Moises Beato Nunez

[Moises.beatonunez@gmail.com](mailto:Moises.beatonunez@gmail.com)

**CMSIS-DSP FIR Filter Instructions**

The following is a complete, step-by-step instruction into how to design a FIR filter using the CMSIS-DSP library, MATALB and the Teensy 3.6 MCU.

Who is this for?

This guide is for EET students taking a DSP course who are novice programmers and must do a DSP project. (that’s me). The [CMSIS-DSP library](http://www.disca.upv.es/aperles/arm_cortex_m3/curset/CMSIS/Documentation/DSP/html/index.html) is not very novice friendly, and the gap in programming knowledge can make it very difficult to understand. So, this guide serves two purposes: To help other people who want to do the same thing, and to document my work.

What are we doing here?

The big picture: You are given a wave file containing the background noise of a city. If you listen carefully, you can hear birds chirping. We are designing a filter to block out everything except the sound of the birds.

How are we doing it?

The idea is to convert the wave file into binary using MATLAB, save the data into a text file, put the text file in a SD card, have the Teensy 3.6 read and filter the data, then export it into another text file that we, in turn, read back in MATLAB.

Couldn’t we just filter it in MATLAB?

Absolutely. It would take 4 lines:

1. Read the data
2. Design the filter
3. Filter the data
4. Play filtered data back

But that would hardly be a challenge.

What do we need?

* MatLab: I’m using R2015a. For earlier releases, when reading the wave file use the “waveread” function, rather than the “audioread” function.
* Teensy 3.6: We are using it because of its floating-point-unit (FPU) and because its AMR Cortex-M4 processor makes it compatible with the MCSIS-DSP library
  + We are using the [Arduino IDE](https://www.arduino.cc/en/Main/Software) with the [Teensyduino add-on](https://www.pjrc.com/teensy/teensyduino.html).
* The CMSIS-DSPlibrary: copy this files in to your work directory (where your .ino file will live), and follow [this guide](https://forum.pjrc.com/threads/38325-Excellent-results-with-Floating-Point-FFT-IFFT-Processing-and-Teensy-3-6?p=119797&viewfull=1#post119797) on how to modify the library to work in the Arduino IDE
  + arm\_common\_tables.h
  + arm\_const\_structs.h
  + arm\_math.h
  + core\_cm4.h
  + libarm\_cortextM4lf\_math.a
* Micro SD card: The SD library is included with the Teensyduino add-on.
  + Use the “CardInfo” example from the SD library to confirm that your Teensy recognizes you micro SD card.

OK, let’s get started!

**Section 1: Exporting the Data**

Turn on MATLAB and switch to your working directory (the place where the wave file is saved). Run the fileExport code:

[Sound,fs] = audioread('background\_noise\_city.wav');

mono = (Sound(:,1)+Sound(:,2))/2;

mono = mono';

N = length(mono); % number of samples

fid = fopen('input.txt', 'w');

fwrite(fid,mono,'float'); % outputs data as binary rather than ASCII

fclose(fid);

This code should:

1. Read the wave file into the Sound variable
2. Get the sampling frequency. In this case . You will need this when designing the filter
3. Convert the Sound file from stereo to mono, essentially cutting the number of samples in half.
4. Acquiring the number of samples. In this case . You will need this in the Teensy code.
5. To play the data sounds like use
6. Exports the file as binary data into a text file called “Input.txt”.

Note: You can name this file whatever you like, just make sure that your Teensy code reflects it.

Why do we export the file as Binary data?

Per the [SD library](https://www.arduino.cc/en/reference/SD) documentation, the *file.*read() function expect an array of characters or bytes, yet the mono data is in an array of floats. So, the Teensy reads the data in as bytes via the SD card, then we convert the data to float before filtering.

Now, save the “input.txt” file into the Micro SD card, put it to the side and let’s design a filter.

**Section 2: Designing the Filter**

Unlike IIR filters, FIR filters are completely defined by their coeffects. This means that we don’t have to deal with the difference equation. So, all we must do is design a filter in MATLAB and export the coefficients.

After doing some [research](https://www.allaboutbirds.org/do-bird-songs-have-frequencies-higher-than-humans-can-hear/), I found that, for humans, the perceptible range of a bird song is between 1 kHz and 8 kHz. So, after some trial and error, I decided to creak a bandpass filter with bandwidth between 2 kHz and 5 kHz. Open MATLAB and run the coeffExport.m script.

This code should:

bpFilt = designfilt('bandpassfir','FilterOrder',128, ...

'CutoffFrequency1',2000,'CutoffFrequency2',5000, ...

'SampleRate',fs);

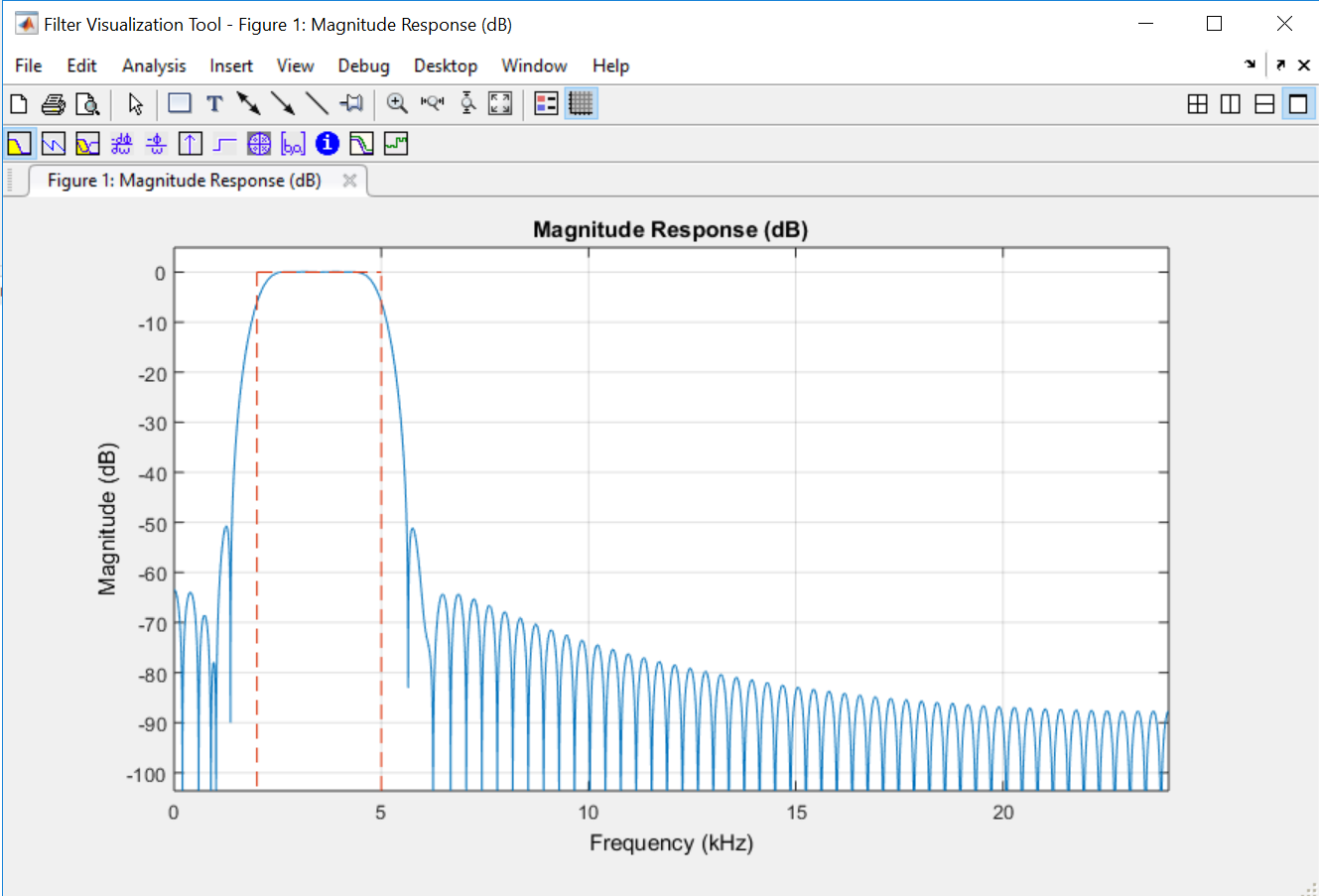
coeff = bpFilt.Coefficients;

fid = fopen('coeff.txt', 'w');

fprintf(fid,'%.7f,\n',coeff);

fclose(fid);

1. Design a 128 order FIR bandpass filter with a bandwidth range between 2 kHz and 5 kHz, and a sampling frequency of 48 kHz. (see the previous section for the proper sampling frequency).
2. Export the coefficients as a float with 7 decimal places into a text file called “coeff.txt”
3. To view your filter, use the FVTool. Ex: which output the following image.



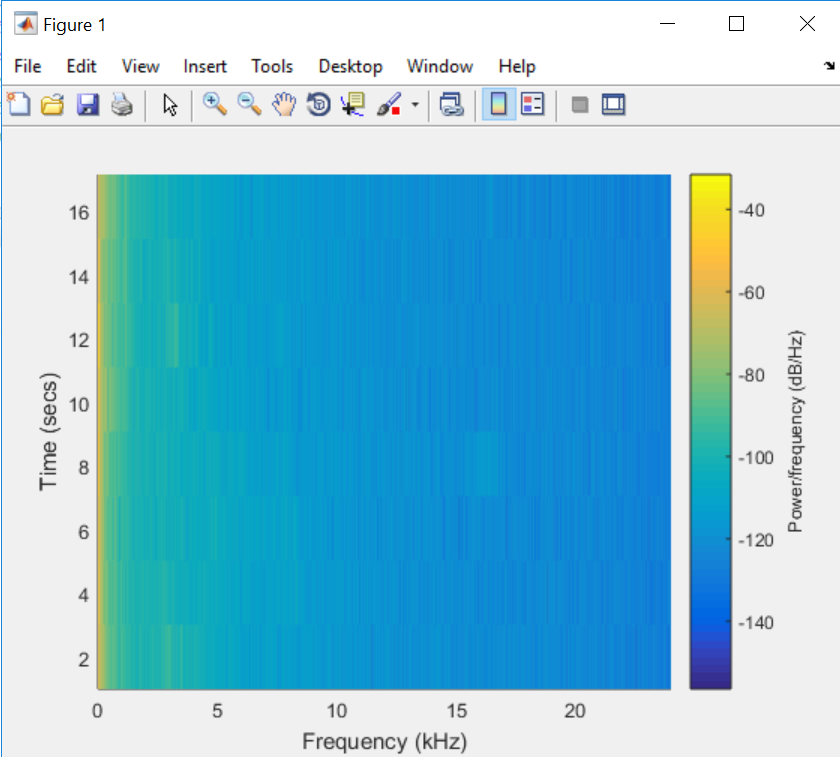
**Section 3: Test your Filter**

Before we go into the Teensy code, lets confirm that the filter does what we want it to do.

Let’s start by looking at a spectrogram of the unfiltered data. Use

spectrogram(mono, [ ], [ ], [ ], fs);

You should be getting this image:



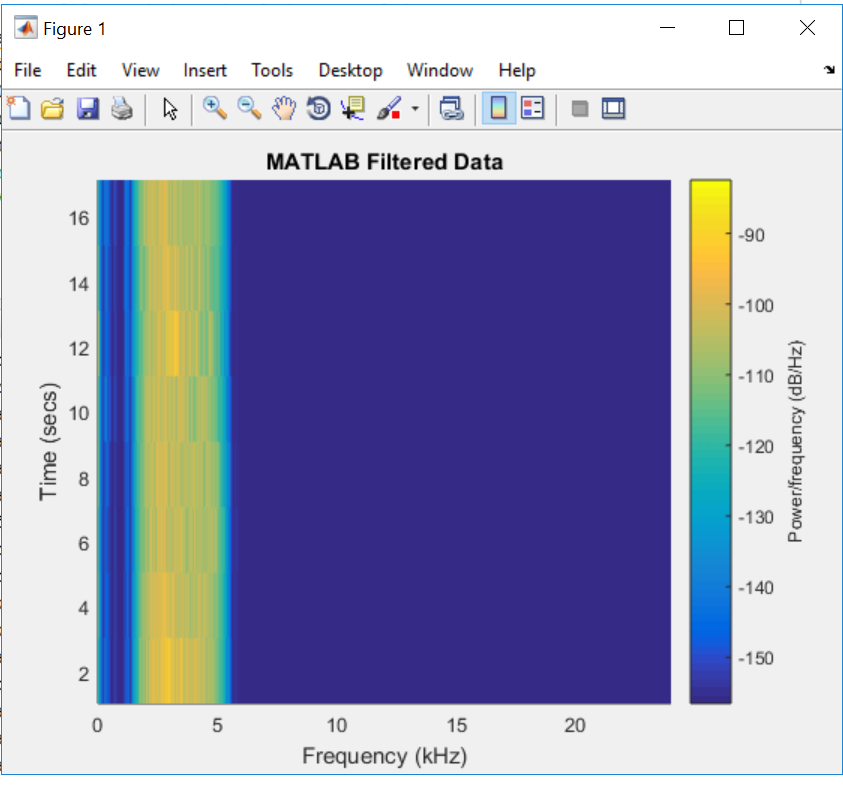
As you can see, and probably heard in section 1, most the power is in the lower frequencies (yellow area), with some power between 1kHz and 5kHz, then nothing but high frequency noise after that.

Now, we will filter the data and run another spectrogram

output = filter(bpFilt,mono);

spectrogram(output,[],[],[],fs), title('MATLAB Filtered Data');

sound(2\*output,fs);



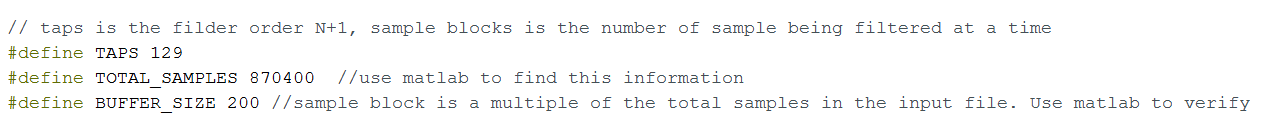
As you can see from the spectrogram, and here from MATLAB, the filter works.

Now we replicate this in the teensy code.

**Section 4: The Teensy Code**

Remember the “input.txt” file? If you haven’t already done it, copy the file into the micro SD card, the inset the SD card into the Teensy 3.6 SD slot.

Now open the cmsis\_fir\_filter.ino file and follow along. The code has comments where I felt it needed them, but we will still go over the lines that are not immediately clear.



TAPS:

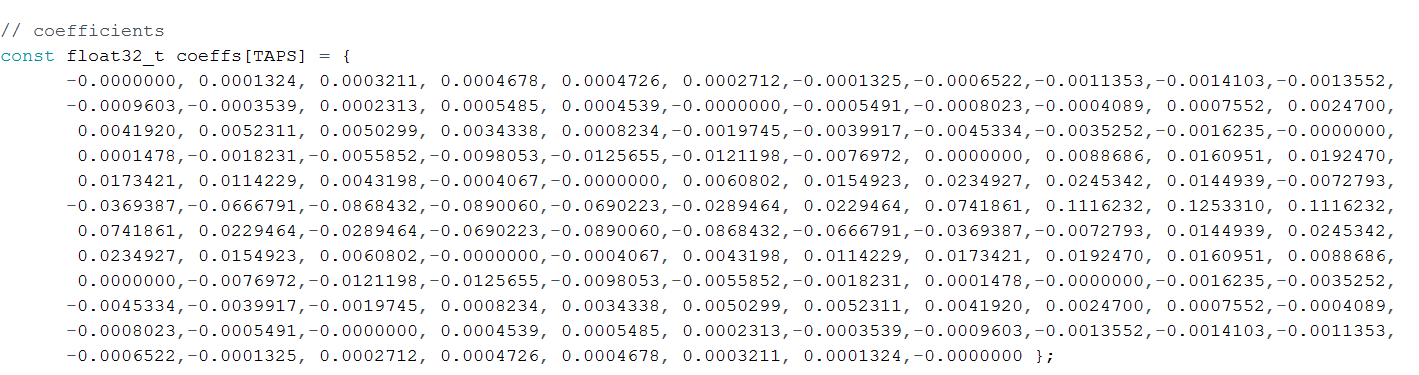
is the order of your filter + 1. Since we made a 128-order filter, our TAPS = 129. If your filter was higher order (or lower) make sure you change this.

TOTAL\_SAMPLES:

Remember N from section 1? That’s where this goes. We are going to filter 200 samples at a time, and this helps us define how many loops it takes to filter the entire file in chunks of 200 samples; (870400/200 = 4,352 loops).

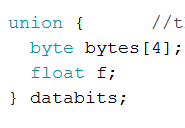
BUFFER\_SIZE:

This is the number of samples that are being filtered at a time. We filter data in blocks, because of the Teensy’s limited memory. Notice that the buffer size is a factor of the total number of samples. So, if you are filtering a different file than the one provided, make sure you change the TOTAL\_SAMPLES and BUFFER\_SIZE accordingly.

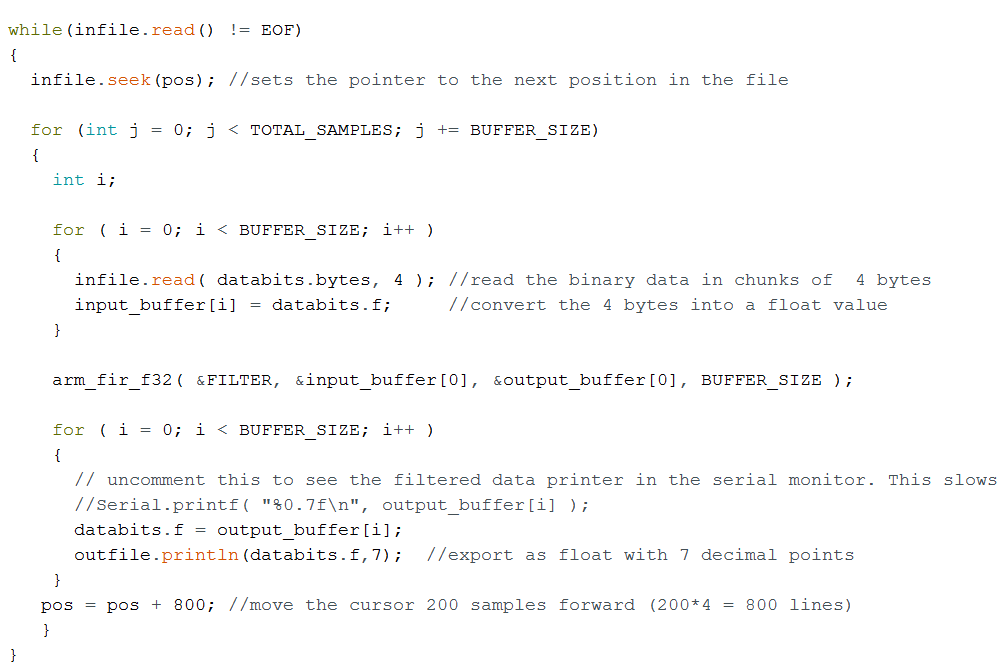


coeffs[TAPS]:

Remember the “coeff.txt” file? This is where that data goes. This was arranged to look neatly… have fun with that.



The binary representation of a floating-point number is 4 bytes. This union allows us to read samples, 4 bytes at a time, then cast them into floats.



This is where the filtering takes place, so let’s run through it.

* Read the input file until you reach the End-of-File marker
  + Set file cursor to position = pos and start reading from here. Pos is initialized to zero at setup
  + Loop for TOTAL\_SAMPLES/BUFFER\_SIZE (870400/200 = 4,352 loops).
    - Copy 200 samples into the input buffer
      * Samples are brought in 4 bytes at a time then converted into floats before being saved in to input buffer
    - Filter 200 samples and write them into the output buffer
    - Write the filtered data, one sample at a time, from the output buffer into the output file as floating point values with 7 decimal places.
  + Move the file pointer to the beginning of the next 200 samples so that they may be filtered.
* End of File reached, stop filtering. Write “Finished!” on the serial monitor.

This filtering process takes about 1 minute. Of course, the bigger the file being filtered, the longer it takes to filter.

Now we go back to MATLAB and listen to the filtered file. The filtered data from the Teensy should match the output file generated in section 3.

**Section 5: Reading back the filtered data**

Now that the Teensy is done filtering, remove the SD card and open it in your PC. Copy the “FILT.TXT” file from the SD card and place it in your working directory, then ruin the filtImport.m script.

filt = textread('FILT.TXT','%f');

filt = filt';

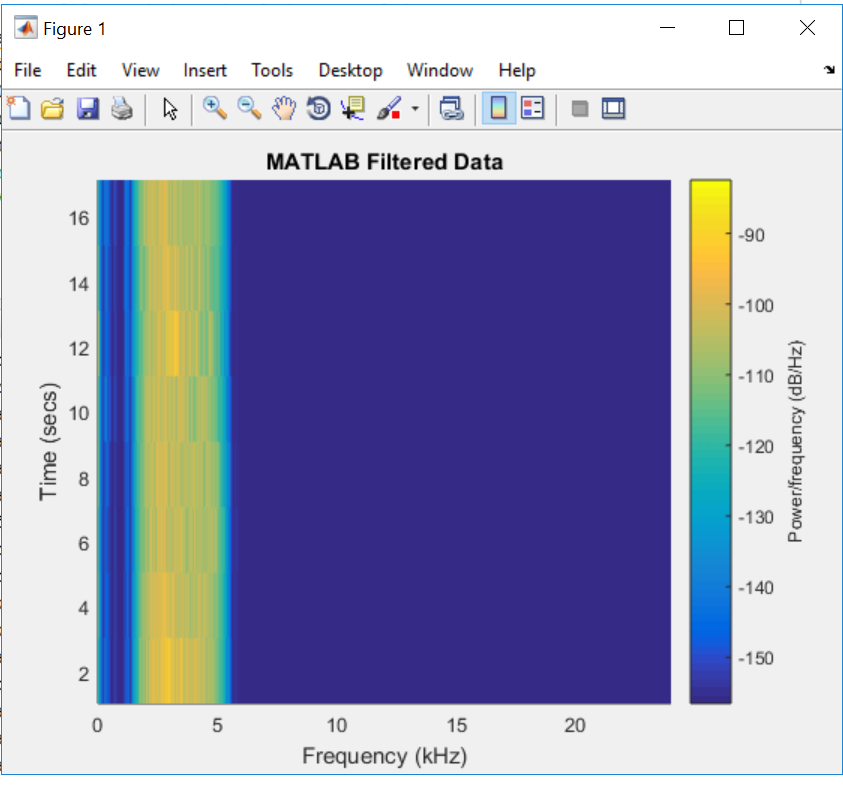
fs = 48000;

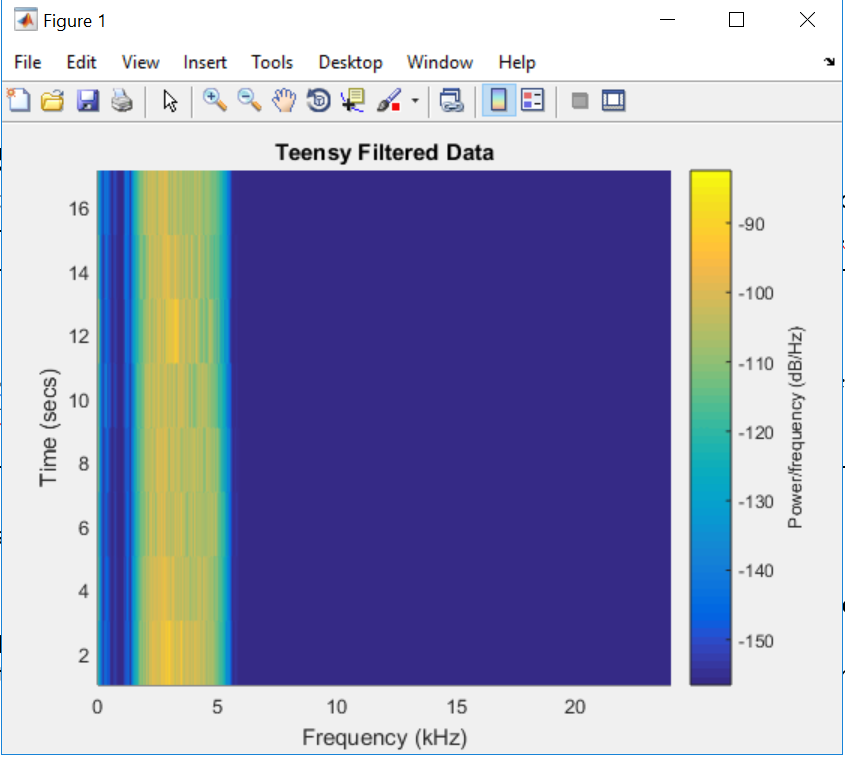
spectrogram(filt, [],[],[],fs), title('Teensy Filtered Data');

audiowrite('filtered\_birds.wav',filt,fs);

This code should:

1. Read the “FILT.TXT” file into the filt variable.
2. Print a spectrogram that should look identical to the one generated in section 3.
3. Export a wave file called “filtered\_birds,wav”

If you want to hear what this data sounds like use . The reason for the 2\* is because the sound is low and hard to hear.



And there you have it. You replicate this with any other wave file, all you need to do is change the appropriate parameters; sample size, buffer size, and filter parameters (low pass, high pass, band stop, cutoff frequency, etc.)