ESE 150 – Lab 04: The Discrete Fourier Transform (DFT)

LAB 04

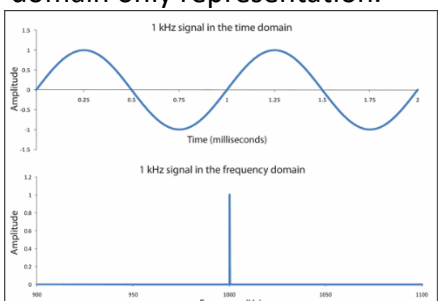
In this lab we will do the following:

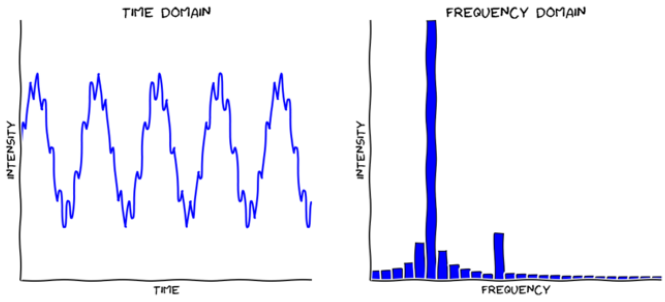
1. Use Matlab to perform the Fourier Transform on sampled data in the time domain, converting it to the frequency domain
2. Add two sine waves together of different frequency.

Background:

In lecture we studied the Fourier Series and the Fourier Transform. The Fourier series is a representation of a periodic function using a summation of sines and cosines. It offers a way to represent a time-based periodic signal in the frequency domain. The Fourier Transform is an extension of the Fourier Series that allows us to represent non-periodic functions using a summation of complex sinusoids. It allows us to take signals in the "time domain" and see their breakdown or "frequency domain" components. The Discrete Fourier Transform (DFT) is a variation of the Fourier Transform that applies when our function is discrete. This version of the Fourier Transform becomes very useful in computer engineering, where we have "digitized" incoming analog signals, taking them from a continuous form to a discrete form. In our case, we've sampled "music" using our Arduino A2Ds and as a result, we have a set of discrete sampled data. In this lab, we'll apply the DFT to our discrete sampled data to transform it from the time domain to the frequency domain and look more carefully at its frequency components.

In previous labs we've only sampled simple waveforms: sine wave, square wave, triangle wave. We could "convert" them to the frequency domain without doing the Fourier transform, they simply have one frequency, so only one sine wave could represent them. We need to create a more interesting waveform. So we'll begin the lab by using a sound program to produce separate sine-wave. Then we'll combine them together. Next we'll sample this data, import it into Matlab and apply the discrete Fourier Transform to it, so we can see the frequency components of our sampled signal, as opposed to its time-domain only representation.





These two graphs compare the frequency domain and time domain. On the left, you can see a pure sine wave will have one spike in the frequency domain, where its height is the amplitude of the wave and the frequency is just the frequency of the sine wave. On the right, you can see a sound comprised of many sine waves, which will have many spikes of different frequencies with their heights corresponding to how prevalent that frequency is in the time domain (aka the amplitude of that frequency).

ESE 150 – Lab 04: The Discrete Fourier Transform (DFT)

Prelab: Applying the Discrete Fourier Transform in Matlab

- In this section, we'll take time sample data and convert it from the time domain to the frequency domain using the DFT
- We'll use a built-in function in Matlab to help us apply the DFT, called FFT()

1. If you take ESE224, you will implement this formula in MATLAB by hand. However, MATLAB provides an implementation of this formula, so you don't have to worry about it for this class! (This is one of the reasons why many people use MATLAB). The implementation is called a FFT, or Fast Fourier Transform, because of the efficient algorithm for computation. You can read about the FFT in MATLAB here: http://www.mathworks.com/help/matlab/ref/fft.html

2. The basic idea is that it takes in N samples from the time domain and determines the sine/cosine components at various frequencies.

3. Begin by importing the following sine wave data into Matlab as you did in Lab 3. Use the 5000Hz sampled data in this document to fill the matrix: https://docs.google.com/spreadsheets/d/1sgrckPswtGqRn_D5-sQy7QXVlpzElWfK90TxpxZ7f34/edit?usp=sharing]

Importing Sampled Data into Matlab:

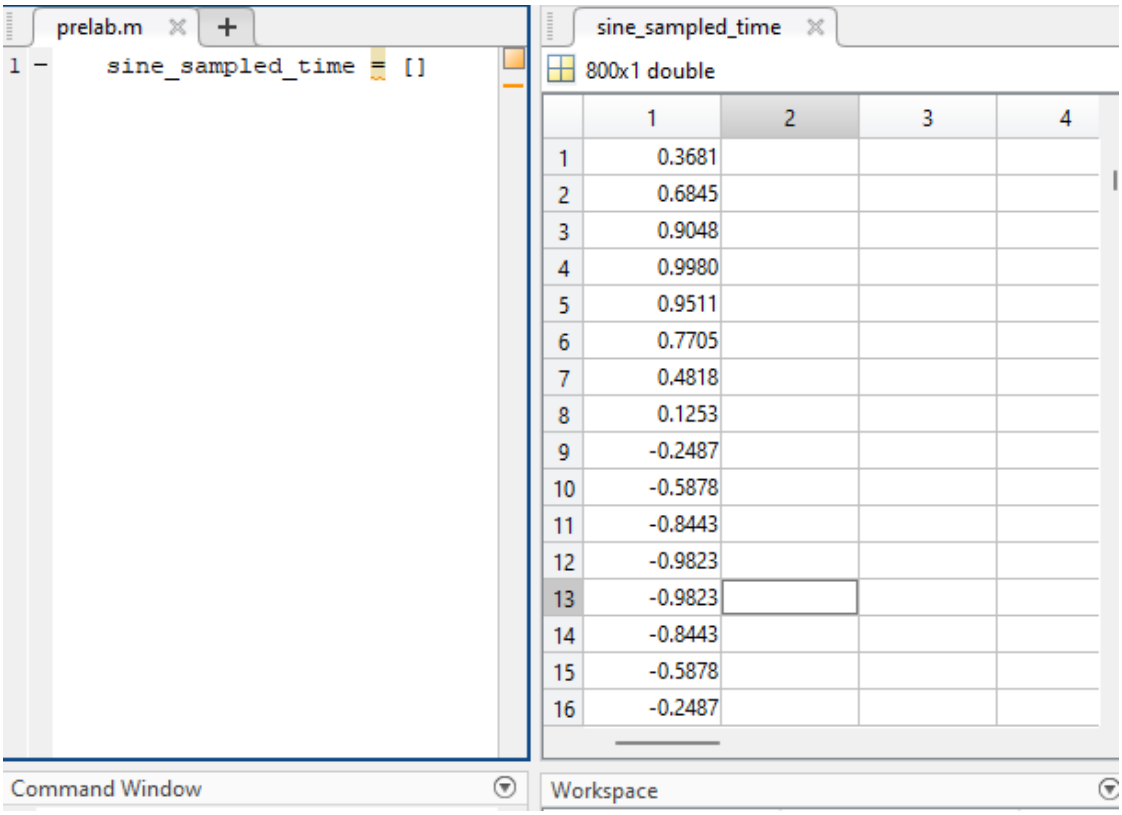
1. From the sampled data given to you, highlight the samples for the wave, and "copy" them (using <ctrl> C).

2. In Matlab create an empty matrix, named "**sine_sampled_time**" by typing the following:

Sine_sampled_time = []

3. In the "Workspace" pane, click on your newly created *sine_sampled_time* matrix

    a. This will bring up an editor, that looks a little like a spreadsheet.



	1	2	3	4
1	0.3681			
2	0.6845			
3	0.9048			
4	0.9980			
5	0.9511			
6	0.7705			
7	0.4818			
8	0.1253			
9	-0.2487			
10	-0.5878			
11	-0.8443			
12	-0.9823			
13	-0.9823			
14	-0.8443			
15	-0.5878			
16	-0.2487			

4. Paste your spreadsheet data into Matlab:

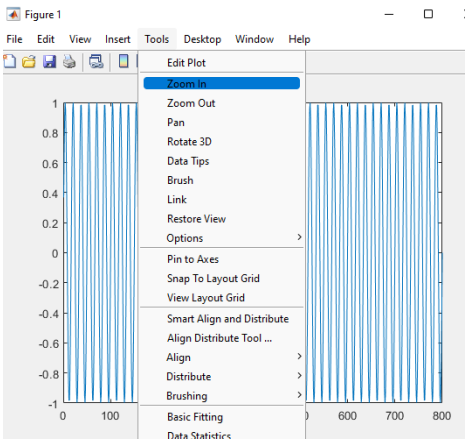
    a. RIGHT click on the very first square at the top left, row 1, column 1.

    b. Choose “Paste Excel Data”; once pasted close the variable editor.

c. Notice in the “workspace” pane, *sine_sampled_time* is now an 800x1 dimension matrix.

4. Now, plot the data against the appropriate time axis (you must turn in this plot), **label all axis (voltage vs. time and the units, and title)** - *zoom in so we can see 4 cycles.*

To zoom, go to Tools -> Zoom In and then, select the first 4 cycles



5. Now, convert the sine wave from the time domain to the frequency domain using Matlab's built in DFT by typing the following (*Make the sine data you are converting is in Volts!)*

```
sine_sampled_frequency = fft(sine_sampled_time)
```

What you will notice is 800 complex #'s are produced (see the real # + imaginary #'s).

Why complex #'s? Recall Euler's identity, where $e^{i\vartheta}=\cos\vartheta+i\sin\vartheta$ ∎

6. Now, let's prepare to plot the converted data in the frequency domain. Type the following:

```
samp_period = 0.0002+0.000125    % sampling period + analogRead()'s delay
samp_freq = 1/samp_period        % sampling frequency
samples = 800                    % # of samples

sine_sampled_frequency = abs(sine_sampled_frequency / samples)
```

- Notice in the above code, that our sampling period is not just 200uS; there is a delay associated with analogRead(). So our sampling frequency is actually a little less than 5000 Hz.
- ***What is that actual sampling frequency when you include this additional delay?***
- Also, notice the last line; it takes the "absolute" value of our frequency data. When one takes the absolute value of a complex #, we get its magnitude (like polar magnitude).

7. Let's plot the data…
   a. Recall, there are 800 elements in the sine_sampled_frequency matrix. These represent the SPIKES or magnitudes of the sine-waves at various frequencies.
   b. But what are the x-axis values? They will now be frequency!
   c. What is our range? Let's say 0 Hz to start with, but what about the upper bound? Use the actual sampling frequency you determined in Step 6.
   d. Create a vector called: freqs = (0:799).
   e. Scale it so that the highest frequency is "samp_freq" (again, determined in Step 6).
   f. Plot your data by using `plot(freqs, sine_sampled_frequency)`
8. Interpreting the plot…
   a. You will see two spikes, one at approximately 300 Hz, and one ~2800Hz. Why is the high one false? [**Hint:** What's your actual sampling frequency from Step 9? What do we know about the relationship between sampling frequency and the frequencies we can accurately represent without aliasing?]
   b. Change your "sine_sampled_frequency" matrix and cut off false frequencies.
      i. Create a shorter vector that only includes the true frequencies based on your sampling rate. **Hint:** it is easy to do this in MATLAB with index range selection

1. https://www.mathworks.com/company/newsletters/articles/matrix-indexing-in-matlab.html

ii. Note that both `sine_sampled_frequency` and `freqs` should be shortened to only include the valid frequencies.

c. Also, notice the amplitude is ½ of what it should be? Double the amplitudes in the sine_sampled_frequency matrix; this is a byproduct of the absolute value function.

d. Lastly, replot your data…does it line up with what you expected?

i. You could adjust that value of "+0.000125" to figure out analogRead()'s exact delay.

e. Make sure there is a title and axis labels with units.

9. Make your own Matlab FUNCTIONs plot_time() and plot_dft() that takes in a vector of time domain samples and plots them in the time domain (subtracting the offset voltage offset appropriately) and in the frequency domain, respectively — that is, automate what you did in steps 7 and 8.

a. This is the Matlab code that must be turned in for this section.

b. You will also need to use this function in lab.

c. Your functions should work for any size time domain samples vector, not just 800. Use the MATLAB length function to determine the number of samples in the input vector.

d. Your two functions should take in arguments as specified below:

i. plot_time(***time domain samples, figure number, start sample, end sample, sampling frequency***)

Specifically, ***start sample*** and ***end sample*** let you plot within a time range, so that you don't have to zoom in manually as you did in Step 5. But you may need to do some calculations according to the sampling frequency to pick the correct start sample and end sample with which to call plot_time.

ii. plot_dft(***time domain samples, figure number, sampling frequency***)

This function should generate the frequency domain plot.

Note: ***figure number*** should be used as "figure(***figure number***)" when you open up a figure window.

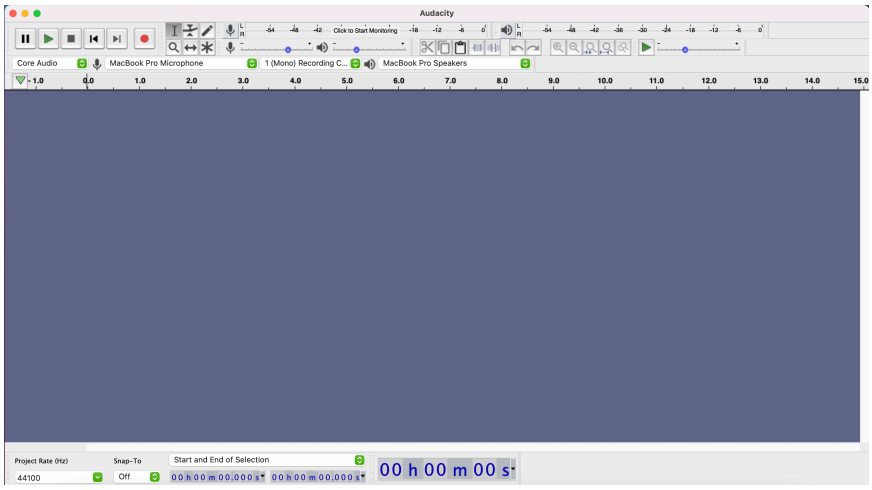
Check that your functions work correctly by using them to reproduce the graphs from steps 7 and 8.

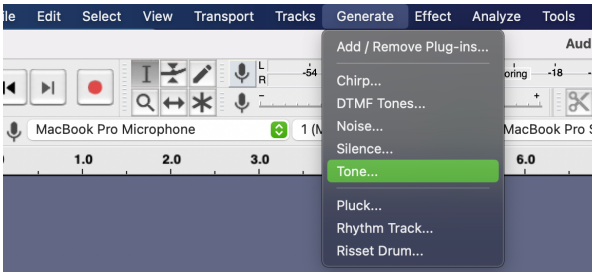
Lab Procedure:

Lab – Section 1: Adding two sine waves together using Audacity

- In this section, you'll add two sine waves together in Audacity and observe their combined sound

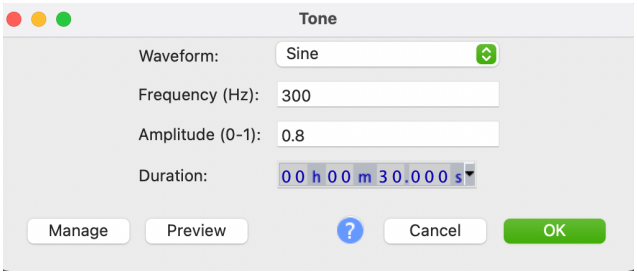
1. [if we're not in Detkin for this lab, or you want to run this on your laptop] Download Audacity at: https://www.audacityteam.org/download/
2. Start Audacity.
   It should look like this:





3. Now to add our waves. Find and click on the Generate tab and then click on Tone…

4. Input the following parameters and press OK:

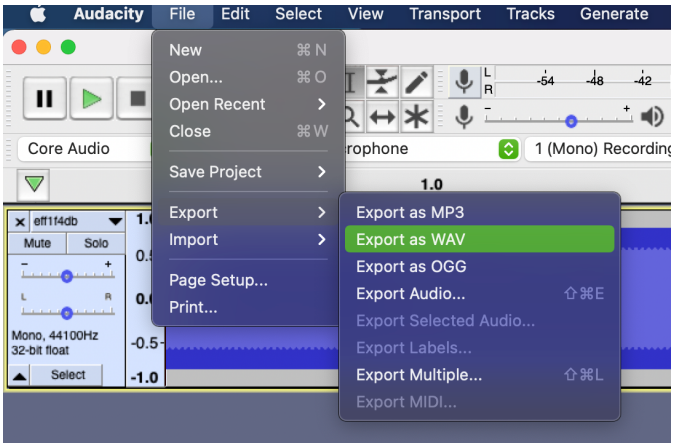


5. To play the tone, click on the green triangle at the top left of the screen.

6. Now mute the first track.

7. Click outside of the first track, so that the track is greyed out. Add another wave (repeat Steps 3 – 5), except with a 600 Hz sine wave. Observe the difference in sound from the 300 Hz wave.
8. Now unmute the first track and press play so they both play at the same time. What do you hear? How does this sound differ from the first two?

9. Now export the audio as a .wav file. To do this, go to File > Export > Export as WAV



Caption

10. Change the name to whatever you like (e.g. addedWaves.wav) and keep track of where you download it to (e.g. your S: drive on eniac or Desktop or Documents or Downloads or…). Don't change anything else.

Lab – Section 2: Importing to MATLAB, and Plotting in the Time Domain

- Import the data into MATLAB

1. Import the samples into Matlab:
    a. To import your samples, simply run the following code:
       [samples_time, samp_freq] = audioread('filename.wav')
    b. Make sure you're in the right folder (a.k.a. where you saved the .wav file to).
    c. It might be helpful to copy and paste .wav file into the same folder as your plot_time() and plot_dft() scripts so as to avoid going back and forth through folders.
    d. Use your Matlab functions: plot_time() and plot_dft() to plot the signal in time and in frequency.
        i. note that the samp_freq will be different from the frequency you used in prelab. Adjust your samp_period accordingly.
        ii. In your plot_dft(), you should have two tall clear spikes at approximately 300Hz and 600Hz as expected. However, you may notice some much smaller spikes at other frequencies (because the audacity waveforms aren't clean). So don't worry if you see these!
    e. Properly label and turn in the plots.
2. Download the three "unknown/mystery" signals from the link on the syllabus and import the 3 different sets of samples into Matlab.
3. Create plots of the provided data in the time domain, just like you have done with the other sampled data in labs:
    - Note, these samples are already scaled to voltage values
    - the sampling period for this data was 0.00002s
    - please plot only the first 200 samples
    - make sure to label axes and title your plot
4. Create plots of the provided data in the frequency domain. Use plot_dft() developed in prelab:
    - change the sampling period to 0.00002s
    - the length of the provided data is no longer 800. If you correctly coded your plot_time() and plot_dft() functions, they should work for the longer data. If not, this is a chance to test and refine them so that they do.
    - change the title for each mystery file.
5. Write down the functions that sum up to make each mystery wave. They will all be in the form:
   $A\sin(2\pi ft)$
    o You should find A and f from the plots created in step 4.
        ▪ Don't include any frequencies in the mystery wave with an amplitude lower than 0.2 (we will consider this noise rather than part of the wave).
    o Report the A and f values for the functions that make up each of the mystery waves.

6. Turn in all plots created **and** your resulting functions.
7. Show your time and frequency plots to your TA and answer a few questions. This is the Lab Exit Check-off.
8. Make sure both partners have access to the data collected before leaving lab. We recommend setting up a shared folder on Google drive.

ESE 150 – Lab 04: The Discrete Fourier Transform (DFT)

Postlab

Perform a dot product in MATLAB to extract specific frequencies.

While we used the FFT in MATLAB, it is effectively computing the dot product between each of the frequencies and the sampled data. To see this, do the following:

a. Create a vector that contains the time-sample coefficients for a 300 Hz sine sampled at the sample frequencies identified in Step 6 of the prelab (possibly refined in Step 8d.i).

i. Use your equation developed from Lab 2 prelab to create this vector in Matlab.
ii. To do this, first create a vector t1 = (0:799).
iii. Then, create a t2 by scaling this vector by the sample period you calculated in step 6/8d.i of the prelab.
iv. To apply a function to the elements of t2, you can perform an operation like $y = sin(5 * t2)$.This gives you y, the vector with time-sample coefficients.

- Try this out with the function given: $y = sin(5 * t2)$
- Now replace the arguments to sine with the appropriate ones to describe a 300Hz sine wave.

e. Perform a dot product between that vector and the sine_sampled_time vector.
   i. To perform dot product of two vectors v1 and v2, use **dot(v1, v2)**.
      1. Example: `dot(y, sine_sampled_time)`
   ii. Remember a dot product is an operation between two vectors of equal length that multiplies corresponding entries and takes the sum of the result

$$result = \sum_{i=0}^{L-1} a[i] \times b[i]$$

   iii. What result do you get (remember the dot product is a scalar!)?

[Note: range can be high – properly this should be normalized by a factor of (2/N), where N is the number of samples.]

f. Repeat a and b for a 300Hz cosine.
g. Repeat a and b for a 200Hz sine and cosine.
h. Repeat a and b for a 400Hz sine and cosine.
i. Report all results and relate to the input signal and FFT plot. Do you see a correlation between the magnitude of each of your results and the frequency of the sine wave?

ESE 150 – Lab 04: The Discrete Fourier Transform (DFT)

HOW TO TURN IN THE LAB

- Upload a PDF document to canvas containing:
  - **All Plots with axis and labels and titles!**
  - All Matlab code – just the functions you created in prelab are sufficient
  - Answers to all questions in the lab
  - Answers to postlab