# SUSTECH ME 425: Sensing Technology

**Sound Lab Procedure**

# Introduction

This is our first lab exercise to introduce you to ME 425 essential knowledge and skills. In this lab you will explore how Fourier analysis allows you to determine important signal properties, and then learn about the frequency resolution of the FFT spectrum. You will also learn about the signal-to-noise ratio (SNR), which helps you obtain useful data. The final topic covered in this lab takes you out of the frequency domain and back into the time domain, where you will learn how auto-correlation techniques can give you information about the repeatability of a signal, information that is not available to you through frequency domain techniques such as Fourier analysis. You will then use a cross-correlation function to make a simple measurement of the speed of sound with two microphones.

Finally, in the postlab you will learn (1) how to make a histogram and convert it to a probability density function and determine if the signal is Gaussian or not, and (2) determine the uncertainty on your speed of sound measurements.

**Essential Knowledge and Skills in this Lab:**

**Signal analysis:**

* Identify system resonance using Fourier analysis
* Integrate a power spectrum obtained from Fourier analysis to determine the power distribution in a signal as a function of frequency

**Experimentation and data collection**: Vary experimental parameters as needed to produce useful data

* Recognize useful vs. useless measured signals and if necessary resolve issues such as inadequate signal to noise ratio and resolution limitations.

**Data analysis:** Use filtering to remove noise or signal at unwanted frequencies, e.g. to increase the signal to noise ratio.

**Signal analysis and system identification:**

* probability density and distribution functions
* auto- and cross-correlation functions

**Communication:** Communicate results through visual communication in a style and format suitable for publication in a refereed journal or presentation at a conference in the field.

* Present data in a graphical form acceptable for publication in a peer-reviewed journal or presentation at a professional conference in the field.

# Using Fourier Analysis to Characterize Sound

This lab has three parts, first, in this section, continuing your study of using Fourier analysis to characterize sound. You will then explore important concepts in taking useful data of signal-to- noise ration (SNR) and Fourier transform resolution. Finally, you will use techniques in the time domain to analyze signals via auto-correlation and then measure the speed of sound via cross-

correlation. You will perform additional data analysis tasks in the postlab using Matlab.

## Lab Setup

1. Logon to the ME425 Lab Computer with Arduino IDE and Matlab installed (or use your own Laptop if you have both software)
2. Getting familiar with Arduino UNO and Arduino IDE. To read the values of an analog input, use the pins labeled A0 to A5 with function: analogRead(). The physical input you measure from the board is a voltage signal between 0 and 5V. To convert the number (in the range of 0-1023) you get back from analogRead(), you can simply divide it by 204.6. Here is how you would do it in the Arduino program:

Int raw = analogRead(A0);

Float volts = raw / 204.6;

Serial.println(volts);

In the world of Arduino, programs are called ***sketches***, and the file menu of the Arduino IDE allows you to Open and Save sketches as you would do for a document in a word Processor. The File menu also has an Example sub-menu from which you can load Arduino’s built-in example sketches.

1. Connect one of the microphones (such as Sparkfun BOB -09964) to Arduino UNO following the instruction on Make: Sensors Chapter 11, Figure 11-2. Access the data taken from the microphone. **Record** in your lab notebook the part numbers of your microphone and the Arduino UNO board.
2. One of you should sing a tone across the microphone – do not sing straight into it or your beautiful tone will be overwhelmed by the high flow of air from your mouth into the microphone. Keep the mic about 25 mm from your mouth (please remember that ME 425 is a fully metric course!). Once you have established your tone, use the program to collect 1 second of data at serial rate 115200.
3. Save the waveform data from IDE into a proper txt file so you can import it using Matlab.

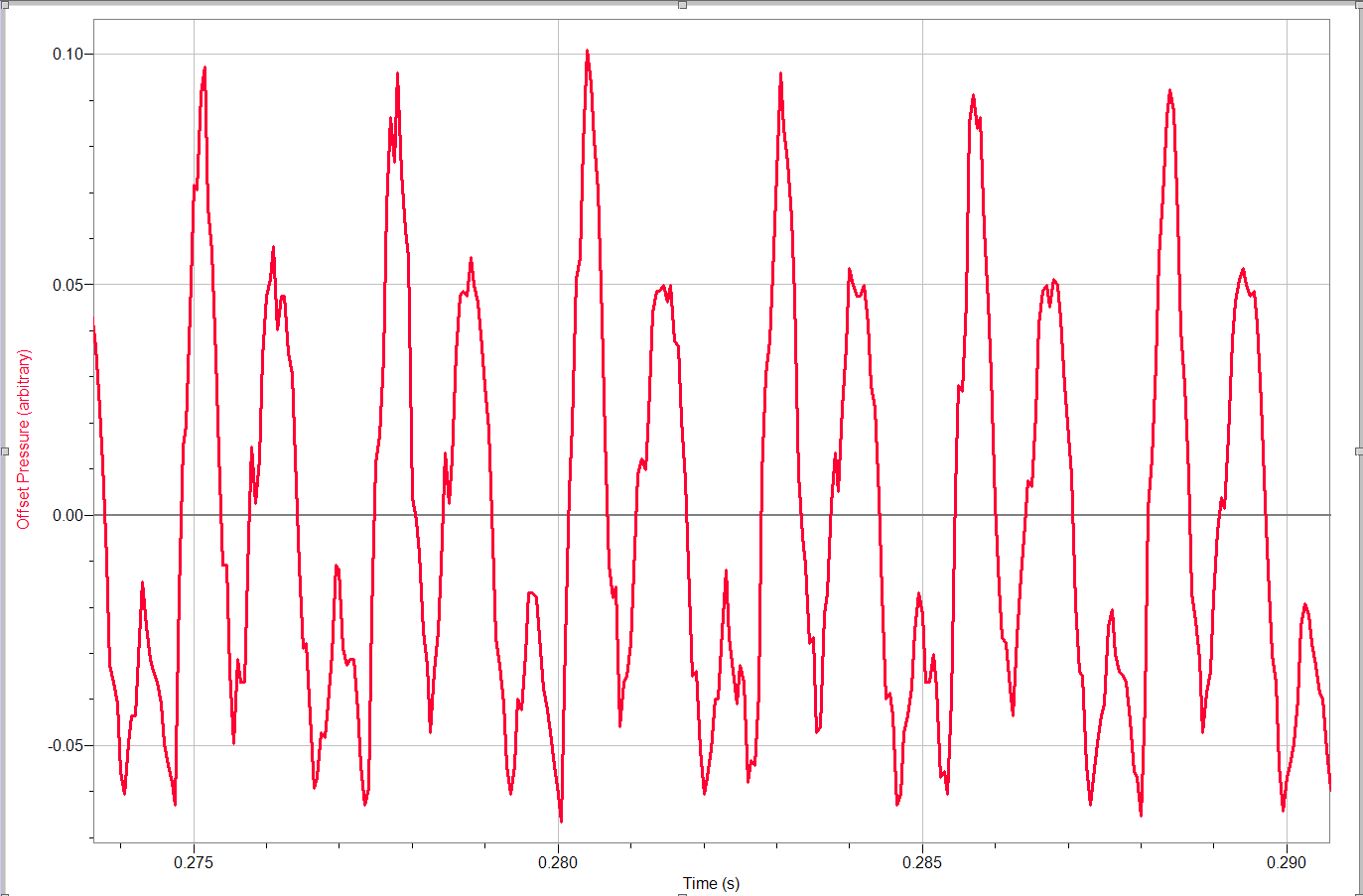
## Resonance

1. **Open Matlab and Measure characteristics** of the waveform you recorded. Write down the Mean value, Std Dev, #Samples, and Max and Min values in your lab notebook.

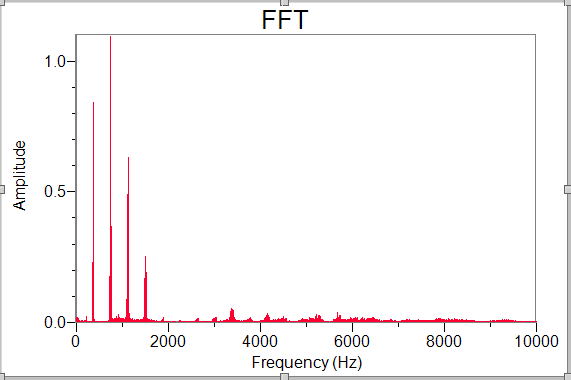
Sound is a pressure wave, which is carried by variations in pressure around atmospheric pressure, i.e. a small variation sitting on top of a large DC (constant value) offset. You should see a varying waveform. What is relevant when characterizing any sound signal is not the DC (mean) value, but the variation around that value.

**IMPORTANT:** You must **never** blindly take data! You must always examine your data to see if it looks reasonable, before wasting time taking more data. Can you tell the structure of the signal from these data with this time scale?

1. Select a subsection of the sound signals you collected to see the underlying waveform (so that you can see if the data are reasonable!), expand the horizontal axis until you see about six cycles of the waveform on the screen. You should see something similar (but NOT identical) to that shown below.



1. Make the X-axis limits “nice” numbers (i.e. 0.03 to 0.06 sec, or 0.51 to 0.61 sec, depending on how much time is required to show about 6 cycles)
2. Count the number of “fundamental” cycles in this time window (ok to estimate fractions of a period) and use this to compute the approximate period and then the frequency. Note that in the screen shot above, the lowest frequency, or “fundamental” cycle is the time it takes for the full pattern to repeat, i.e. from the big peak to the next big peak (not counting the smaller peak in between. The spacing between the large and small peaks above corresponds to the 2nd harmonic, at twice the fundamental frequency. Record your observations and calculations in your lab notebook.
3. Even though your sung note has a specified pitch (= frequency), it probably does not look like a pure sine wave, which means that there are multiple frequencies present in the signal. You computed above the “fundamental” frequency, i.e. lowest frequency in the signal. You will now take the Fast Fourier transform (FFT).
4. Your **FFT Graph** will be something like that shown below, although at a different frequency and probably with different numbers of visible peaks. **If your peaks are not this narrow, you probably took the FFT of only a portion of the waveform, not the full 1 sec trace.**



1. Insert **Legend** with **Peak Frequency** at the top left of the FFT Graph Options popup box. The frequency of the highest amplitude peak is now shown on the graph. **Beware:** the highest amplitude is not necessarily the fundamental (lowest frequency) peak. The range of frequency is indicative of the FFT resolution.
2. **Save this file** and record the filename in your lab notebook. As always, list the experimental conditions, including sample rate, sample time, source of the sound, how close the microphone was to the sound source, etc. We suggest a **meaningful** file name, such as “singing.CSV”, or “singing\_Lab4\_Sect2\_1.CSV” but not just “Lab4Sect2\_1.CSV” so that you can immediately recognize what conditions you used, without having to refer to the procedure.
3. **Measure** the frequencies (in Hz) of the first three peaks (assuming you have at least three visible peaks). Expand the horizontal axis scale if necessary to help you find the peak frequencies. Write the peak frequency above the corresponding peak in the FFT printout in your lab notebook.
4. How does the **measured** frequency of the lowest frequency peak compare to your estimate above? (Answer in your lab notebook) (*It should agree well, depending how accurately you estimated the period*)
5. Is there a relationship between the frequency spacing (difference in frequency between adjacent peaks) and the fundamental frequency (frequency of the lowest peak)? What is it? If you are unsure, please as your Lab Supervisor or Lab Manager.

The presence of additional harmonics creates differences in “timbre”, or tone of the note. Notes containing more high harmonics will often sound harsher. Try singing the same pitch but in a more forced (“ugly”) tone and see if the harmonic structure changes. Make comments in your lab notebook.

You have just observed one of the fundamental principles: RESONANCE

– the phenomenon on which all musical instruments are based. Each note on an instrument (including your voice) has a “fundamental”, or lowest frequency, but also present in the sound for all but the purest tones are “harmonics” which are integer multiples of the fundamental. Some instruments (like the saxophone and the oboe) include both even and odd harmonics, others (like the clarinet) include primarily odd harmonics. Fundamentals of Fourier transforms will be covered further in seminar; in lab we focus on showing you how to use Fourier analysis to compare different signals and draw conclusions from data.

## Taking Useful Data

## Signal-to-Noise Ratio (SNR) (optional)

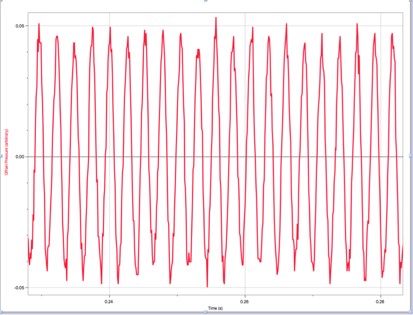
The signal-to-noise ratio (SNR) describes how much power is in the desired signal as compared to the background noise. We will examine how the noise present in a signal can effect SNR by recording the sound from a very pure source produced by a tuner and then introducing noise created by hissing into the microphone while recording the sound together with the tuner.

In this class we will use an APP (Pro Tuner； 专业节拍器) to create accurate pitches. Set the pitch (A, B, C… G) and frequency (in Hz) and record the data each time you start your experiment in your lab notebook.

Turn the instrument tuner APP on and choose Piano or Guitar. We

*Note: the “scientific” scale tuner differ from the conventional American instrument, which middle-C is 256 Hz and the A above middle C is thus 426.6 Hz, not the standard tuning A in America of 440 Hz.*

* + 1. Open a new Arduino file and set the Data Acquisition parameters to serial rate 9600， 1 sec Duration. Turn on the Tuner program again on your cellphone and hold it sideways to the microphone, with the tips about 1 cm in front of the microphone, then press Collect (or have your partner press collect).



* + 1. Change the graph to display Offset Pressure, and then zoom in to see the details of the waveform. The waveform should appear much more sinusoidal than your singing did, as shown on the right, but may still exhibit “beating” (periodic oscillations in amplitude) which indicates the presence of higher harmonics.
    2. **FFT Graph.** Double click on it and select Legend – Peak Frequency and also select Logarithmic Axis for the Amplitude-Axis Options. (Leave the Frequency-Axis as linear, not log). Changing to log y-scale will allow you to measure the very small noise level (the noise should have been hard to see before you changed to log scale)
    3. Adjust the lower limit of the vertical (Amplitude) axis so that only a small amount of noise is visible at the bottom across the entire frequency range, as shown below (remember that you can just click on the lower axis number and then type in the desired value). Your FFT should look something like that below. SAVE YOUR FFT FILE in Matlab. (We suggest a meaningful name such as “Tuner 384 no noise.txt”, not a not terribly helpful name such as “Lab 1 Graph 1.txt”) You will need this file in the postlab analysis.
    4. Find the peak amplitude in FFT, along with the peak frequency. Record in your lab notebook the frequency and amplitude of the highest peak. As a reality check, does the amplitude agree with what you see on the graph? Write both values next to the plot in your lab notebook. How well does the frequency agree with the specified value on the tuner? Compute the discrepancy (defined as measured – expected) in Hz, and the fractional discrepancy (discrepancy divided by expected) in % and record in your lab notebook. **Sign matters for discrepancy: if the measured is less than expected, discrepancy should be negative, as should the fractional discrepancy.**
    - Now record the approximate noise amplitude visible in the graph, just by reading the scale (in the graph above, the appropriate noise amplitude is indicated with a horizontal block arrow). Notice that unlike the exact peak amplitude, you can choose any of a range of values for the noise level and still be “correct”, i.e. the noise level is not as clearly defined. This is fine, just pick a level that makes sense to you. Indicate the noise level you chose with an arrow on your graph and record the value in your lab notebook.

You can now define an “Amplitude SNR” as the ratio of the FFT Amplitude in the largest peak (the signal) divided by the FFT Amplitude of the noise level. However, SNR is in fact defined by the ratio of **power** in the signal to that of the noise. As you learned above, the power is proportional to the **square** of the FFT amplitude (remember the PSD, power spectral density, was proportional to the FFT Amplitude squared). This is analogous to the way that electrical power is proportional to *V*2, i.e. voltage squared.

* + - Record in your lab notebook your measured SNR, defined as

(1),

where the numerator was measured in #8 and the denominator in #9. List your result with only 2 sig figs (remember, the noise amplitude is very approximate). Please clearly label this for yourself as “**SNR tuner alone**”.

Do a “reality check” – look at the graph and figure out the approximate amplitude ratio then square it to get the SNR ratio (i.e., signal ~ 1, noise ~ 0.01 means amplitude ratio ~100, SNR ~ 10,000). This type of “mental math” or “back of the envelope” or “order of magnitude estimate” calculation is CRITICAL for you to be able to quickly spot mistakes in your data or your analysis.

You should have found a VERY high SNR (but probably lower than the 26,000 I measured in an empty lab since you will be in a full lab section with more background noise).

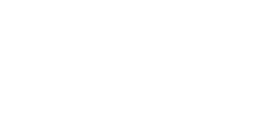
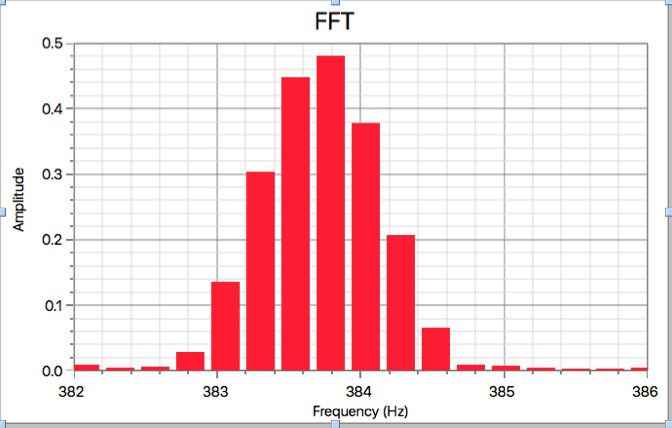
In order to **decrease** the SNR, you will now add noise to your recording, by “hissing” into the microphone while recording the sound of the tuner. You will need to “hiss” later in this lab in order to measure the speed of sound using cross-correlation, so consider this a practice run! We suggesting hissing like a snake (“sssssssssss”), not by using a “shhhhhhh” sound. There should be noise easily visible on the graph of Offset Pressure vs Time, but the FFT peak at the tuner frequency must still be visible above the noise.

* + - In order to make it easier to see if you have “swamped” your signal, change the Amplitude (vertical) axis to linear and Autoscale the FFT graph.
    - Excite the tuner, place it next to the microphone, place your mouth about 10cm from the microphone, start hissing, and then press Collect (or preferably have your partner press Collect!). Check to see if you have a visible peak at the signal frequency, or if it is lost in the noise. If the signal peak is not clearly visible, record a new sample and hiss more quietly. You should be able to easily see the noise with the Amplitude still on a linear scale. As long as you have a visible peak at the signal frequency, **Export the FFT graph** and save the file name in your lab notebook.
    - **SAVE YOUR Data FILE** (You will need it for the post lab analysis), again using a meaningful name such as “tuner 384 hiss.txt”. Note of course that if you are NOT using the G (384 Hz) tuner, your filename should not contain 384!
    - Measure the signal peak frequency and amplitude and record next to the graph, and then indicate the noise level and record the approximate value as you did before.
    - Compute the SNR as you did before, remembering to square the amplitude ratio as shown in Eq. (1) and record in your lab notebook. Depending on how loudly you hissed, you should find a significantly lower SNR, of order 10 – 1000.

**Make sure you have all the Arduino (.txt) files you recorded in this section accessible to you when you leave lab**, as you will use these data to generate probability density functions (histograms). Hopefully you have learned to make meaningful file names (mine were called “tuner 384 no noise.txt” and “tuner 384 hiss.txt”.)

You have just observed another ME 425 fundamental principle on **PRODUCING USEFUL DATA: Recognize inadequate signal-to-noise ratio.** As you just discovered, increasing the noise (by hissing into the microphone) decreases your SNR. You can therefore increase SNR by either decreasing the noise (hissing more quietly) or increasing your signal strength (putting the tuner closer to the microphone). In some cases you could consider amplifying the signal, but must be careful since amplification can increase the noise proportionally to the signal, thus not changing the SNR unless filtering is applied before amplification.

1. Expand the horizontal scale of the FFT Spectrum graph so that you see the large peak in detail, as shown below. It will help interpret the graph if you add gridlines with FFT Graph Options



FWHM

frequency resolution

– Major **Tick** Style – Solid (dark gray) and Minor Tick Style – Solid (light gray).

1. We have defined the **FWHM** (Full Width at Half Maximum) of the FFT peak with the arrows above. This is a commonly used metric to describe the width of a peak. The width should be measured at amplitude of half the peak height. Estimate the FWHM from your data by counting gridlines and record in your lab notebook. Notice that the FWHM is greater than the resolution (the peak is more than one bar wide), due to a phenomenon called “leakage” that is another important aspect of Fourier analysis, but not one that we have time to introduce you to in lab.
2. We have also defined the **frequency resolution (**spacing between frequency values in the FFT, drawn from the center of one bar to the center of the next bar). The **frequency resolution** is most easily determined from the first non-zero frequency value (row 2) in the FFT Freq (Hz) column. Record this value **for your data file** in your lab notebook (which should be what is shown to the right, but check in your data file!)

The **frequency resolution** of the Fourier transform is independent of the sampling rate, and is approximately given by the inverse of the total sample time. In other words, if you sample for 2 sec, you will have approximately 0.5 Hz frequency resolution (approximate because of the effect of padding we described in the prelab). This resolution corresponds to the distance between the **center** of the individual bars in the FFT graph.

## Power Spectral Density (optional)

The most useful information can be gained not from the Fourier amplitude spectrum (which is simply the magnitude of the complex value of the Fourier transform at each frequency), but from the **Power Spectral Density (PSD)**, which, as your heard from the first lecture, gives the power in the signal in a given bandwidth, with units Power/Hz. **The PSD is proportional to the square of the FFT amplitude** and allows you to determine what fraction of the total power in the signal occurs in the frequency band (or bands) of interest, as you will learn today.

To more fully explore the capabilities of the PSD, we want to create a signal with more spread out frequency content, i.e. not just a single pitch. We suggest talking rapidly into the microphone, varying the pitch of your voice between squeaky and low voice, and trying to avoid the natural silent pauses in speech.

1. Keeping the serial rate 115200 and increase the sample duration to 5 sec. **SAVE** your data and again record the experimental conditions and filename - perhaps “talking.csv” in your lab notebook. (This recording of information every time you write a computer file is training for using your Lab Notebook for your Go Forth experiment – **every time you record a data file on your computer, you must record the file name and conditions in your lab notebook!)**

How to compute the PSD from your data.

You could also use Matlab to integrate the PSD. A commonly used integration command is IntPSD = cumtrapz(f, p); (where we are calling the integrated PSD “IntPSD”). This can be normalized by with normIntPSD = IntPSD / IntPSD(length(IntPSD)); Please compute the correct integrated PSD, and remember this should you need it for your Go Forth and Measure project.

1. Generate a new PSD graph on the vertical axis and the **frequency** on the horizontal axis (the frequency column, called **Time – Frequency (Hz),** is accessed through in the **More…** menu of the x-column selection tool). You will probably need to use Autoscale to see the full PSD. Make sure your horizontal axis is frequency (not time), with range 0 – 10000 Hz (the maximum frequency in the FFT is half the sampling rate, as you learned in Lab 2). Notice that the magnitude of the peaks relative to each other should have shifted with a larger difference between small and large peaks because the PSD is proportional to the square of the amplitude.
2. Determine the numerical values.
   * What is the value of the Normalized Integrated Approximate PSD at zero frequency and why?
   * What is the asymptotic value of the Normalized Integrated PSD at high frequency, and why?
   * What does the numerical value at a given frequency indicate?

SAVE YOUR FILE! (same file name is fine)

A helpful way to characterize the integrated PSD, which may be relevant to your Go Forth project, is to include the following type of sentence (where XX, YY, and ZZ would need to be replaced with the numbers corresponding to your measured PSD):

*The measured frequency distribution of speech for this test subject indicates that 80% of the power (10% to 90% fractional power) is contained in the frequency band between XX Hz and YY Hz, with the half power point occurring at ZZ Hz.*

You have just learned one of the ME 425 fundamental principles: **INTEGRATE A POWER SPECTRUM** to determine the power distribution in a signal as a function of frequency. This technique is a powerful way to characterize and compare the frequency content in different signals and to obtain information about a system through the frequency content of the output for varying inputs.

# Using Auto- and Cross-Correlation

Analysis of signals in the time domain is primarily accomplished using ***correlation functions***, as you explored in the prelab. In the next subsection you will use two microphones and a white noise source (your hiss!) to measure the speed of sound. We also want you to measure the autocorrelation function of two signals you have already recorded, but will defer that to the postab to save you time in lab.

1. Make sure you have saved the “singing” data file and also the “tuner hiss” data files that you created in Sect. 2 and 3, respectively
2. Make sure that you email them to yourselves (or save on a USB stick) before leaving lab! You will need them in the postlab.

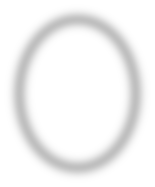
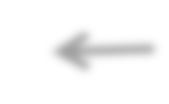
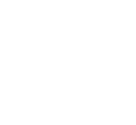
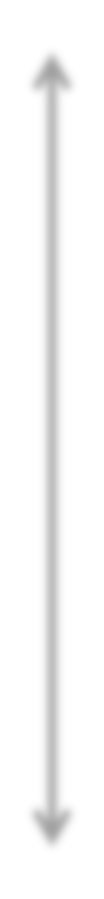
## Using the Cross-correlation Function to Measure the Speed of Sound

Now we are going to use two microphones and a white noise source to measure the speed of sound.

* + - Find the other microphone on your bench and connect it to Arduino UNO. Open a New File in Arduino IDE and it should now recognize the two attached microphones.
    - Open the Arduino **to run data collection**. Since you have two sensors attached to the Arduino, you may have to limit the maximum sample rate to 57600,. (*If for your Go Forth experiments you need to use two sensors with higher sample rate, you will need to use two* Arduino *interfaces. This is fine.*) Change the sampling **Duration** to 0.5 sec.

(*space intentionally left blank*)

* + - Place the microphone connected to **A1** (**Mic 1**) at the outer edge of the long side of your lab bench, facing outward. Place the other microphone connected to A2(**Mic 2**) facing in the same direction with its tip 1 m away, as shown below. (Use the orange metric tape measure in your lab bench drawer). You may find it helpful to use the tape on your lab bench shelf to fix the microphones to the lab bench



Toward Mic 1

100 cm = 1m

Mic 2

1 m

**HISS HERE!**

(100 mm = 4”

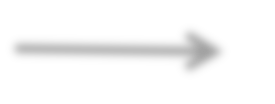
from Mic 1)

Computer

Mic 2 tip

Mic 1 tip

* + - Determining the delay time via cross-correlation works best when the signal is white noise with the sharp autocorrelation function you observed for the hissing you measured in the previous section.



Toward Mic 2

Mic 1

**HISS HERE!**

* + - **Collect data** in Arduino UNO. QUIETLY place your mouth near Microphone 1, **inline** with the 2 microphones but NOT IN BETWEEN THEM (i.e. on the continuation of the line from Mic 2 to Mic 1, approximately 100 mm (4 inch) from Mic. 1, as shown on the previous page). Take a DEEP BREATH and start producing white noise. You will actually have to make the sound for more than the 0.5 sec data duration, because of some delays in the start of data collection from the start of the sound.
    - Unfortunately, Arduino is not set up to compute correlation functions, so you need to export the data to Matlab to do so. You will compute the cross-correlation function by modifying the m-file you or your partner created in the prelab. You must now export the data from Arduino as a CSV, but first must increase the numeric resolution of the data. If you were successful, you should see two traces on the screen, with the Mic1 signal having larger amplitude than the blue Mic2 signal, assuming you correctly set Mic1 as the one closer to the “hisser”.
    - Increase the resolution of both **Mic** columns to **3 Significant Figure**s (the default should be 3 Decimal Places) by double clicking on the Column title and changing the Displayed Precision in the Options tab). Then use **File – Export As** to save as a **CSV** version of this file. Also save your Logger Pro file at this point, and record both file names in your lab notebook, making sure to also record which microphone was closer to the sound source and the **actual microphone spacing with 1 mm resolution** (should be approximately 1.000 m, but could be 0.998 m, 1.003 m, etc, it doesn’t matter if it is not exactly a meter as long as you record the exact spacing with 1 mm resolution). We suggest putting the nominal microphone spacing in the filename, i.e. **2mics\_hiss\_1m.txt.**

You will now use *xcorr* in Matlab to compute the cross-correlation function between the signals detected by the two microphones. If there is a delay between two signals, *x* and *y*, the Matlab function *xcorr(x,y)* will return a peak at **negative time if *y* is lagging *x***, and at positive time if *y* is leading *x*.

* + - Open Matlab and press the Import Data button. Browse to the saved CSV data file and open it as you did in the Prelab. It should open in the Import window. After you select the file, make sure that you import as **Column Vectors,** not Table. (Table is the default option on a PC, so you must change to Column Vectors). Import all the data. The Workspace window should now contain 5 column vectors, labeled with the Logger Pro column headings, each with 5001 elements (0.5 sec Duration at 10 kHz Sampling Rate).
    - Assuming you did not insert an FFT graph (not required, but you may have done so if you were curious!) you should have imported 5 columns of data. The ones of interest are the time column (**LatestTimes)** and the two Offset Pressure columns (**LatestMic1** and **LatestMic2** if you changed the names as directed in Step 2 above)
    - Open the m-file you created in the prelab. Make changes to the variable names as required, for example, **time** is now called **LatestTimes** and **signal1** (or **signal2**, whichever you had when you last saved the m-file) should be called **LatestMic1.** You want to compute a cross-correlation, rather than the autocorrelation you did in the prelab, so you need to change the arguments to *xcorr* to xcorr(LatestMic1,LatestMic2); or whatever the columns for the two microphones are named.

NOTE: You can use the autocomplete feature in Matlab by typing LatestT and then tab and it will suggest option for completion, the only one will be LatestTimes

* + - **Run** the m-file.
* Verify that 1/dt is the sample rate (*what was your sample rate?*)
* Look in the Workspace window and make sure t and output both have 10001 elements.
  + - Graph the result! plot(t,output). You should see a large spike near the center of the graph. Zoom in on it by changing the x axis range to 0.01 using **Edit – Axes Properties** in the **Figure 1** window. Keep zooming in if necessary until you can clearly see the peak.
    - Find the index of the peak as you did before: [peak,index]=max(output); (*we have assumed that the largest peak has* ***positive*** *amplitude. If it is* ***negative****, use the min function rather than the max function*).
    - Record in your lab notebook the amplitude (positive or negative) and index of the peak, as well as the lag time at which the peak occurs.

If there is a delay between two signals, *x* and *y*, the Matlab function *xcorr(x,y)* will return a peak at **negative time if *y* is lagging *x***, and at positive time if *y* is leading *x*.

* + - Which signal is lagging? Mic1 or Mic2? To which physical microphone does this correspond? (i.e. closer to or further from the sound source). Does this make sense? (*always THINK about your results*).

Create the table shown below in your Lab Notebook and use your results to fill in the first row, using the **measured** distance (with 1 mm resolution) and delay time from Matlab to determine the speed of sound. (*Note that although the sound speed is* ***computed*** *from your measurements, it should be referred to as a* ***measured*** *quantity, since it is derived from measurements, rather than a* ***calculated*** *quantity which implies calculation from first principles or a numerical model. Please remember this distinction throughout your time in ME 425 and refer to any results obtained from measurements as* ***measured*** *quantities!)*

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Nominal Distance (m)** | **Measured Distance *x* (m)**  ***0.001 m resolution*** | **Delay time *t***  **(s)** | **Sound Speed *v***  **(m/s)** | **Filename** |
| 1 |  |  |  |  |
| 0.8 |  |  |  |  |
| 1.2 |  |  |  |  |
| 1.4 |  |  |  |  |

You should notice that we have left you three other rows in the data table below. Why? When taking data, it is always good practice to vary an input parameter in a known way, and make sure the result changes in the way you expect. When you compute the speed of sound for your first data set, you should get something close to 343 m/s, the theoretical speed of sound in air at 20C. However you should **ALWAYS** check your data (and analysis method). We will do so in this case by performing the experiment at three other values of microphone spacing, and making sure we get consistent results for the speed of sound. Don’t worry if your value is off by 10-20% - we will have you estimate the uncertainty of your result in the postlab. If, however, you do not observe a clear peak in the cross-correlation function, or your value is less than 300 m/s or

greater than 400 m/s, please repeat the experiment, perhaps using your lab partner as the noise generator.

Please note that when you start taking measurements for your Go Forth and Measure project, you should start with conditions under which you expect to measure an effect, ideally with a known value, but if not known, at least change the conditions and see if your results change in a reasonable way. This is important to make sure the sensors and your analysis method are working correctly. You can then proceed on to the effect you actually want to study.

Repeat the experiment above, changing the distance between the microphones as indicated in the table, saving the **Arduino** and CSV files. If you are running out of time, it is NOT NECESSARY to do the Matlab cross-correlation now – simply collect the data, **SAVE THE Arduino FILES**, and fill in the rest of the table during the postlab.

Show your calculations of Sound Speed from Distance and Delay time in your lab notebook.

You have just leaned another ME 425 fundamental principle: Use **auto-correlation** to determine properties of a signal, and **cross-correlation** to measure a system parameter (the speed of sound).

# You are done with the experiments! Please do the following before leaving:

* 1. Please replace all equipment in the correct box or bag.
  2. Please remove any used tape from the bench and throw out, and put the tape dispenser back on the top shelf of your lab bench
  3. Please recycle all your paper scraps and put the scissors back in your lab bench drawer
  4. Make sure to **email all your data files** to yourselves (or take away with you on a USB stick)
  5. Please **Log Off** of the computer – in Windows 10, if you leave yourself logged one, the next student cannot log you off and must Restart the computer in order to log on L.
  6. Finally, please remember to get your lab notebook signed before you leave lab. We will want to talk to you about the experiment to make sure you understood it all.

# POSTLAB

# Complete this section by yourself (i.e. not with your lab partner, although you are welcome to compare your results with him or her), include all required graphs, and submit as PDF on Stellar (preferred) or in paper copy to the Completed Assignments bin in the ME 425 lab by the date and time listed on Stellar

The boxed areas below will be graded and have been provided in a Word document on the Wiki called “Sound Postlab Questions Word”. Please use this document to record your answers, and make sure to include all the requested graphs, as listed at the end of this (and that) document.

## Creating a Good Histogram of Hiss Data

Analysis of signals in the amplitude domain is primarily accomplished by creating an amplitude histogram, which can be related to the probability density function. In this section you will create an amplitude probability function and then covert to a probability density function for both your tuner signals with and without noise.

* + 1. Open the “tuner with hiss” file you created in Sect 3.1.

Histograms are defined by the size of the **bin** – i.e. the size of the buckets into which the incoming signal is sorted. You therefore need to determine the minimum and maximum amplitude of your signal, in order to figure out the appropriate bin size.

Measure the **Statistics** on the Offset Pressure vs Time graph to find the min, max, and y values:

**Question 1:** min: max: y:

Start with 20 bins across y, so figure out the bin size you would like, and round to the nearest

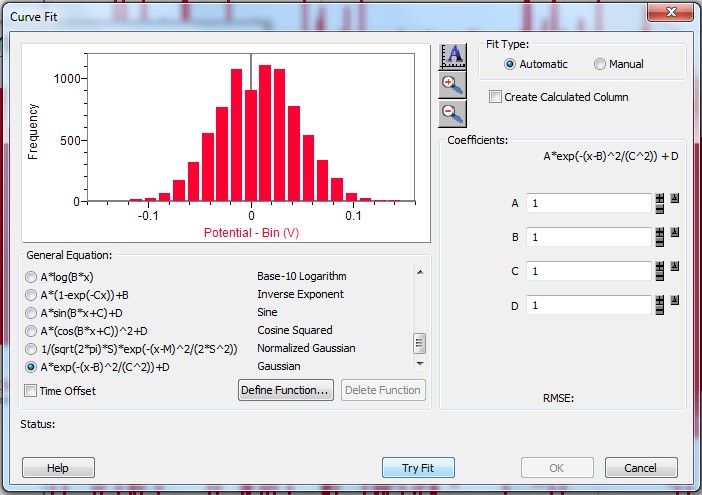
0.005 (arbitrary units). You should end up with no more than 2 sig figs on the number listed below:

Bin size:

* + 1. Find the Histogram Options – Bin and Frequency Options in Matlab. Change the Bin size to that you listed above.
    2. Set the Bin Start to (half the Bin size). The program will change the bin locations as needed so that all the signal fits in one of the bins.

Now look at the histogram – does it look like a function you should be familiar with from statistics? It should appear like a bell curve, or a Gaussian. We can actually fit the data to a Gaussian profile in Matlab. (***Note***: *your histogram should be centered on zero. If it is not, make sure you took the histogram of the Offest Pressure column, not the Sound Pressure Column*.)

* + 1. Fit the histogram using **Gaussian, until the plot** look like a good Gaussian fit. List the equation and the parameters in the space on the next page. Please make the histogram graph larger in the window, select it, copy it into Word or Paint and then print it showing the Gaussian fit to submit with your Answer Booklet.



Save your file and also record below the functions used by MATLAB and fit parameters, as well as the RMSE (Root Mean Square Error, which gives an estimate of the goodness of the fit).

**Question 2:**

1. Matlab Fit Function (nicely formatted with Equation Editor ),
2. Coefficients with Uncertainty, and RMSE:

We should really have you convert this into a **probability distribution function**, by dividing by the total number of samples so that the y-axis is fraction of samples in this bin, rather than number of samples. We should then go the extra step of converting to a **probability density function** by dividing the fraction of samples by the width of each bin. The y-axis units in this case will be 1/V, rather than dimensionless fraction. Once it is converted to a probability **density** function you should fit to the **Normalized Gaussian** function to determine if the distribution is in fact Gaussian. However, you are doing a lot in this postlab, so we will just alert you to the fact that you should use **probability distribution** or **density functions** in your future work, including your Go Forth and Measure project.

## Creating a Good Histogram in Matlab of Tuner Data

Repeat the steps above for the “tuner” data (the one when you weren’t hissing) to make a good histogram. Does the distribution look Gaussian? (It shouldn’t) Does it make sense to fit this to a Gaussian? Think about what the probability distribution of a sine wave should be, does this make sense? You will have to explain the shape in your figure caption.

## Auto-correlation Function

Let’s go back and look at the autocorrelation functions of the “singing” and “tuner with hiss” files you recorded in Sects. 2.1 and 3.1. Start by looking at the auto-correlation function for the sung note you recorded in Sect 2.1, and hopefully gave a filename that includes “singing” in the title so that you can easily fine it.

* + 1. As you discovered in lab, Aruidino is not set up to compute correlation functions, so you need to export the data to Matlab.
    2. Open Matlab and press the Import Data button. Browse to the saved CSV data file and open it as you did in the Prelab. It should open in the Import window. After you select the file, make sure that you import as **Column Vectors,** not Table. (Table is the default option on a PC, so you must change to Column Vectors). Import all the data. The Workspace window should now contain 5 column vectors, labeled with the Logger Pro column headings, each with 20001 elements (1 sec Duration at 20 kHz Sampling Rate).
    3. Open the m-file you created in the prelab. As in lab, make changes to the variable names as required, for example, **time** is now called **LatestTimes** and **signal1** (or **signal2**, whichever you had when you last saved the m-file) should be called **LatestOffsetPressurearbitrary.**

NOTE: You can use the autocomplete feature in Matlab by typing LatestT and then tab and it will suggest option for completion, the only one will be LatestTimes

* + 1. **Run** the m-file.
* Verify that 1/dt is the sample rate (*what was your sample rate?*)
* Look in the Workspace window and make sure t and output both have 40001 elements.
* Look at the graph of the autocorrelation function. It should look similar to that shown to the right. (But will be different in shape depending on the harmonic structure of your sung note!)
* Use **Copy Fig** to copy this figure into your Postlab Word file, and give it a short **numbered** caption that says something like “Autocorrelation function of sung note”.
* In order to examine it more closely, expand the x-axis to display

from 0.01s to 0.01s using **Edit – Axes Properties** in the **Figure 1** window. You should see that under the envelope there is an oscillatory function with multiple frequencies visible. The autocorrelation of my data is shown below, yours may look quite different depending on how many harmonics were present in your sung note, but should look periodic. The spacing between the large peaks (one full repeat of the pattern) corresponds to the fundamental period. The spacing between smaller peaks corresponds to the period of the harmonics of the fundamental. If you sung at a lower pitch and don’t see this many cycles, expand over a larger frequency range until you see 6-10 periods in the expanded view.

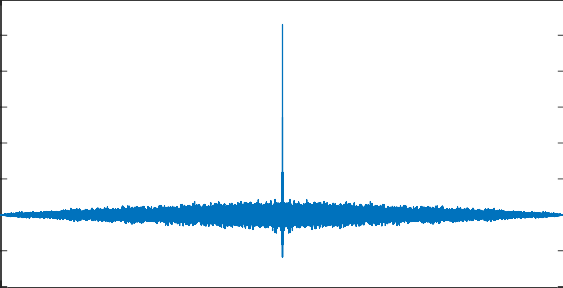
* **Copy the expanded autocorrelation graph** into your Postlab Word file and give it a short numbered figure caption.
* Estimate the frequency from the expanded graph and add this information to the expanded-view figure caption as follows:
* Count how many full (lowest frequency) cycles there are in the expanded graph region and use this to estimate the period and then the frequency shown in the Matlab autocorrelation graph. Record both the estimated period and derived frequency in the Word file caption with the correct units (sec for period, Hz for frequency).
* Determine the fundamental frequency from the Logger Pro FFT (careful! It may not be the Peak Frequency shown in the legend! You want the lowest frequency peak, not the highest amplitude peak) and also record in the figure caption with units. **Use complete sentences to include this information in your caption!**
* Do these two frequencies “agree”? (How accurate do you think your period estimation is from Matlab, and to what frequency accuracy does that correspond? The frequencies only need agree within this amount) **Make comments in the caption about the agreement or disagreement**, be quantitative, and be careful with your sig figs!

As expected, the “pure tone” created by singing is well-correlated with itself – the signal at later time is related in a consistent way (a sine wave) with the behavior earlier in time. If you do not understand this, please ask on Piazza! (We have pushed this part into the postlab to save you time in lab, so unfortunately we are not with you in person to answer your questions)

You will now examine the “white noise” you generated by hissing while recording the tuner in Sect 3.1, the same file you used to generate the histogram earlier in the postlab.

* + 1. Open your “tuner hiss” cmbl file in Logger Pro. Again increase the resolution of the **Offset** column to **3 Significant Figure**s and then use **File – Export As** to save as a **CSV** version of this file.
    2. Type **clear all** in Matlab to remove the data from your previous file. Import the data from the CSV you just created. This should have been acquired with a Duration of 1 sec and Sampling Rate of 10kHz, so you should now have 10001 elements in all the imported Logger Pro columns.
    3. Run the m-file. **Copy Fig** to insert this graph into your Postlab Word file, and then expand the scale to 0.05s to +0.05s and **Copy Fig** to insert this expanded graph into your Postlab Word file. You may either put the two graphs side-by-side with one figure caption describing each graph (labeling them as **a)** and **b)** with text boxes placed in the upper left corner of each graph as shown below, and describing a) and b) in the caption) or as two separate figures with separate captions. The graphs from my data are shown on the next page as a reality check, but if your SNR was larger than my measured value of about 40, you should see much more oscillation, both at zero lag (under the noise spike) and away from the zero-lag peak. (If your SNR is smaller and you see more oscillations, please check your SNR calculation because it is probably incorrect.)

30 30



a)

b)

25 25

20 20

15 15

10 10

5 5

0 0

-5 -5

-10

-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

-10

-0.05 -0.04 -0.03 -0.02 -0.01 0 0.01 0.02 0.03 0.04 0.05

* The large spike at zero time is a result of the uncorrelated signal – the sound level at any later time is related in a random way to that at earlier time, in other words, you cannot predict future sound levels based on the past, unlike the case with a pure sinusoidal tone.
* If you observe oscillations in the expanded view (as shown on the right above), what should the frequency be? Estimate it by measuring the period as you did before, and compare to what you expect.
* You can find the index of the peak with the **min** or **max** command, depending on whether the amplitude of the largest peak is positive or negative. We will define two new variables, **peak** and **index**, which are the amplitude of the peak and the index of the peak using [peak,index]=max(output); (*we have assumed that the largest peak is* ***positive****. If it is* ***negative****, use the min function rather than the max function*). You can then find the time at which the maximum occurs with t(index). Record in your lab notebook the peak value (indicating positive or negative), the index of the peak, and the corresponding time, which should be 0.0 because this is the autocorrelation of noise, and by definition, the largest peak is at zero time. *If you don’t understand why, please ask!*
* Include in your figure caption the following information, incorporated into **complete sentences** with a **logical flow of information**: (i) SNR for the signal as measured in lab (only 2 sig figs please!) (ii) frequency of the tuner, which should be listed in your lab notebook (iii) frequency estimated from the period of the oscillations visible in the expanded view (careful with your sig figs) (iv) a discussion of the agreement or disagreement between the estimated oscillation frequency and that of the tuner, and (v) why the peak of the autocorrelation function is at zero time.

## Measure Speed of Sound with Uncertainty

We introduced you to two three methods for estimating uncertainty in your measured results. These methods are the **statistical method** (useful when you have multiple measurements of the same phenomenon and you want to find the uncertainty of either the mean value of multiple measurements or of best-fit parameters for pairs of measurements, [*x,y*]) the **propagation of uncertainty method** (useful when you can only do one measurement, but have uncertainty estimates for the parameters on which your final result depends), and the **uncertainty on fit parameters.** You will use a linear fit to determine the speed of sound with its uncertainty, but will also include error bars on your graph in both x (time) and y (distance) dimensions.

**Question 4:**

1. Rewrite below the equation for sound speed *v* in terms of ***x*** and ***t*** that you should have written under the table on p. 22
2. For the uncertainty in ***x***, estimate how well you made the measurement with the tape measure (should be 1 – 2 mm unless you were not careful with the measurement). List this value below as 𝑢∆c and give it the correct units

𝑢∆c =

We will assume that the uncertainty in time, 𝑢∆d is given by the time resolution of the data, which is the spacing between time points, or the inverse of the Sampling Rate. Write this value below with the correct units:

**Question 5:**

𝑢∆d =

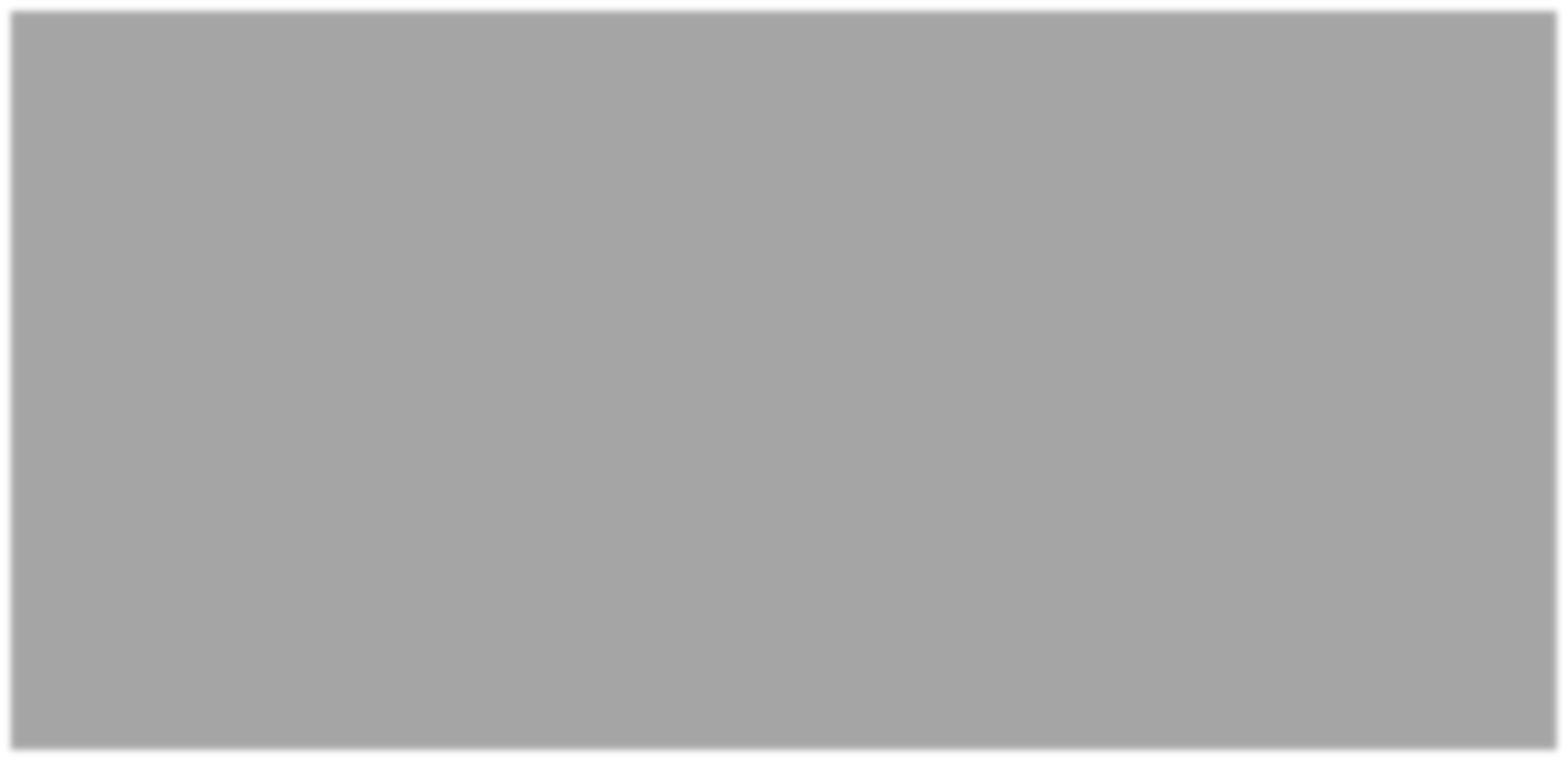
3. Fill in the blanks below, assuming that you will be fitting a line with an adjustable slope and intercept to your data

Number of points Number of parameters: t-factor

4. Create a **professionally formatted graph with descriptive figure caption** of ***x*** vs. ***t*** and use the slope of the best-fit line to measure the speed of sound with its 95% uncertainty. If you used Excel or Logger Pro for the fit, make sure to multiply the uncertainty reported by the program by the correct t-factor listed above. Professional format the graph and write a descripting figure caption, giving the sound speed with its uncertainty (and units!) as well as a description of the meaning of the error bars. As always, the caption must also describe the system so the reader understands what is being plotted. The caption should also comment on the agreement or disagreement of your measured value with the expected value for air at 20C of 343 m/s. *(In other words, when you include the uncertainty on your fit slope, does the expected value overlap measured value?)*

*(please see* ***important information*** *on Sig Figs on the next page, in the yellow box)*

# Please remember these rules in future ME 425 work!



Please remember the Sig Fig rules as given in the Significant Figures handout in your Course Reader:

1. Your computed uncertainty should be reported with **at most 2 significant figures**
2. Your value should be expressed to the **same numerical precision** as your uncertainty, i.e. to the same lowest digit

For example, if you measured a sound speed of 325.264 m/s with a computed uncertainty of 26.745 m/s, this should be reported as

(325  27) m/s

If, instead, the uncertainty were 6.745 m/s, the number should be reported as

(325.3  6.7) m/s

**Never truncate numbers until the final reporting step.** In other words, all calculations should be done with the full numerical resolution available to you and the extra digits should not be dropped until you report the final numerical result. Numerical round-off error can be quite significant, easily 10%, if you drop digits too early.

**Required Attachments: (also include boxed areas above)**

1. Histogram created in Logger Pro for the “tuner hiss” data showing Gaussian Fit (not necessary to professionally format) with a good, **numbered** figure caption (**Figure 1**) that includes the fit coefficients with uncertainty.
2. Logger Pro Histogram of “tuner” data with good, **numbered** figure caption (**Figure**

**2)** explaining the observed shape.

1. Graphs with captions of the autocorrelation function results requested in Sect 5.4, inserted in your Word file in the indicated location (**Figures 3, 4, and possibly 5)**

# Professionally formatted graph with descriptive numbered figure caption of *x* vs.

***t*** used to determine the speed of sound. Be sure to include the measured speed with uncertainty following the sig fig rules, and compare to the expected speed. **(Figure 5 or 6** depending on how many figures you used in #3 above**)**