair is centered at $(10, -1)$. Use the function **`sensordatagen.m`** to create data for a target at the same location as in part (b) above, and show that your function is able to find the correct source location.

(g) Finally, apply your **`findTarget.m`** function to all the data **`srcLocTest`** found in the test file **`Lab02data.mat`**. The format of the **`srcLocTest`** data is a $4 \times N$ matrix where each column of the matrix consists of four complex amplitudes generated from one source position. The sensor positions are those given in the previous part. You should write a loop to process the columns of **`srcLocTest`** one at a time, and then make a plot of all the source locations found. Use **`plot(xT,yT,'rh')`** to plot a red hexagram at each source location.

4.3.1 Miscellaneous Comments

Computing the difference of two angles can be done either in the obvious way, as $(\angle z_1 - \angle z_2)$, or by using the fact that complex conjugate negates the angle of a complex number. Thus, $\angle(z_1 z_2^*)$ gives the same result as the difference. Since angles are equal "modulo-2π" it is usually best to have the smallest angle. Think about the pros and cons of these two methods for the direction finding problem. For example, what happens when $\angle z_1 = 0.9\pi$ and $\angle z_2 = -0.95\pi$? How far apart are these two angles?

[5] The MAT file **`Lab02data.mat`** can be loaded into MATLAB with the **`load`** command.

Lab #2
ECE-2025
Spring-2004
INSTRUCTOR VERIFICATION SHEET

Turn this page in to your TA before the end of your lab period.

Name: _____ Date of Lab: _____

Part 3.1 Replace the `for` loop with a couple of lines of vectorized MATLAB code. Write the MATLAB code in the space below:

Verified:_____ Date/Time:_____

Part 3.3.2 Show that your `addCexp.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.

Verified:_____ Date/Time:_____

General: Use MATLAB's `notebook` capability to create and save the plots when doing the Instructor Verifications. Show the MS-Word document to your TA.

Verified:_____ Date/Time:_____