

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.

1.1 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.

1.2 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 already have high-quality headphones and microphones. Use them.)

1.3 Getting Prepared

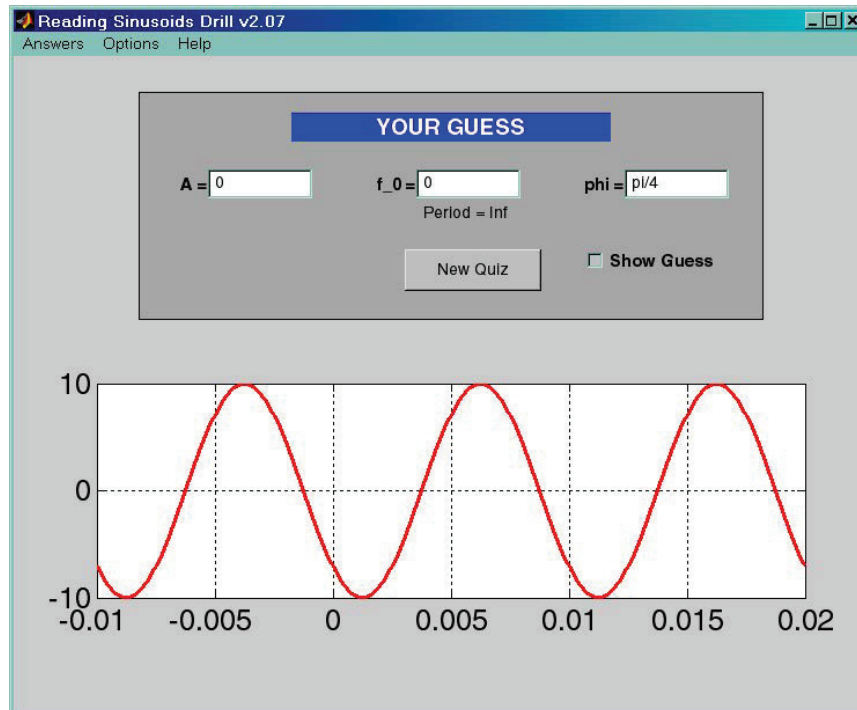
The following list of commands will be used in this lab. Before you come to the lab, read the MATLAB documentation (and use MATLAB’s `help` command) to learn about these commands.

```
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.

1.4 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 Exercise

For the instructor verification, you will have to demonstrate that you understand things in a given subsection by answering questions from your lab instructor . 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.

2.1 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.

Instructor Verification (separate page)

2.2 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 = 7;
freq = 70;
pha = -pi/3;
fs = 8000;
tsta = 0;
tend = 2; %a 2-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.,

$$x(t) = Ae^{-bt} \cos(\omega t + \varphi) \quad (1)$$

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 3 seconds long, with a frequency of 60 Hz, an amplitude of 6, a phase of $\pi/3$ rads, and a decay parameter $b = 0.9$. What happens to the plot if we increase the decaying rate to $b = 4$. Show the plotting result to the lab instructor.

Instructor Verification (separate page)

2.3 Reading WAV File into MATLAB and Playing an Array

Many sound files are stored in .wav format; see <http://en.wikipedia.org/wiki/WAV> for more information about this particular format. Other formats are available, such as mp3; we will focus on .wav in this lab. The MATLAB command `audioread` is the one to use to read data in wav files into MATLAB data arrays. For example:

```
xx = audioread('ece2026Lab01voice.wav');
```

will load the data in the file `ece2026Lab01voice.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('ece2026Lab01voice.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 Website 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.

Instructor Verification (separate 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`.

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

2.4.1 Time reversal

Following Section 2.2, 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.

Instructor Verification (separate 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. But you can experiment with 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.