GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Fall 2023
Lab #1: Signal Data Handling

Date: Weeks of Aug. 28 and Sep. 4

Lab Instructions for Fall 2023: Students must attend the lab for their registered section at the designated times. Attendances will be taken before each class. A total of six labs will be conducted in Fall 2023. Each lab will typically last two weeks. The first week for each lab is devoted to student **Q&As** and the second is for student **demos** to the instructors for codes and lab results. Students will be grouped into teams of 2 or 3 by instructors before starting Lab 1. For each lab session, there will be two instructors administrating all activities. Students are encouraged to discuss lab contents as a team. At the end of each lab, each student is required to turn in an individual lab report in a single pdf file, containing answers to all lab questions, including codes and plots. Georgia Tech's **Honor Code** will be strictly enforced. See CANVAS Assignments for submission instructions.

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

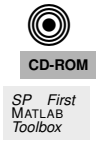
The goal of this laboratory is to gain familiarity with complex numbers and their use in representing sinusoidal signals such as $x(t) = A\cos(\omega t + \varphi)$ as complex exponentials $z(t) = Ae^{j\varphi}e^{j\omega t}$. The key is to use the complex amplitude, $X = Ae^{j\varphi}$, and then the real 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 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) ] )
```

[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:

```
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:

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

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.,

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

then the sum of N cosine waves becomes

$$x_h(t) = \sum_{k=0}^{N} A_k \cos(2\pi k f_0 t + \varphi_k) = \Re\left\{ \sum_{k=0}^{N} X_k \, e^{j2\pi k f_0 t} \right\} \tag{6}$$

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). For $k = 0$, φ_k is often set to 0, and A_0 is an offset, also known as a DC value.

2 Pre-Lab

Please read through the information below prior to attending your lab.

Objective: The objective of this lab is to learn to acquire signals into MATLAB and to display them for visualization. Waveform visualization is an important tool as well as a skill in signal analysis. Signals we learn to process in this course may be generated or recorder internally as a data array within MATLAB, or acquired from external sources such as a `wav` file. Furthermore, with a proper format converter, you may also record your own voice or music with your own computer and then load the recorded results, after converting them to the supported formats, into MATLAB for processing to produce interesting effects.

2.1 Overview

MATLAB will be used extensively in all the labs. The primary goal of this lab is to familiarize yourself with using MATLAB. Please read Appendix B: *M-file Programming in* MATLAB for an overview. Here are three specific goals for this lab:

1. Learn basic MATLAB commands and syntax, including the `help` system.
2. Write and edit your own script files in MATLAB, and run them as commands.
3. Learn a little about advanced programming techniques for MATLAB, i.e., vectorization.
4. Plot two sinusoids, add them and demonstrate the application of the *Phasor Addition Theorem.*

In Lab #0, you have gained or regained familiarity with MATLAB particularly in terms of arithmetic operations. Using simple examples, we learned to assign data values to a variable such as:

```
x = 10;  y = [1 2 3 4 5];   yy = [1 2 3; 4 5 6];
```

A slightly more sophisticated example in Lab #0 was:

```
xx = -0.2:0.01:0.3 ;
```

which will create a row vector of dimension 51, with values that span between -0.2 and 0.3 at an increment of 0.01. The indices of a vector `zz` can be manipulated to perform simple operations such as reversal, e.g.,

```
zz = exp(-0.1*(0:20)); zzReversed = zz(length(zz):-1:1)
```

In this lab, you will learn to use these expressions to generate some interesting *signals.*

Once a signal is generated, an immediate tool we use very often is a plot for visualization and inspection. You will learn to use common plot routines to show, in a number of ways, the signal you have generated or acquired (see below).

Often times, we need to process a signal that is obtained by some other means, instead of being generated internally with MATLAB commands. For example, you may have digital music on your computer, stored as files. Or, you may use your own computer to record a sound or your own speech, which can then be stored as an electronic file. In this lab, you will also learn to read or load external data into MATLAB for processing, and to *write* the processed data or signal into a file for future consumption.

2.2 Getting Prepared

The following list of commands will be used in this lab. Before you come to the lab, read the MATLAB documentation (and use MATLAB's `help` command) to learn about these commands.

```
plot; subplot; figure; hold; input; audioread; audiowrite;
audiorecorder; record; play; getaudiodata
```

In case you have an early version of MATLAB that does not support `audiorecorder` and related commands, search MATLAB and/or the web for the appropriate command.

2.3 Complex Numbers

This section will test your understanding of complex numbers when plotted as vectors.
For all parts in this section, assume that $z_1 = \sqrt{2}e^{j0.25\pi}$ and $z_2 = 3 + 4j$.

(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.

2.4 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)
```

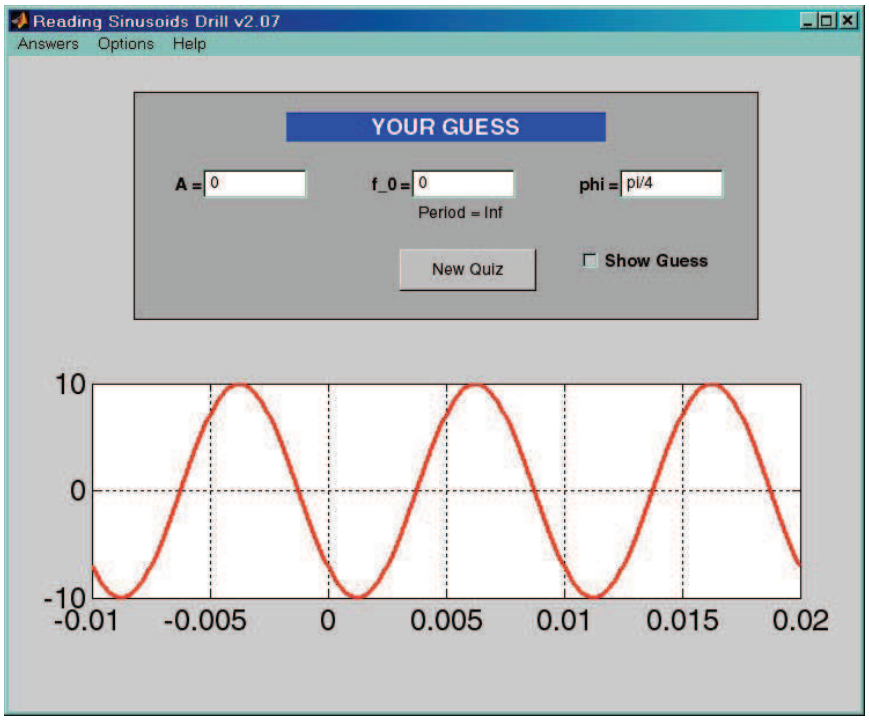
The `meshgrid` function generates all the pairs (u, v) for the domain.

2.6 GUI for `zdrill`

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.

2.7 GUI for `sindrill`

Demonstrate that you can use the `sindrill` GUI. Set the Level to "Pro" under the `Options` menu. Work a few problems; you can check your answers with the drop-down menu `Answers`.



2.8 Microphone and Headset/Loudspeaker

For this lab and many future labs, you will need a microphone and headset. Many laptops already come with these devices. If you need to acquire these devices, there are many inexpensive selections available from Internet vendors. For the work in these labs, you do not need to become an audiophile and buy expensive headphones. (Of course, if you have a passion towards audio and music, you may also use high-quality headphones and microphones that you already have.)

3 In-Lab Exercises

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

3.1 Interactive Input in MATLAB

MATLAB allows the user to provide input through interactive commands. This may be handy, particularly when you want the user to be able to specify some data values or strings in a function (.m file) during the so- called run-time. Here is a function example (take the cube root of) that involves a user input:

```
function a = crootof
x = input('what: ');
if x >= 0
a=(x)^(1/3);
else
'non-negative number only'
end
```

Then, in MATLAB, the following shows one application of the command crootof:

```
>> crootof
what: 287318
ans =
65.9864
```

where the number 287318 was given by the user. A user can use the input command to provide text strings to MATLAB as well. This is done by specifying a 's' argument in the command line:

```
>> ss= input('Text input: ','s')
Text input: once upon a time
ss =
once upon a time
```

Now, write a MATLAB code that asks the user to input a string, say `Mary had a little lamb`, and then prints out the characters in *reverse order,*, i.e., `bmal elttil a dah yraM`. Demonstrate your working function to the instructor; input your full name as the test case.

Completion of Lab Results (on separate Report Page)

3.2 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
dt = 1/800;
XX = rand(1,3).*exp(2i*pi*rand(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
```

Completion of Lab Results (on separate Report Page)

3.3 Generating Sinusoids and Decaying Sinusoids

An example for generating a sinusoid was included in Lab #0. Here, we will again use the built-in MATLAB editor to create a script file called `mySinusoid.m` containing the following lines for future use:

```
function xs = mySinusoid(amp, freq, pha, fs, tsta, tend)
% amp = amplitude
% freq = frequency in cycle per second
% pha = phase, time offset for the first peak
% fs = number of sample values per second
% tsta = starting time in sec
% tend = ending time in sec
tt = tsta : 1/fs : tend; % time indices for all the values
xs = amp * cos( freq*2*pi*tt + pha );
end
```

This function `mySinusoid` can be called once the argument values are specified. For example,

```
amp = 6;
freq = 80;
pha = pi/6;
fs = 8000;
tsta = 0;
tend = 3; %a 3-sec long signal
xs = mySinusoid(amp, freq, pha, fs, tsta, tend);
%<--- plot first three cycles of the generated sinusoid
ts = tsta:1/fs:tsta+3/freq;
Lt = length(ts);
plot( ts, xs(1:Lt), 'b-', ts, 2*xs(1:Lt), 'r--' ), grid on
title('TEST PLOT of TWO SINUSOIDS  (scaling by 2)')
xlabel('TIME (sec)')
```

You may want to try other numbers to appreciate the use of the function. In lecture, the three parameters of a sinusoid have been studied, along with many properties of sinusoids. Here we are most interested in the plotting tool which has many options (so you may want to read its documentation carefully).

Now, extend the previous example to make a decaying sinusoid, i.e.,

$$x(t) = Ae^{-bt}\cos(\omega t + \varphi) \tag{7}$$

Write a function called `myDecayingSinusoid(A, b, omega, phi, fs, tStart, tEnd)`

Then pick the correct numbers to generate and plot a decaying sinusoid `xDecay` that is 2 seconds long, with a frequency of 40 Hz, an amplitude of 10, a phase of $\pi/4$ rads, and a decay parameter $b = 0.8$. What happens to the plot if we increase the decaying rate to $b = 3$. Show the plotting result to the lab instructor.

Completion of Lab Results (on separate Report Page)

3.4 Reading WAV File into MATLAB and Playing an Audio Array

Many sound files are stored in `.wav` format; see `http://en.wikipedia.org/wiki/WAV` for information about this particular format. Other formats, such as `mp3`, are available. We will focus on `.wav` here. Use the MATLAB command `audioread` to read data in `wav` files into MATLAB data arrays. For example:

```
xx = audioread('myvoice.wav');
```

will load the data in the file `myvoice.wav` into the array `xx`. Another command `length(xx)` will tell you how many values have been read into `xx`. In many applications, you may need to know what is called the sampling rate, which is the number of sample values per second at which the data was acquired into the file. These parameters can be retrieved as part of the `audioread` result:

```
[xx, fs] = audioread('myvoice.wav');
```

For this exercise, you need to access some `wav` files. If you have your own inventory of `wav` files, you are welcome to use them here. (Make sure the file is long enough for the exercise; see below). If not, a file called `ece2026Lab01voice.wav` can be downloaded from canvas for your use. Learn to use the `audioread` command to read the data into an array, say `xx`. You can find out how long the signal is in seconds by reading the length of the array via `length(xx)` which gives you the total number of samples and then divide it by `fs`, the sampling rate in samples per second. Then, plot the sound wave `xx`, from $t = 0.25$ to $t = 0.5$. You need to know how to translate time into the index of the array; an example has already been shown above. Show the plotted result to the instructor.

Completion of Lab Results (on separate Report Page)

In MATLAB, an array of data can be played to produce audible sound, using:

```
soundsc(xx, fs);
% what is xx and fs??? Make sure you know what these are
```

Try the command yourself and listen to the data you just read in from the file `ece2026lab01voice.wav`.

3.5 Processing the Data and Writing the Result into a wav File

3.5.1 Time reversal

Following Section 3.3, you have an array `xx` that contains a signal or a sound. There are many simple operations that can be performed on `xx`. For example, scaling where we reduce the signal's amplitude by a half (called attenuation),

```
xh = xx * 0.5;
```

A more complicated operation would be to reverse the time axis of the signal and then plot it.

```
Lx = length(xDecay);
xDecayReversed = xDecay(??:??:??);  % figure out how to fill in those ??s
plot( ... xDecayReversed ... ??? )
```

In the plot of `xDecayReversed`, it should be easy to see the time reversal. Show the plot to the instructor for verification.

Completion of Lab Results (on separate Report Page)

If the signal were a sound file, then the output of the time-reversal could be written out to a new `wav` file:

```
Lxh = length(xh);
xhReversed = xh(??:??:??);  % figure out how to fill in those ??s
audiowrite( 'ECE2026lab01outRev.wav' , xhReversed , fs );
```

If we listen to `xhReversed`, the sound will be played backwards. Note that the argument `fs` can be copied from the `audioread` result. You can use your own number, such as reducing it to half or doubling it to twice the original value. Using `soundsc`, play the output `wav` file `ECE2026lab01outRev.wav`. You can even combine two operations to have the sound file be played backwards with every other samples skipped.

3.6 Recording and Playing Sounds in MATLAB

You can also record your own sound with MATLAB. To record data from an audio input device (such as a microphone connected to your system) for processing in MATLAB, follow the sequence below:

- Create an audiorecorder object:

```
>> audiobject = audiorecorder(8000, 16, 1);
% an object here is like a device or channel
% fs = 8000; use 8000 values for each second of sound
% nbit = 16; use 2 bytes to represent a value
% nchannel = 1; use 2 for stereo recording...
% (need stereo microphone)
% you may try other numbers
```

- Call record or recordblocking:

```
>> record(audiobject)
>> stop(audiobject)
% record opens a channel that links to audiobject
% stop closes the channel; if you do not close the channel..
% recording continues and audiobject cannot be accessed
```

  or

```
>> recordblocking(audiobject, 5);
% record a fixed duration of 5 seconds into audiobject
```

- Create a numeric array corresponding to the signal data using the getaudiodata:

```
>> xx = getaudiodata(audiobject);
```

  For example, connect a microphone to your system and record your voice for 5 seconds. Capture the numeric signal data and create a plot (from Mathworks):

```
% Record your voice for 5 seconds.
recObj = audiorecorder(8000, 16, 1);
disp('Start speaking.')
recordblocking(recObj, 5);
disp('End of Recording.');
% Play back the recording.
play(recObj);
% Store data in double-precision array.
myRecording = getaudiodata(recObj);
% Plot the samples.
plot(myRecording);
```

  In the above, the array `myRecording` contains the voice data, which can be processed. The recording (or processed results) can be written into a `wav` file using the aforementioned `audiowrite` command.

$\Longrightarrow$ For Self-practising, you should record a vowel sound containing some quasi-periodic waveform section with higher amplitude than neighboring consonant sections. Measure the pitch period, which is the duration of a period in the vowel sound. The inverse of it is called pitch which is a vibrating frequency of the speaker's glottis when pronouncing the vowel. We will come back to this issue later in Lab #3.

Lab #1
ECE 2026 Fall 2023
LAB COMPLETION REPORT

Name: _____ Date of Lab: _____

Part 3.1 Write the string reversal output below and show it to the instructor.

Part 3.2 Replace the inner `for` loop with only one or two lines of vectorized MATLAB code. Write the MATLAB code in the space below:

Part 3.3 Show the plot of a decaying sinusoid.

Part 3.4 Read in a voice file and plot a section. Locate a vowel region containing a quasi-periodic waveform with higher amplitude than neighboring consonant sections. Measure the pitch period, which is the duration of a period in the vowel sound. The inverse of it is called pitch which is a vibrating frequency of a speaker's glottis when pronouncing the vowel. We will come back to this issue later in Lab #3.

Part 3.5.1 Show the plot of a time-reversed decaying sinusoid.