EGR 111
Audio Processing

This lab shows how to load, play, and create sounds and music with MATLAB.

Resources (available on course website): speech1.wav

New MATLAB commands: `audioread, sound, soundsc`

1. Download the Files

Download the file EGR111_Files.zip by going to the course website, right-clicking on EGR111_Files.zip, and clicking on "Save Link As…" or "Save Target As…" Then save the file in your P:\MATLAB folder. Extract the files by going to your P:\MATLAB folder, double-clicking on EGR111_Files.zip, click on Extract all Files, change the folder to P:\MATLAB, and click Extract. This process should place Solar_Panel_Data.xlsx and several other files that we will use in later labs into your P:\MATLAB folder. Check to make sure that these files are in your P:\MATLAB folder before proceeding with the lab.

2. Load and Play an Audio Signal

In the previous lab we downloaded the EGR111 files (which included speech1.wav) and saved them in your P:\MATLAB folder. Change the Current Folder in MATLAB to P:\MATLAB. The MATLAB command `audioread` can be used to load .wav files into MATLAB as follows:

```
[x,fs] = audioread('speech1.wav');
```

If you get an error message that says something like "Error using audioread. The filename specified was not found in the MATLAB path.", check to make sure that you have saved the file speech1.wav in P:\MATLAB and that you have changed the Current Folder to P:\MATLAB.

When a function returns two or more variables, we list the output variables in square brackets. The `audioread` command above places the audio signal from the file 'speech1.wav' into a vector called `x` and puts the sampling rate (in Hz) into a variable called `fs`. The sampling rate is the number of times that the original signal was measured per second, and this value is needed when we play the signal to the speakers.

We can use the `sound` function to play the recording through the speakers as follows:

```
sound(x,fs)
```

If you play the sound back at a higher sampling rate, it increases the pitch. Type the following:

```
sound(x,2*fs)
```

Try playing the sound at a lower sampling rate to see how it sounds.

3. Reversing a Vector

In this section, we will learn how to reverse the order of the elements in a vector, and then we will reverse the order of a sound vector in order to play it backwards.

As an example, generate the following vector:

```
>> v = [10 20 30 40]
```

We can copy the elements of the vector `v` and store them in the variable `vc` as follows:

```
>> vc = v(1:end)
vc =
    10    20    30    40
```

Note that when the keyword `end` is used to index a vector, MATLAB replaces it with the length of the vector, so the command above is equivalent to `vc = v([1 2 3 4])`.

If we reverse the order of the index values, then MATLAB will copy the elements in reverse order as follows:

```
>> vr = v(end:-1:1)
vr =
    40    30    20    10
```

When the expression `end:-1:1` is used to index a vector, it generates a vector starting with the length of the vector, with an increment of -1, and ending with the value 1. So `vr = v(end:-1:1)` is equivalent to `vr = v([4 3 2 1])`, which copies the elements of the vector `v` in reverse order.

Exercise 1: Generate the vector x = 10:10:100, and then copy the elements of x in reverse the order and store the result in a vector called xr.

Exercise 2: Write a script file called AudioEx2.m that clears the workspace, loads the sound file `speech1.wav`, reverses the vector, and plays the reversed vector through the speakers. **Use a semi-colon at the end of the statements to suppress unnecessary printing.** Also plot the original data and the reversed data.

Checkpoint 1: Show your instructor your Exercise 2 script file and plots.

4. Generating Tones

Consider the signal $x(t) = Acos(2\pi f_0 t)$ where A is the peak amplitude and f_0 is the frequency of the cosine in Hz. Cosine is a periodic signal with a period of $T = \frac{1}{f_0}$ seconds, so it repeats every T seconds.

The MATLAB commands below create a cosine with a peak amplitude of 0.5 and a frequency of 500 Hz. The cosine is then plotted, and played through the speakers. The frequency of the cosine (500 Hz) determines the pitch, and the peak amplitude determines the loudness of the tone. The sampling rate (`fs`) is the number of times that the cosine is sampled (or evaluated) per second.

```
clear                    % clear variables
fo = 500;                % frequency of the cosine (in Hz)
A = 0.5                  % peak amplitude
fs = 44100;              % sampling frequency (in Hz)
Ts = 1/fs;               % sampling interval (in Sec)
t = 0:Ts:2;              % sample every Ts seconds for 2 sec
x = A*cos(2*pi*fo*t);    % generate cosine
plot(t,x)                % plot cosine
sound(x,fs)              % play through speakers (no scaling)
```

Exercise 3:

a. Open the editor window (click on File, New, Script) and type the commands above into a MATLAB script file and name the file AudioEx3.m.

b. Run the program by either clicking the run button near the top of the editor, or by typing the filename without ".m" into the command window: `AudioEx3`

c. The plot looks like a big blue screen because there are 1000 cycles squeezed into the graph. Use the zoom tool in the figure to zoom in on just one or two periods (see next page).

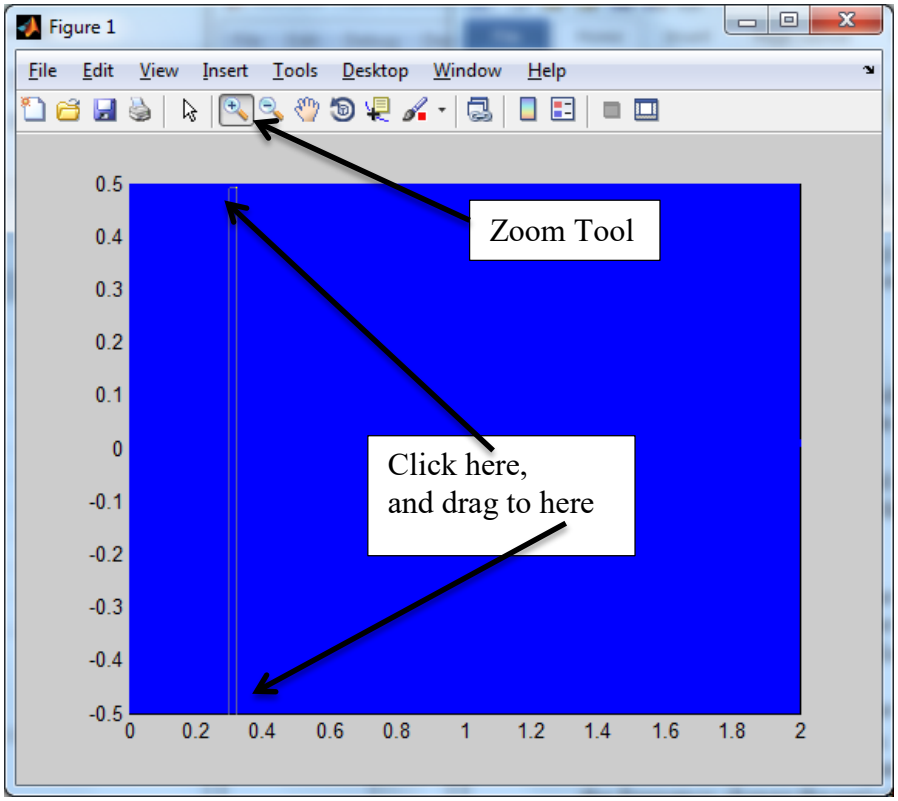
Figure 1
File Edit View Insert Tools Desktop Window Help
Zoom Tool
Click here,
and drag to here
0.5
0.4
0.3
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0 0.2 0.4 0.6 0.8 1 1.2 1.4 1.6 1.8 2

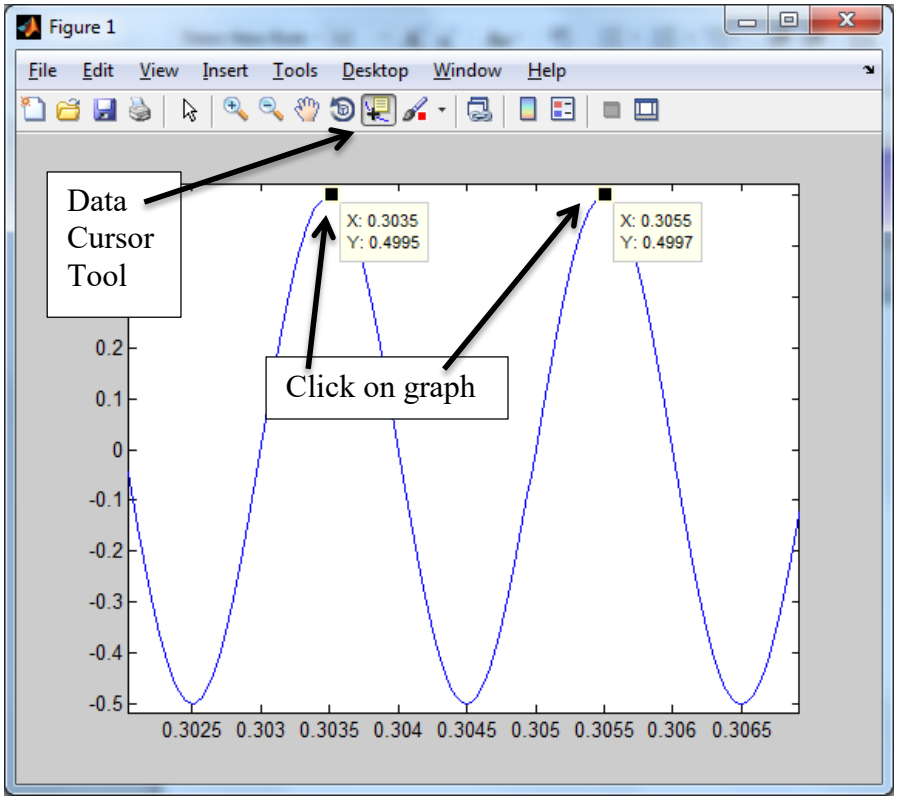
Figure 1
File Edit View Insert Tools Desktop Window Help
Data
Cursor
Tool
X: 0.3035
Y: 0.4995
X: 0.3055
Y: 0.4997
Click on graph
0.2
0.1
0
-0.1
-0.2
-0.3
-0.4
-0.5
0.3025 0.303 0.3035 0.304 0.3045 0.305 0.3055 0.306 0.3065

d. Use the Data Cursor to measure the period (see previous page). Compare the measured period to the theoretical period. Are they close?

e. Edit the script file to change the frequency of the cosine (fo) and run the program again. How does changing the frequency change how the tone sounds?

f. Change the peak amplitude (A). How does changing the amplitude change how the tone sounds? (The maximum amplitude that the sound card can handle is 1.0, so any values that are bigger than 1 are clipped before the signal is sent to the soundcard, and so the sound may be distorted. There is also a command that scales the signal to prevent clipping and distortion called `soundsc`.)

5. Synthesizing Music

The pitch of a musical note is determined by the frequency of the signal, with higher pitch corresponding to higher frequency.

Look at the following webpage to see the list of frequencies and how they correspond to piano notes.

http://en.wikipedia.org/wiki/Piano_key_frequencies

Note that the frequency of the A above middle C (which is called A4) is 440 Hz. What is the frequency of the next higher A (which is called A5)? What is the frequency of the A below middle C (which is called A3)? How does the frequency change between each octave?

Let's make the A note above middle C (i.e. A4) sound for .5 seconds. Modify your script file to make the frequency of the sinusoid be 440 Hz, run the file, and listen to the sound.

Now modify the file to create the tone for the A an octave below middle C (A3).

In order to play music we need to generate several notes and concatenate them together. We can concatenate several row vectors together by enclosing them in square brackets as follows:

```
a = [10 20 30]
b = [40 50]
c = [60 70 80]
d = [a, b, c]
```

If we want silence in between the notes, we can use the zero command to generate a vector of zeros and place this vector between the notes. The following command would generate a column vector with 10 zeros in it:

```
e = zeros(1,10)
```

The following commands generates the notes A, B, and C with 0.1 seconds of silence between each note.

```
fs = 44100;        % sampling frequency (in Hz)
Ts = 1/fs;         % sampling interval (in seconds)
t = 0:Ts:1;        % sample every Ts seconds for 1 second
A = 0.5*cos(2*pi*440*t);         % generate note A
B = 0.5*cos(2*pi*493.883*t);     % generate note B
C = 0.5*cos(2*pi*523.251*t);     % generate note C
s = zeros(1,fs*0.1);  % generate 0.1 second silence
y = [A, s, B, s, C];  % concatenate notes
soundsc(y, fs)         % send to speakers (with autoscaling)
```

Exercise 4: Create a script file named AudioEx4.m to generate your own short piece of music that has at least 4 different notes and 3 rests (silent sections). You can re-create an existing song or make an original composition. The sheet music for the first few notes of Fur Elise are on the course website, in case you would like to synthesize that song. There is a tutorial on reading music at http://www.wikihow.com/Read-Music.

Checkpoint 2: Show the instructor your commands and the results for Exercise 4.