# 1 Introduction

In this lab, we will construct an audio spectrum analyzer using an Arduino microprocessor connected to an integrated circuit containing a microphone and amplifier.

# 2 Fourier Series for Discreet Data

To tune our guitar, we need to determine the frequency spectrum of sound when a string is plucked. Conceptually, the guitar provides $f(t)$ and we simply wish to calculate the $\widetilde{f}(\omega)$.

But we do not have a continuous function $f(t)$ Instead, we will be sampling the microphone at discrete intervals separated by the sampling period $\tau$. Our time series data will consist of $n$ microphone values $x_j$, collected during a time interval of duration $T = n\tau$, and approximately evenly spaced.

Let's start with the Fourier Series for a function of time with period $T$. In this case, neglecting the overall normalization, which we don't care about, the formula for calculating the Fourier coefficients is:

$$\begin{aligned} A_m &= \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \cos\left(\frac{2\pi m}{T}\, t\right)\, dt \\ B_m &= \int_{-\frac{T}{2}}^{\frac{T}{2}} f(x) \sin\left(\frac{2\pi m}{T}\, t\right)\, dt \end{aligned}$$

we can describe our discrete data sample, however, as:

$$f(t) \to \sum_{i=0}^{n} \delta(t - i\tau) x_i$$

that is, it is zero everywhere but at the $n$ sample points, each at location $i\tau$ in time, where the value is $x_i$. The delta function replaces the integrals above with sums, and the coefficients are now given by:

$$\begin{aligned} A_m &= \sum_{i=0}^{n} x_i \cos\left(\frac{2\pi m}{n}\, i\right) \\ B_m &= \sum_{i=0}^{n} x_i \sin\left(\frac{2\pi m}{n}\, i\right) \end{aligned}$$

The sample theorem informs us that we won't handle properly periods less twice the sampling period $\tau$

$$\omega_{\max} = \frac{2\pi}{2\tau} = \frac{2\pi(n/2)}{T}$$

from which we see that we must ensure that there is no component to our signal with frequency $f > 1/2\tau$ and then we need only consider Fourier coefficients up to $m < n/2$.

# 3 Design Considerations

For the Geiger lab, we found that the Arduino could sample a digital input pin about once every 4 $\mu$s. The analog read time needed in this case is much slower, about 150 $\mu$s. That means we will

need to worry about aliasing from signals with frequency greater than 3.3 kHz. In principle, we should put a filter in place to ensure there are no such signals, but this is high enough in the audio frequency that we can probably skip this. Instead, we'll just let the natural frequency falloff from the microphone handle this for us.

# 4 Inital Setup

**Step 1: Build the circuit**

Obtain your Arduino, protoshield, and microphone IC. Connect the LED of the protoshield to digital pin 13 on the Arduino (this will be used later). Connect the microphone IC as follows: GND to ground, VCC to the 5 V supply, and AUD to the Arduino analog input A0.

Test that microphone is working properly by connecting your digital scope between ground and AUD. If you whistle near the microphone you should be able to produce a plausible sine wave. Measure the whistling frequency range of your group.

**Step 2: Install the Whistle Detector Arduino Sketch**

To further test your setup, I have provide a "Whistle Detector" sketch. If the microphone and LED are installed correctly and operating on the Arduino, then this sketch will light the LED when it detects significant sound (such as a whistle) on the microphone. Download this sketch and ensure that your hardware is all working.

# 5 Spectrum Analyzer

The rest is up to you. Design a sketch for the Arduino which calculates the Fourier coefficients of your discreet time samples.

You can handle this in many different ways, e.g. on the board, or by shipping the time series out the serial port and processing it offline with python. It is all up to you, the designer!

Some considerations:

- Make sure you monitor you sample time. If you load down your sampling code, it will increase dramatically, and likely it will not work.
- Test your plan in python with a toy model for your discrete data sample.
- Keep in mind that the microphone input has a 2.5 volt pedestal... this allows for negative excursions while remaining in the range 0 to 5 V.
- Always test your code with fake data. Real data is noisy. For instance, in your sketch, have the ability to override the analog input with fake data containing a known sine wave. You should reconstruct a nice sharp wave function.
- This is a challenging lab. If you manage to produce something that can tell the difference between a high and low pitch guitar string, I consider it a great success.

# 6 Write-Up

You should provide a short write-up describing your plan and showing some python plots for your algorithm.

Include any results you obtain in the lab. If you were unsuccessful, describe where you encountered difficulty.