Nolan Anderson

Dr. Jovanov

CPE 381

19 April 2021

# Programming Phase 2 Report

## 1. Introduction

Phase 2 of the Real-Time processing assignment is to remove the noise that was implemented in phase 1. To do this, we will need to implement filter values in MATLAB. These filter values will be used in a C++ program (see zip file) to remove this noise. We will then analyze the dominant spectral component for the modified and original input files. This will be crucial in understanding if our filter implementation is correct.

## 2. Matlab Filter Components

This section will answer questions 2a, 2b, and 2c.

### a. Plot of filter characteristics (magnitude and phase) for both filters/sampling frequencies

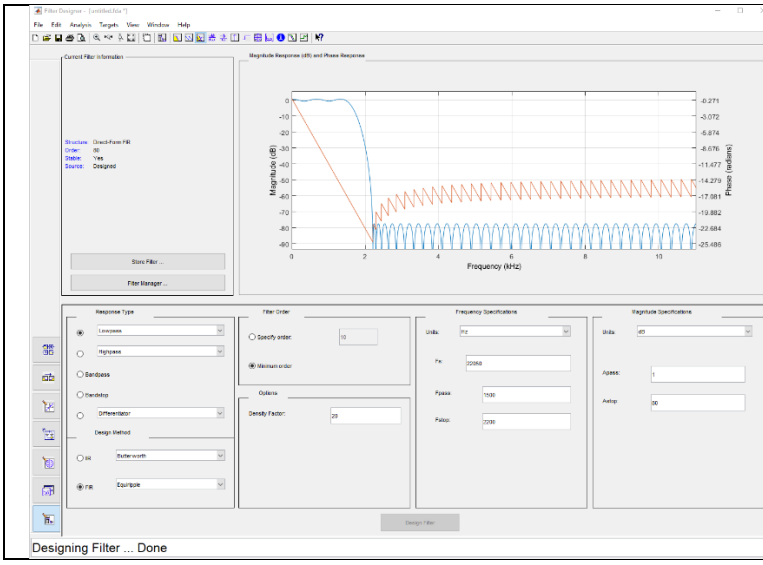


Figure 2.1: Filter Design for 22050 Hz

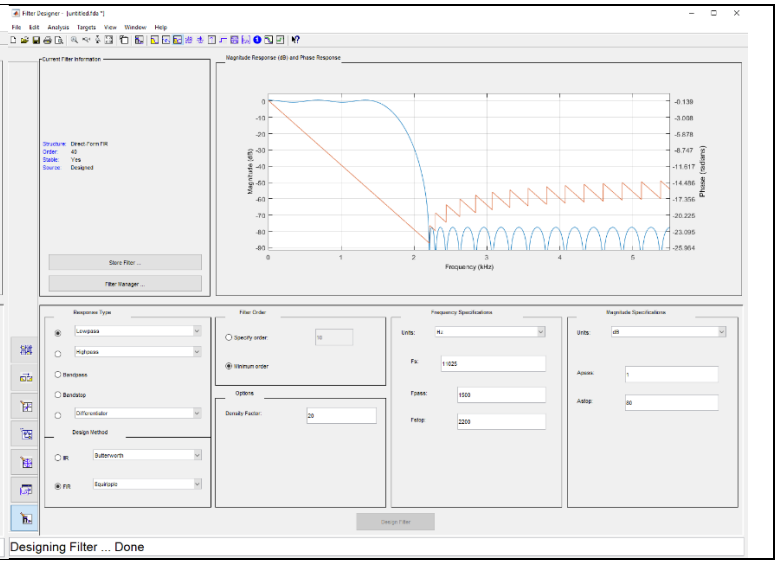


Figure 2.2: Filter Design for 11025 Hz

Both of the filters have a Fpsass of 1500 and Fstop of 2200.

### b. Type of filter (FIR/IIR) and reasons for choosing that filter.

I chose FIR Equiripple filtering for this portion of the assignment because we do not need to worry so much about performance, and accuracy is much more important for us. Continually, IIR cannot implement linear-phase filtering like FIR can. This means that we do not have a phase shift over the frequency band. Lastly, it is simply more accurate. IIR is meant to be used on the

fly and quickly change values, which we do not need since our sound file is very short. Overall, we use FIR instead of IIR for accuracy over performance.

### C. Organization of processing; how do you perform your processing in C/C++.

I have inserted many comments in the Anderson_N_Phase2.cpp file, but I will describe it in more detail here. The first step is to create a header file for the two filter coefficients, see filter_11025.h and filter_22050.h. These values will be used to process the left and right channels of the wav file. To do this, we first open the input files and check for errors of the sample rate. We assign the values to the left and right channels and then begin looping through the file. This is where the while loop comes into play. We read each sample from the input file, and if the iterator is even, we assign this sample to the left audio and vice versa for the right audio. Inside assigning the left and right audio, we copy the value and then call the filter function. This function is where we do the processing. Essentially, we add the values that we are taking away by using the filter. That is what the for loop is for. Once we are done with the filtering function, we write it to the output wave file called Anderson_N_filt.wav. We then output the information found and the time to process. Overall, the basic idea is you are looping through the left and right channels and filtering out the noise in both.

# 3. Input / Processed WAV File Spectrum

This section will display the dominant spectra of the input and processed files.

## 3.1 Input Spectrum

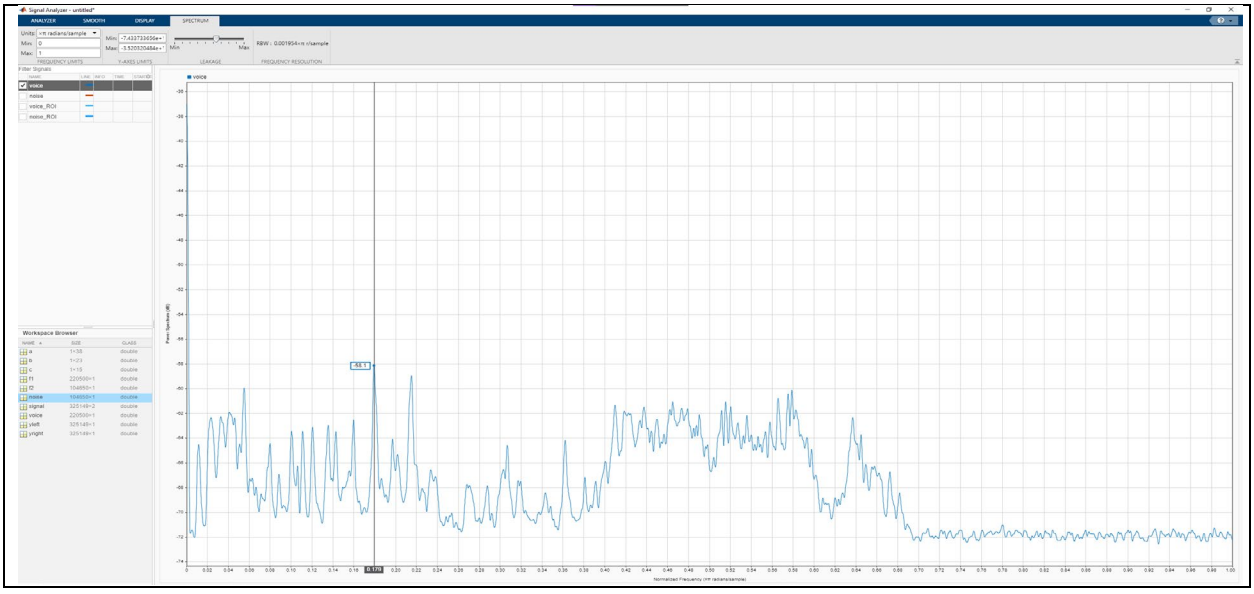

Figure 3.1.1: Spectrum of WAV Input file for t < 10s – Value is -58.1

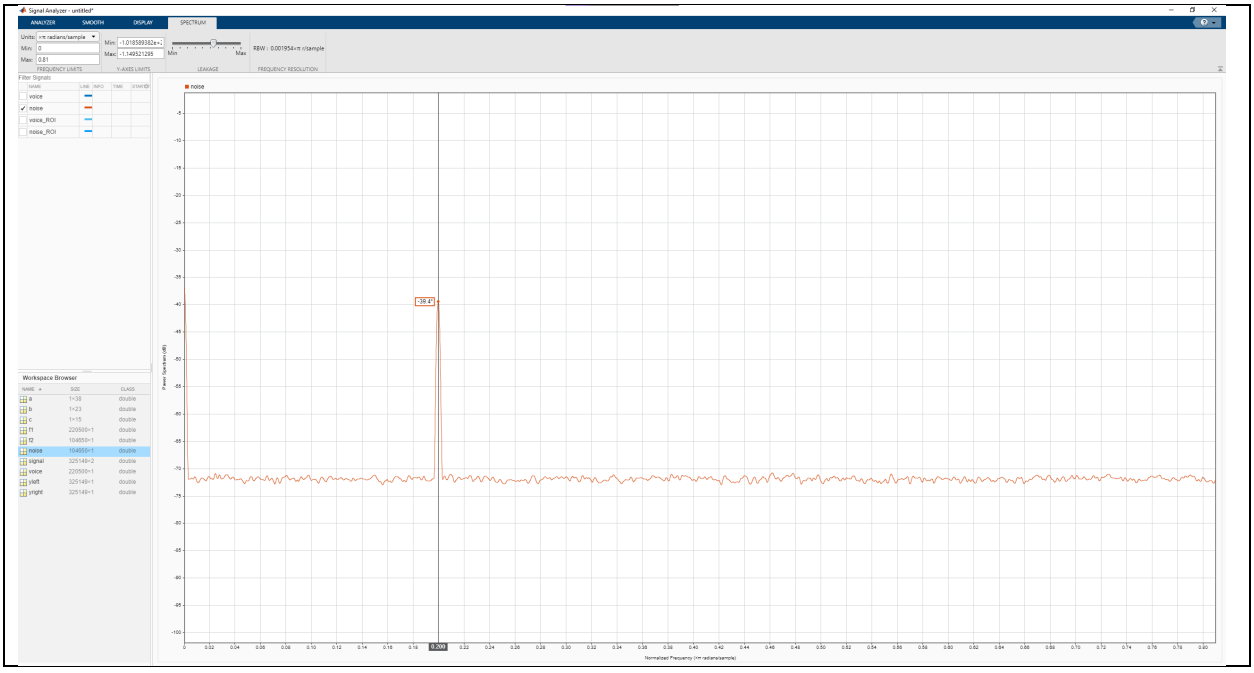

Figure 3.1.2: Spectrum of WAV Input file for t > 10s – Value is -39.4

## 3.2 Processed Spectrum

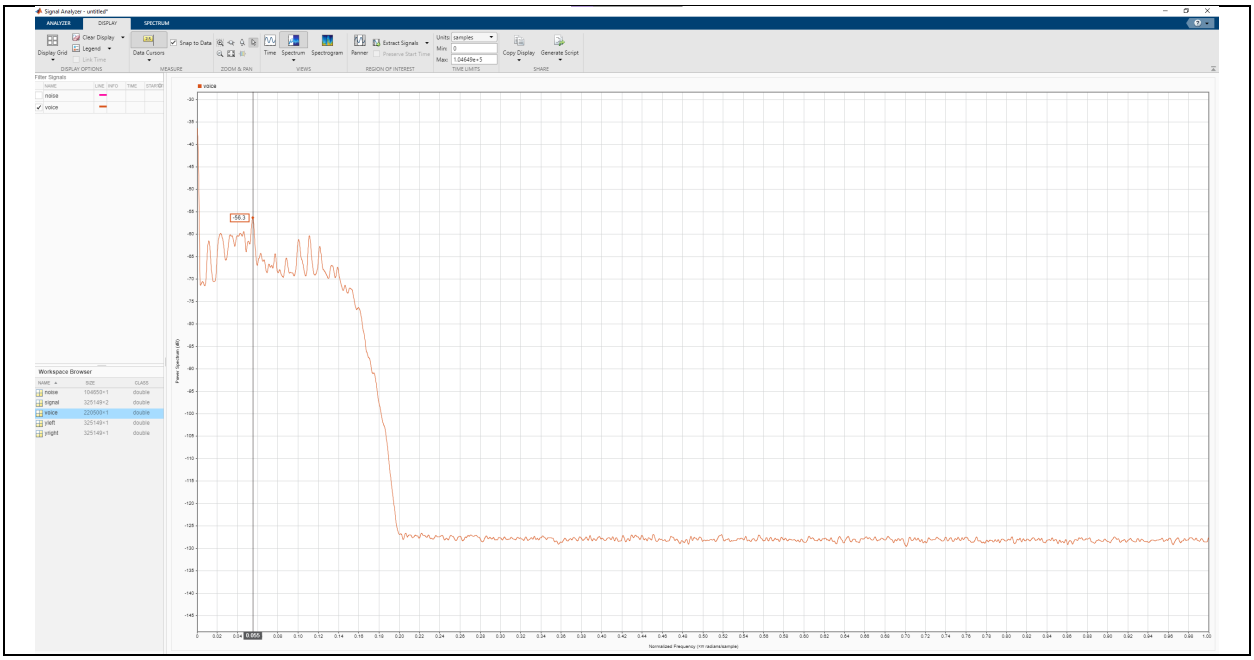

Figure 3.2.1: Spectrum of WAV Processed file for t < 10s – Value is -56.3

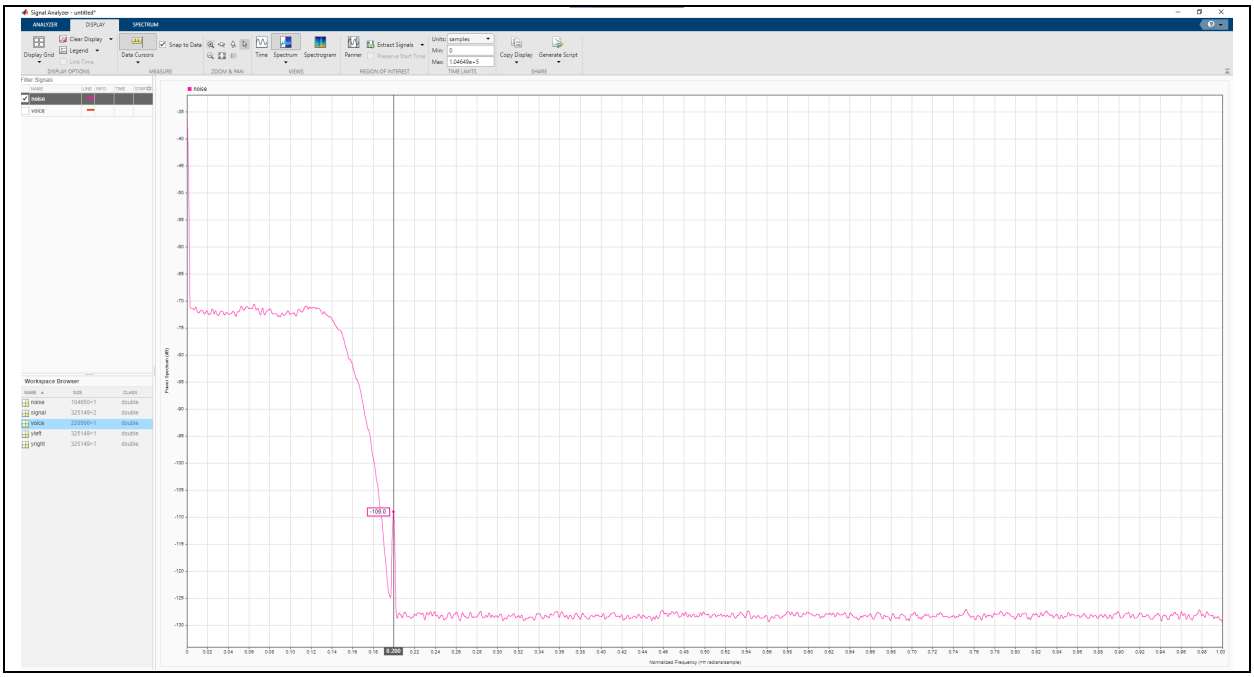

Figure 3.2.2: Spectrum of WAV Processed file for t > 10s – Value is -109

## 4. C++ Performance

For real time performance to be possible, the processing time must be less than the sampling time. The sampling time for this program is 45.35 µs and a processing time of .615 µs. So yes, this program could work in real time.

Sampling frequency (Fs) = 22050 Hz (found from wave file)

Sampling Time (Ts) = 1 / Fs = 45.35 µs.

Number of Samples: 325149

Time = 0.11 Seconds

Processing time (Tp) = Time / Samples = 0.2 / 325149 = .615 µs.

## 5. Lessons Learned

This project was much more intuitive, and personally interesting, than phase one. I felt like I was genuinely figuring things out and not just adding noise to a wave file. Most importantly, playing around with different filter designs showed me how important accuracy is. If the values we're off, you really needed to change them to get it sounding just right. Continually, this phase genuinely helped me to better understand signals. Knowing what to input into the filter designer took some thinking and math, but once you did figure it out you realized how important that work was initially. Overall, I would say this portion of the project was very enjoyable.