Seth Arnold & Will Cass
Signals and Systems Final Project

Introduction

This report will summarize our groups methods for creating the Guitar Tuner and our findings from completing the project. The report is separated into 5 parts:
Part 1 will discuss the creation of the DFT for a single sinusoid using the hamming window. Effective use of the hamming window will be necessary to achieve accurate results throughout this project.
Part 2 will discuss the creation of the DFTs for 2 sinusoids; both of which are plotted onto a single graph. Through varying the frequency, amplitude, number of samples, and zero padding we explored the effects these variables had on the plots.
Part 3 will see us computing the DFT of a real guitar signal. This is the first time we will compute the DFT of a real musical note. The plots created will be useful later for confirming that the synthetic signals we create appear realistic.
Part 4 includes the creation of synthetic guitar signals. By creating synthetic signals, we can generate signals for a large number of different frequencies very quickly. We can use these synthetic signals in Part 5.
Part 5 is the creation of a guitar tuner which will use the DFT of a signal to determine the frequency at which the signal oscillates. This is achieved through the creation of several filters which are applied to the input signal.

Part I. DFT of Single Sinusoid and the Use of the Hamming Window

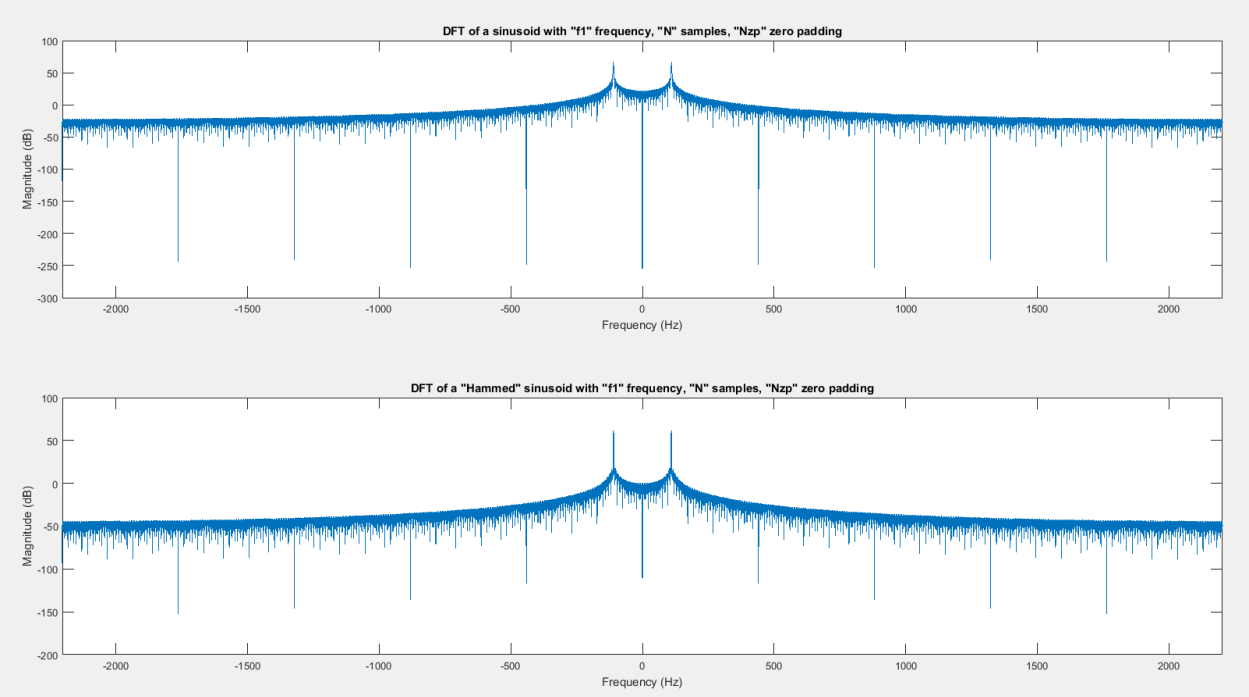


Fig. 1 DFT of a sinusoid with 110 Hz, 4410 samples and 1000 zero padding

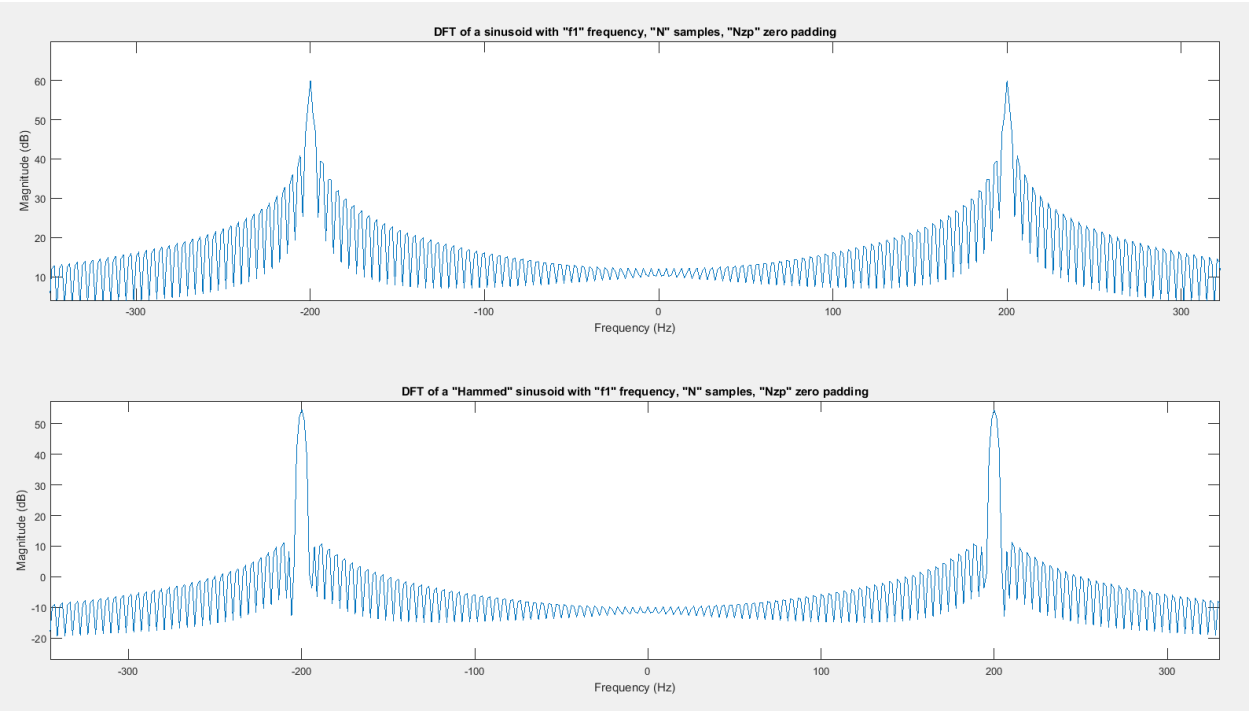


Fig. 2 DFT of a sinusoid with 200 Hz, 2000 samples and 1000 zero padding

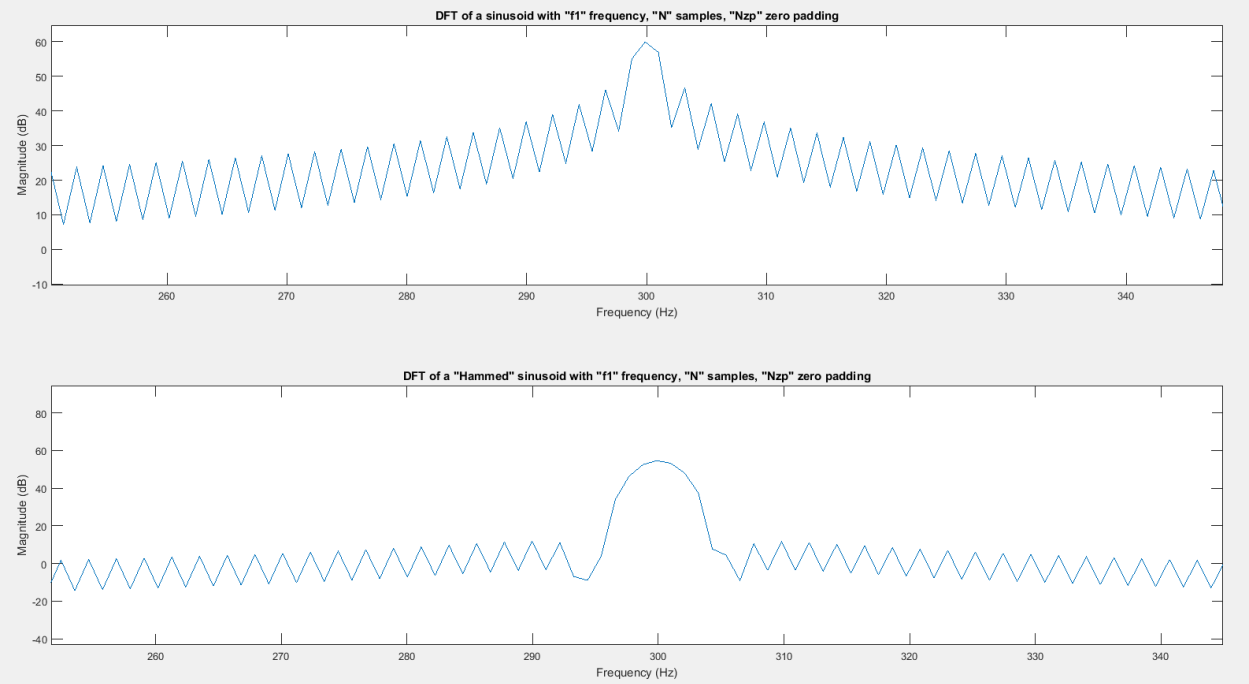


Fig. 3 DFT of a sinusoid with 300 Hz, 2000 samples and 2000 zero padding

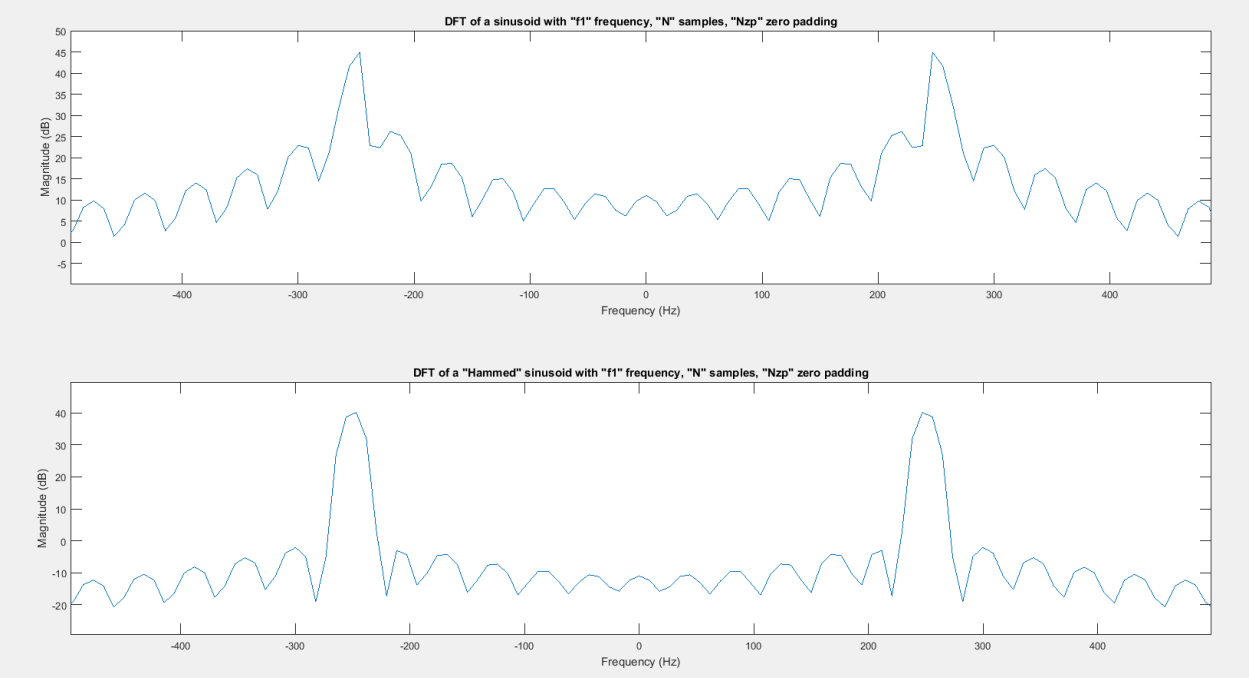


Fig. 4 DFT of a sinusoid with 250 Hz, 400 samples and 100 zero padding

In Part 1, we created a function which generates a single sinusoid and computes the DFT. This DFT was then plotted at various frequencies, samples and amounts of zero padding in order to see what effects these changes had on the DFT. From this, we could see that the number of samples and zero padding affected how smooth the DFT was. The number of samples also affected accuracy while a change in frequency

changed the positions where the mainlobe of the DFT was located. DFT results do not actually change as the number of samples is varied because the sampling rate is held constant. The number of samples and zero padding only affect the smoothness and accuracy of the plot.

Calculating a DTFT may seem like a challenge.
The DTFT is the summation an infinite number of terms, shown by Formula 1

Formula 1 $$X(\Omega) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n}$$

Therefore, it is impossible to calculate the DTFT from data, as an infinite number of calculations would be required. This is where the DFT comes in. In simple terms, the DFT is the DTFT evaluated over a finite set of points. In order to find the DFT, we can use Formula 2.

Formula 2

$$X(\Omega_k) = \sum_{n=0}^{N-1} x[n]e^{-jn\Omega_k} = \sum_{n=0}^{N-1} x[n]e^{-jnk\frac{2\pi}{M}}, \quad for \quad k = 0, 1, 2, \ldots, M-1$$

This finite method of computing the DTFT allows us to calculate the DFT from data. This also results in the DFT being in the frequency domain while the DTFT is in the time domain.

The DFT of a sinusoid should show a spike at the frequency equivalent to the sinusoid frequency. For example, Fig. 2 shows spikes at +-200 Hz, which is the expected result based on the input sinusoid. The high level of accuracy can be shown by zooming in on Fig. 3. It can be seen that the DFT is nearly exactly 300 Hz, which matches the frequency of the input sinusoid for the figure.

Hamming multiplies the time varying signal by the hamming amount which is generated to reduce the amplitude near the beginning and end of the signal and leave the middle unaffected. When you take the DFT of this hammed signal, this causes the non-peak values to be attenuated to a lower decibel level, while the peaks are left mostly untouched. This makes it easier to distinguish the peaks from any noise present.

The window doesn't really affect smearing and it lowers the magnitude of areas that are not peaks. The window also causes the mainlobe to be wider and the sidelobes to be lower.

Part II. DFT of Two Sinusoids

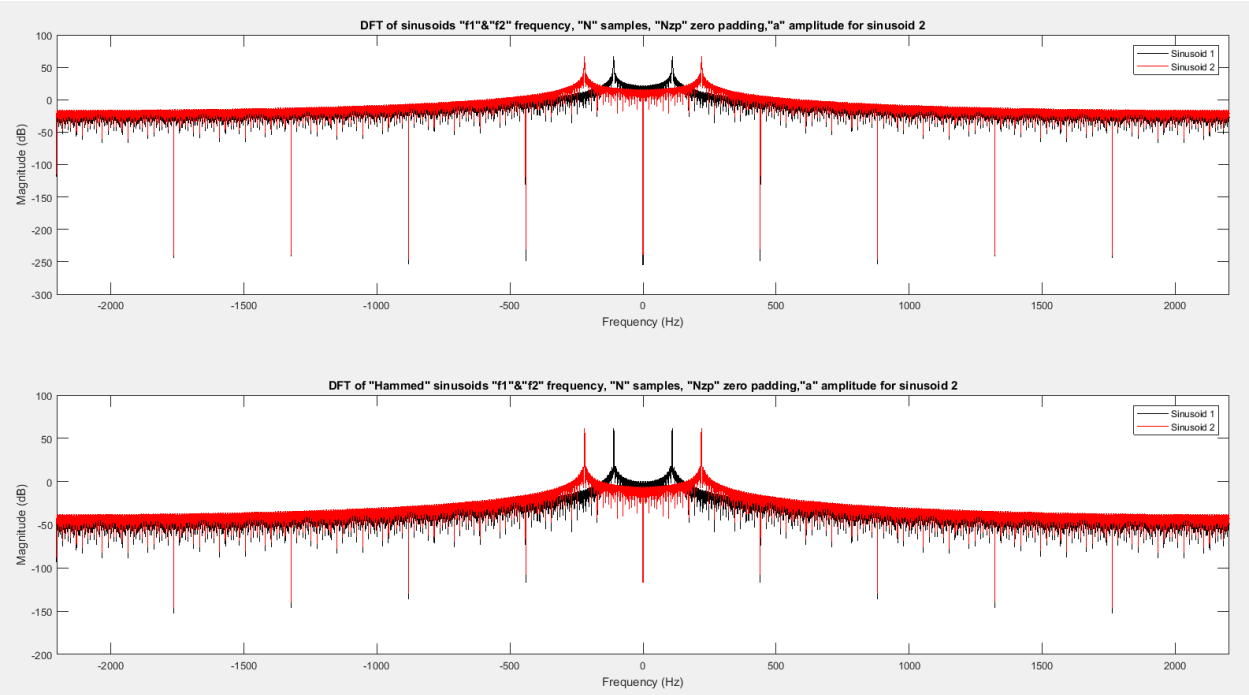


Fig. 5 DFT of sinusoids with frequency 110 (black), frequency 220 (red) & amplitude 1, 4410 samples and 1000 zero padding

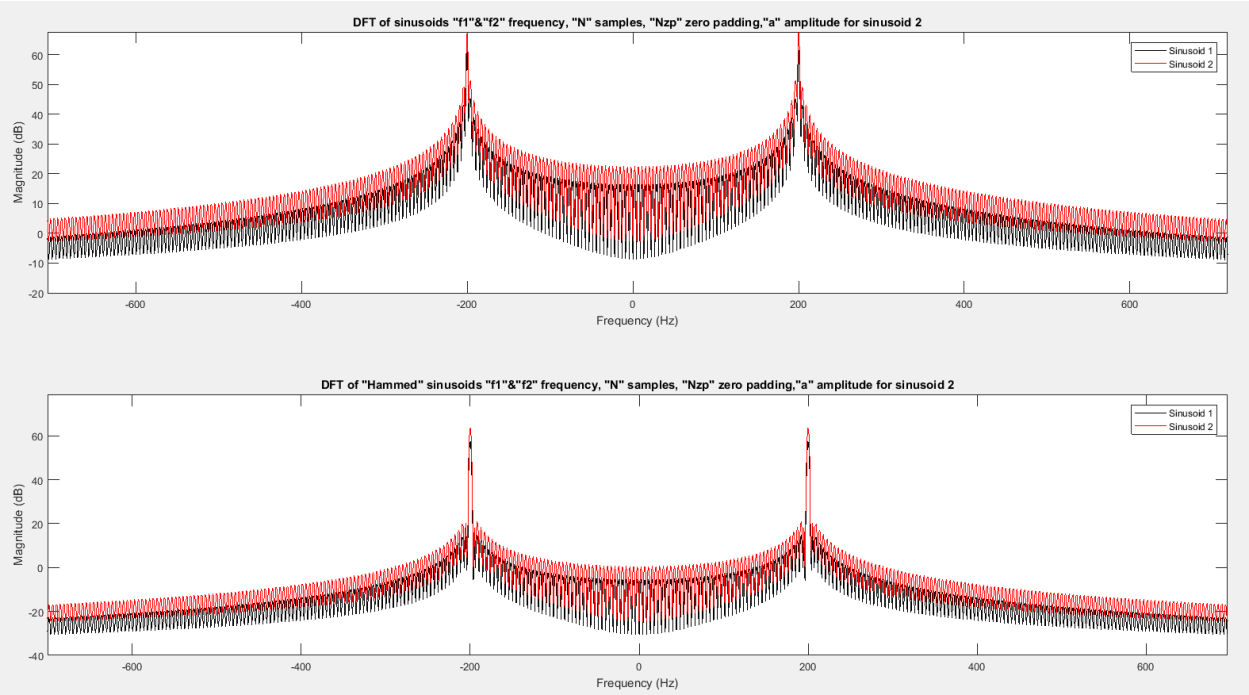


Fig. 6 DFT of sinusoids with frequencies 200 Hz, amplitude 2 (red), amplitude 1 (black) 3000 samples and 1000 zero padding

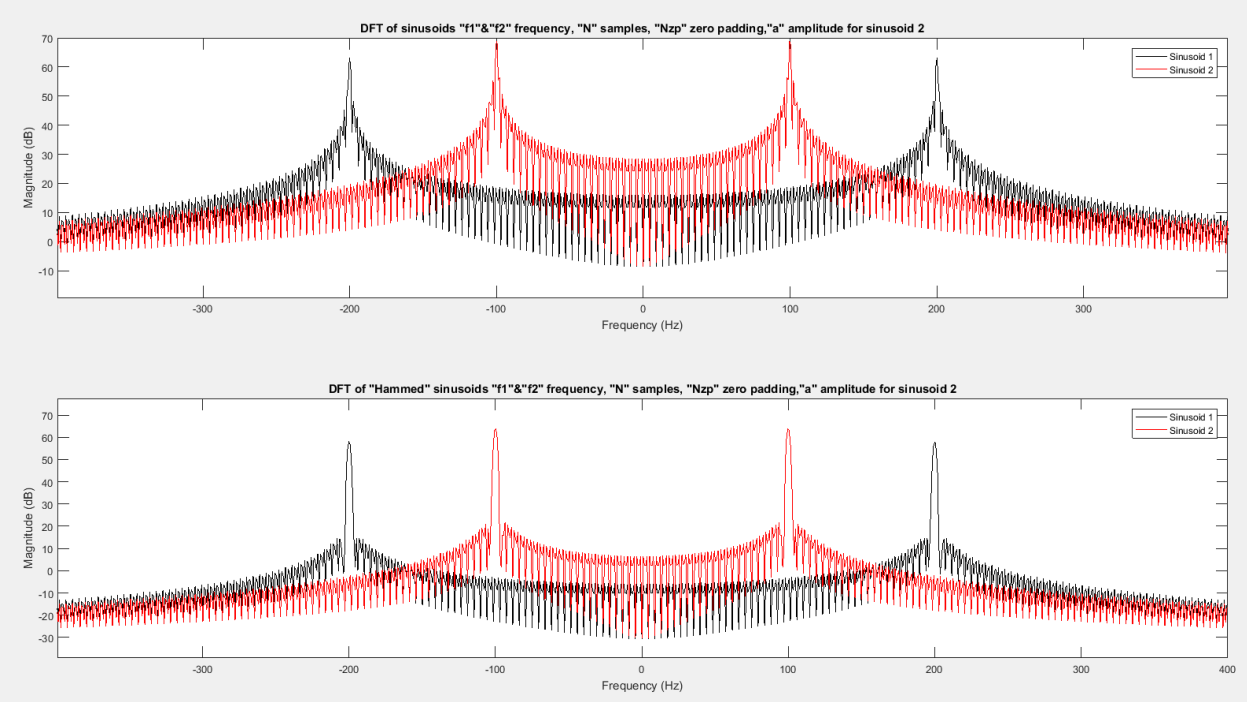


Fig. 7 DFT of sinusoids with frequency 200 (black), frequency 100 (red), amplitude 0.5 (red), amplitude 1 (black), 3000 samples and 2000 zero padding

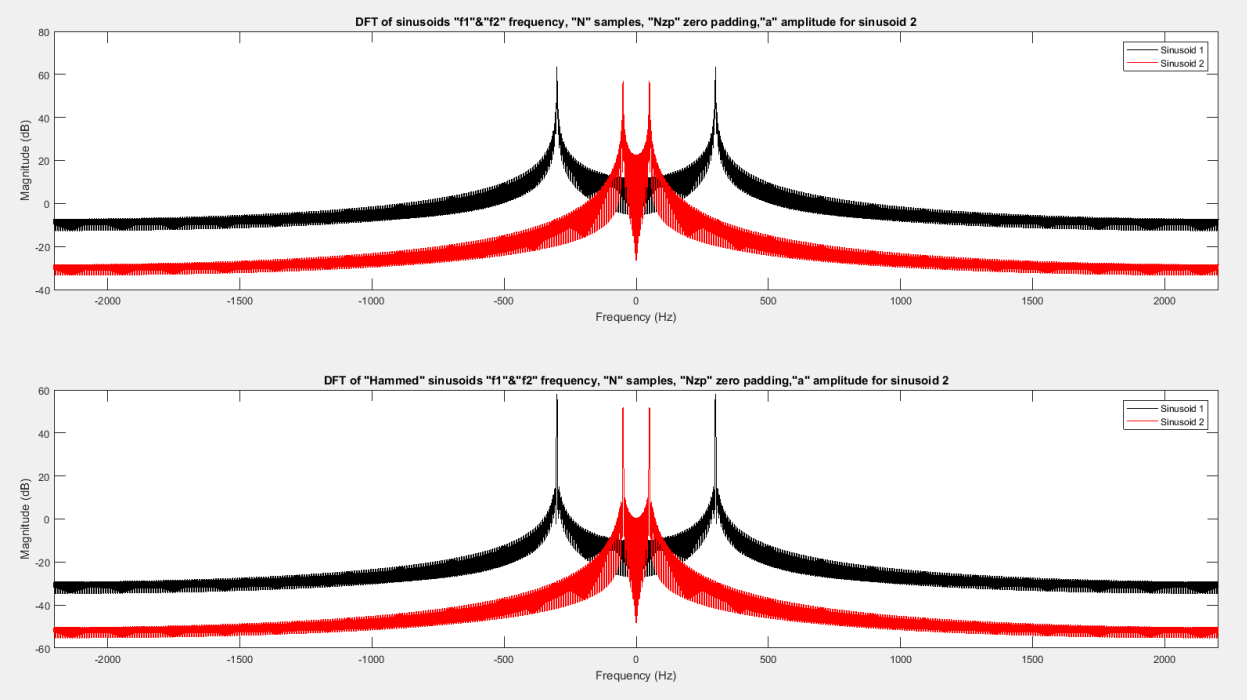


Fig. 8 DFT of sinusoids with frequency 300 (black), frequency 50 (red), amplitude 0.1 (red), amplitude 1 (black), 3000 samples and 2000 zero padding

In this part, we sampled two sinusoids at varying frequencies. Sinusoid 2 had variable amplitude, and we varied both samples and zero padding to observe the effects these changes had on the combined graph.

When both amplitude and frequency are different, it is easy to see the difference between the two sinusoids. When the amplitudes are different, but the frequencies are the same, both peaks are in the same position, but the sinusoid with the greater frequency will have a higher peak.

It becomes very difficult to see the difference between the sinusoids when they have frequencies close to one another and similar amplitudes. This is because there is overlap, so in the graph one sinusoid would cover the other for a portion of the graph.

The windowing makes the peak of the DFT more pronounced. This makes the difference between two signals easier to observe. Windowing therefore offers little improvement for noiseless signals, or on a plot with multiple DFTs where two signals which are separated in frequency but differ in amplitude. As shown in Fig. 8, these are already very easy to differentiate.

Part III. DFT of Individual Real Guitar Notes

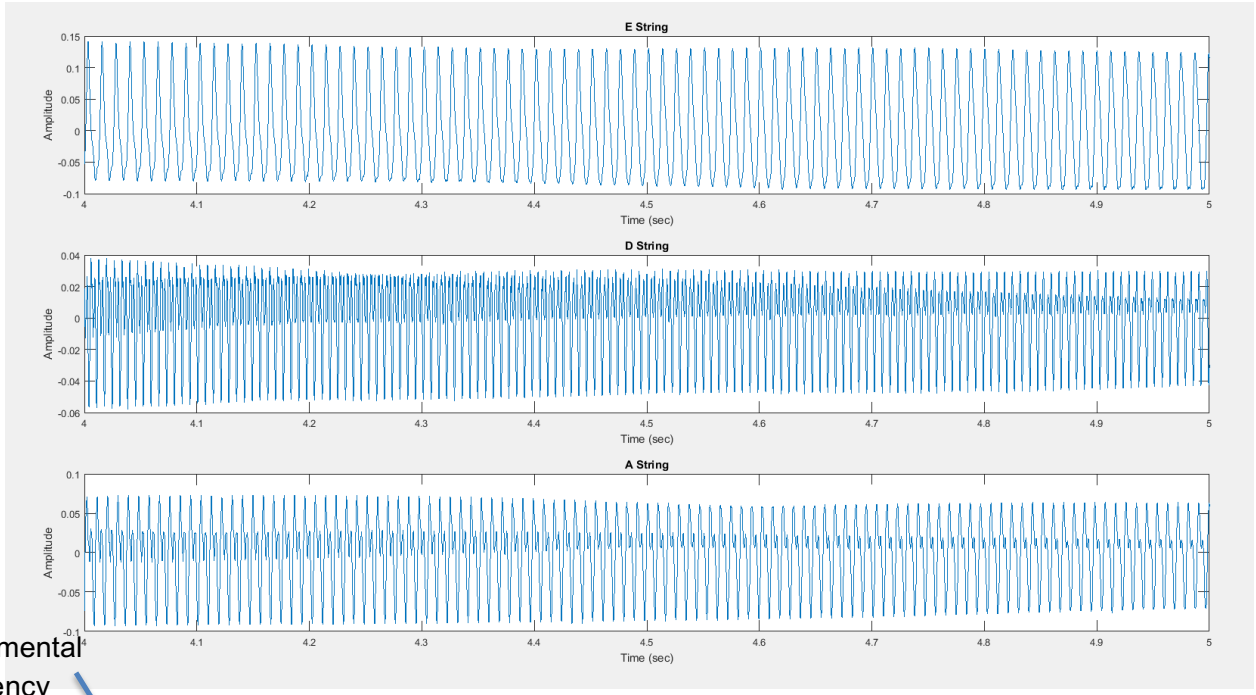
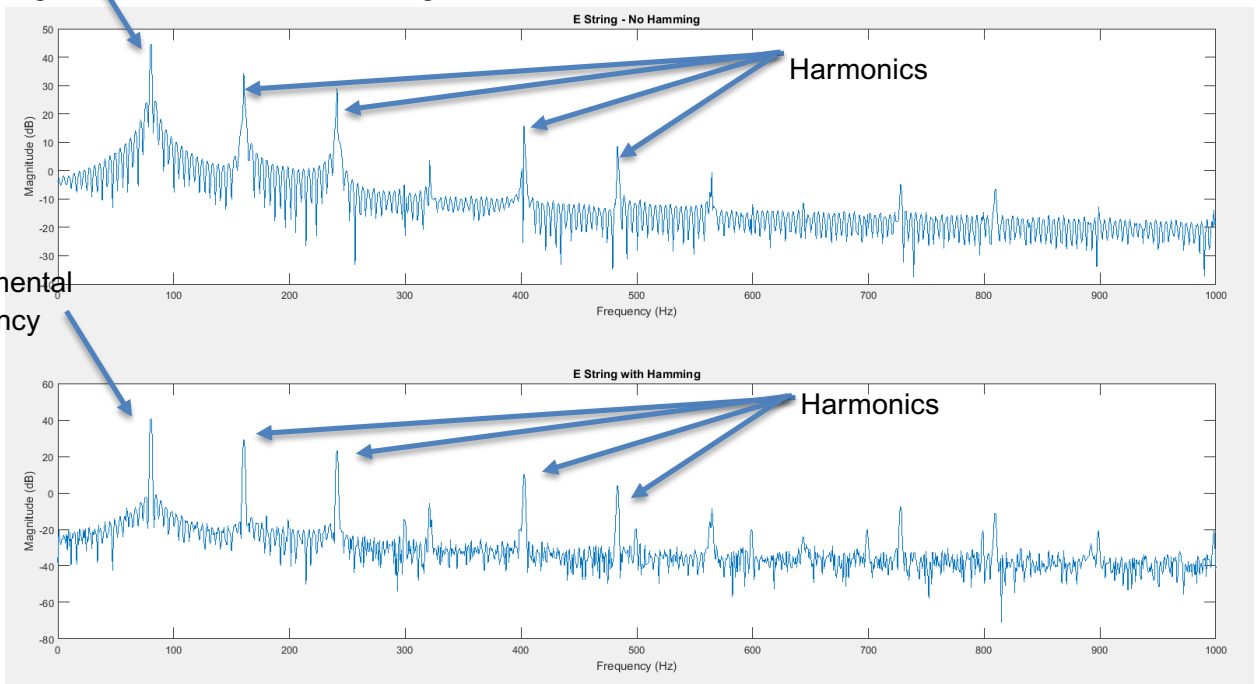


Fig. 9 Time Domain signals from 4 to 5 seconds



Fundamental Frequency

Fundamental Frequency

Fig. 10 DFT of E string with and without hamming

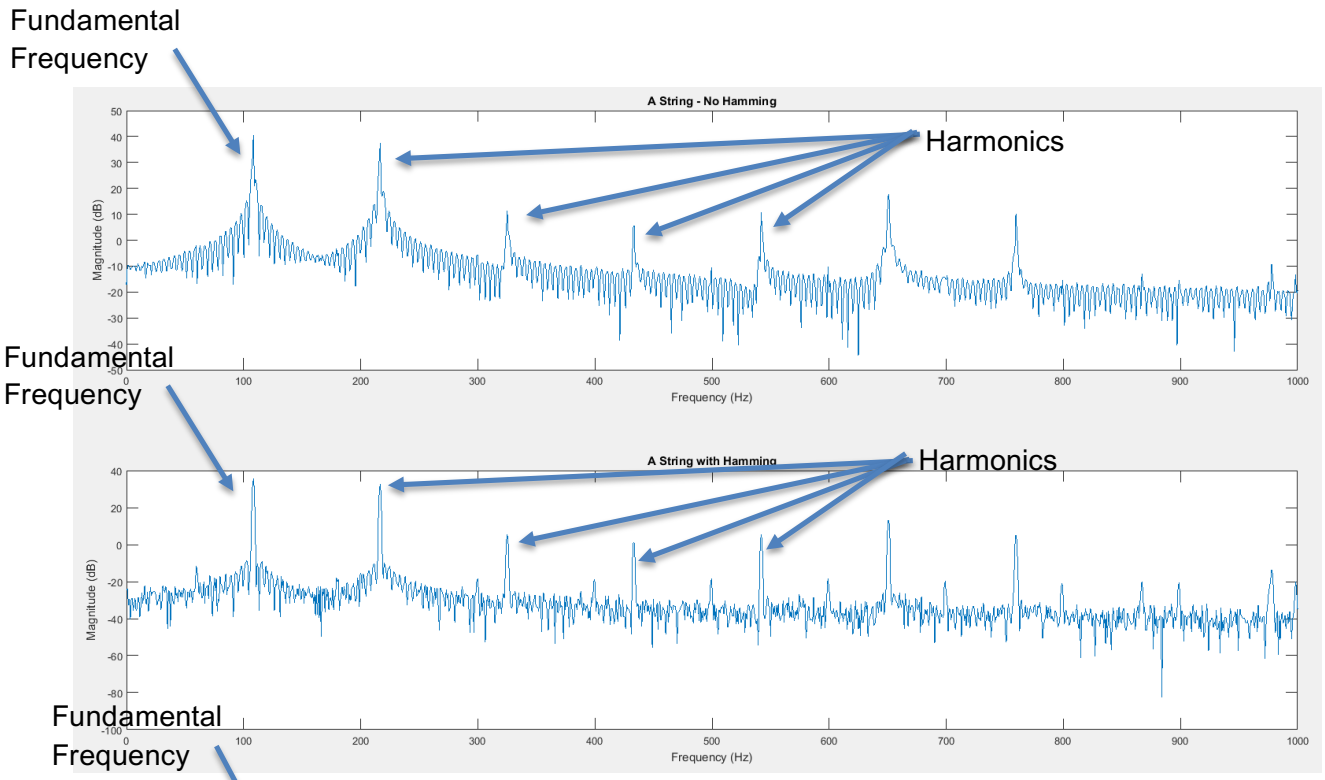


Fig. 11 DFT of A string with and without hamming

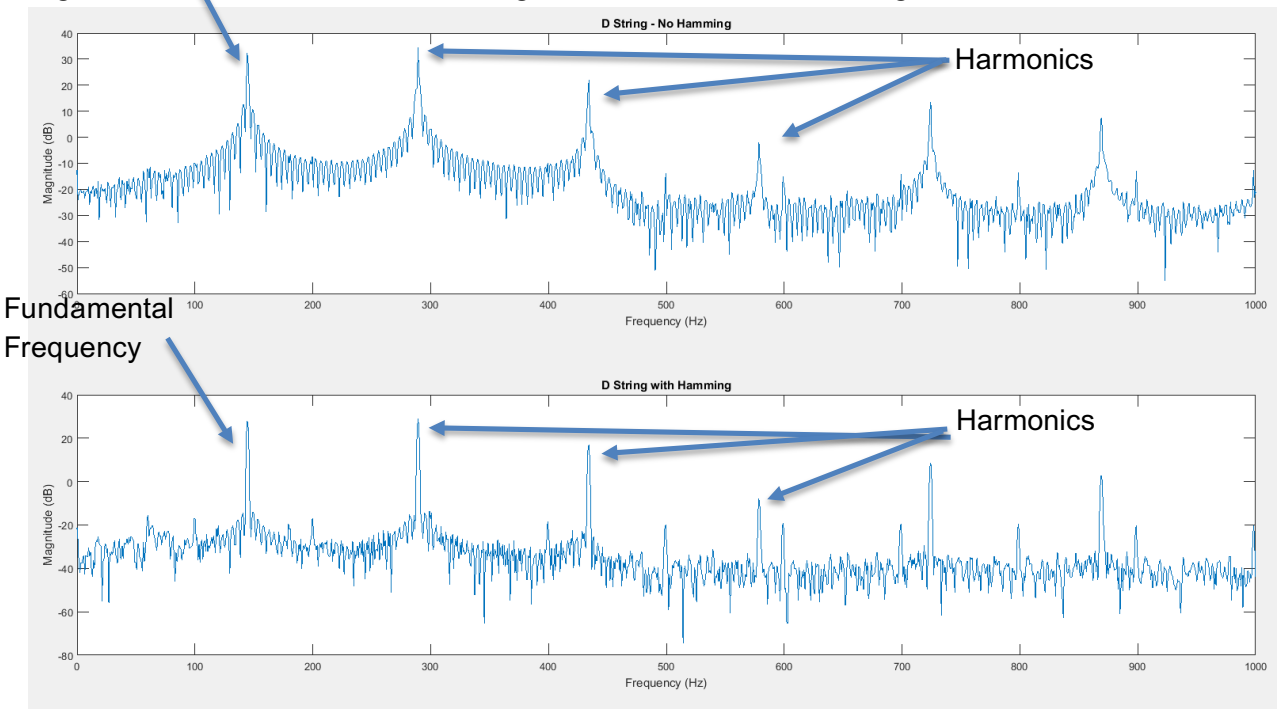


Fig. 12 DFT of D string with and without hamming

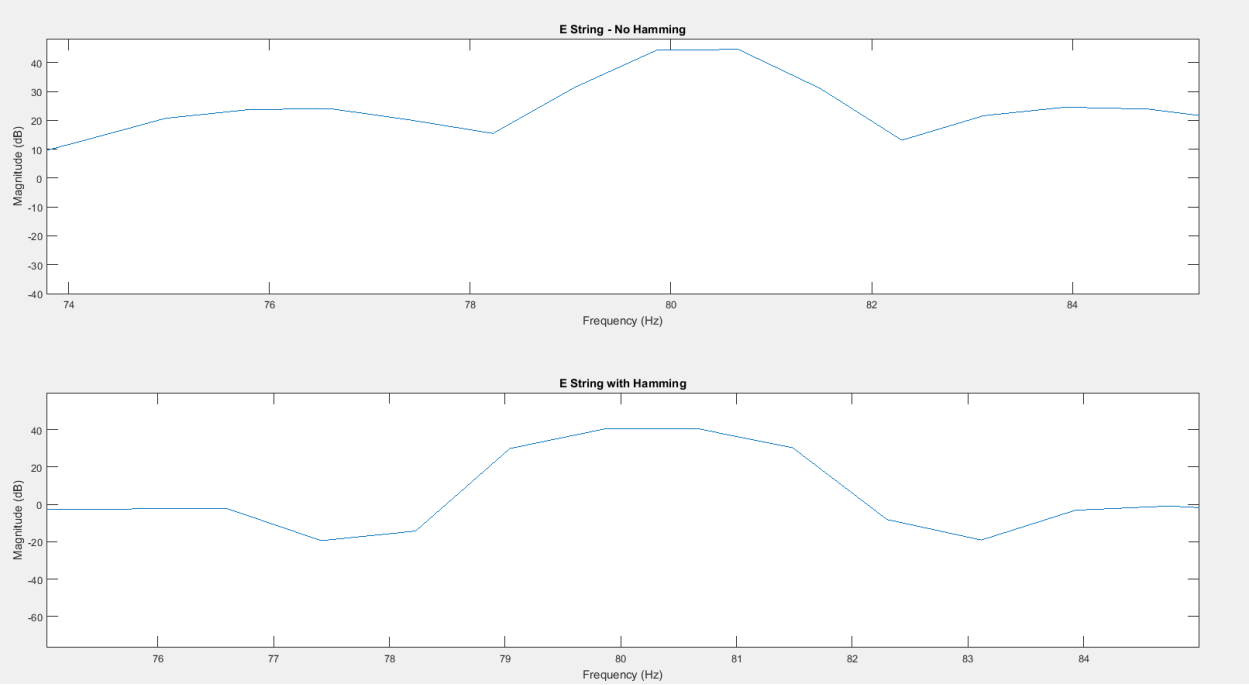


Fig. 13 Zoomed in view of real guitar E note

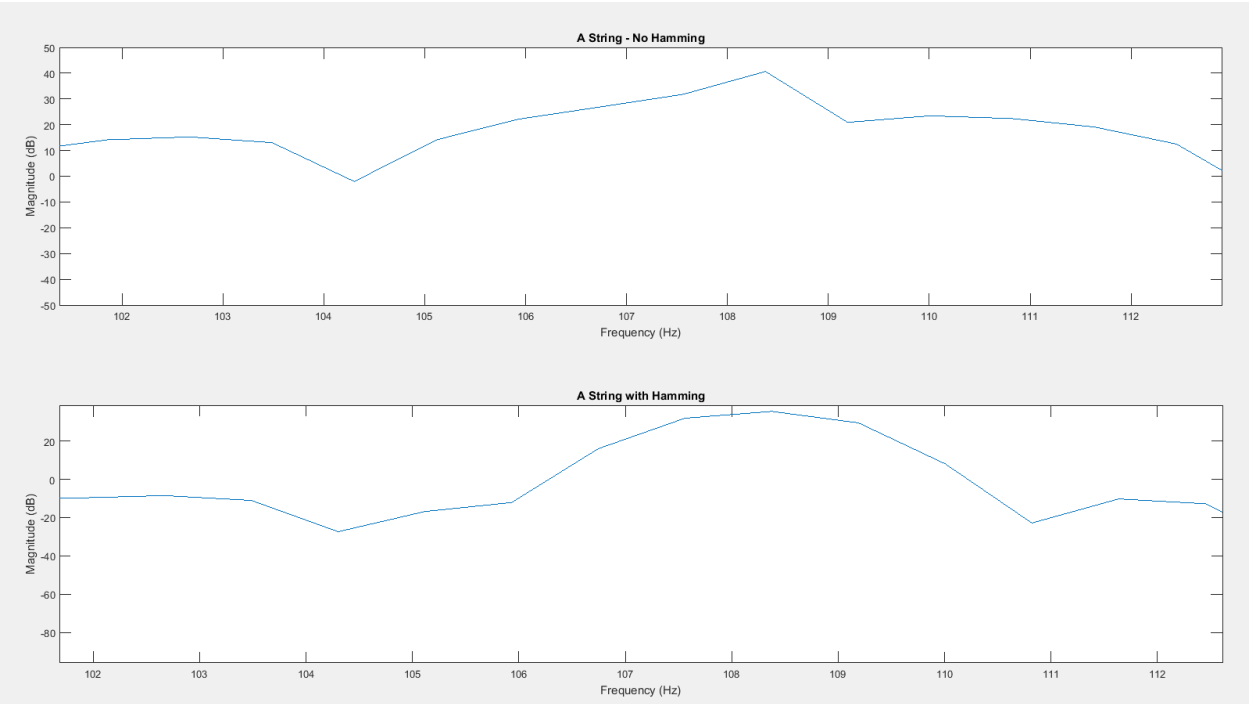


Fig. 14 Zoomed in view of real guitar A note

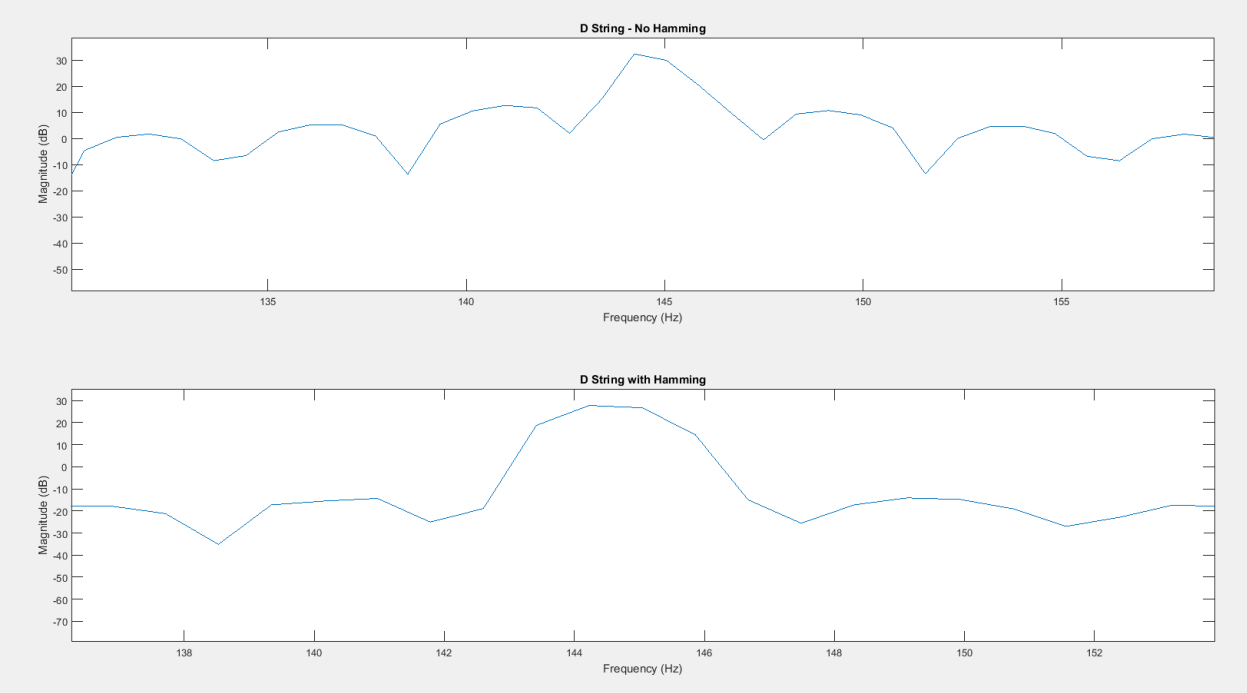


Fig. 15 Zoomed in view of real guitar D note

In Part 3 we took real guitar signals and found their DFT. The DFT of a CT sinusoid will show peaks where the sinusoid has its fundamental frequency and its harmonics. This is absolutely the case here.

In this part, we are able to follow a signal from CT, to DTFT and finally to DFT. When the guitar string is plucked in the real world, the signal it produces is a CT signal. This CT signal is then recorded and stored digitally. In this case, the sampling rate is 4410 Hz, and therefore as long as the signal is under 2205 Hz, the signal will not be under sampled. The DTFT of the signal is the data which we are provided. We take this DTFT and calculate the DFT, changing the domain to frequency. This allows us to identify the frequencies within the signal, and find noise sources. This can be seen in Fig 10-12.

The plots of the time signal look periodic as seen in Fig. 9, the repetition can be clearly seen. This allows us to find the Fourier Transform which is plotted in Fig. 10-12. The multiple spikes seen above can be attributed to the harmonics that a guitar produces. The Hamming window makes it easier to see the peaks we expected because it attenuates some of the noise, making them more defined.
It can bee seen that the guitar was not perfectly in tune. The E note appears to have a fundamental frequency of 80 Hz, instead of 82.41 Hz, so it is slightly flat. The A note appears to be 108 Hz, instead of 110 Hz, so it is also slightly flat. The D note appears to be 144 Hz, instead of 146.83 Hz, so it is slightly flat as well. The guitar was tuned flat.

There are peaks that our theory doesn’t predict. These occur at frequencies which are multiples of 100. This relationship would indicate that most of the peaks are the harmonics of a source of noise which is causing the initial 100 Hz signal. This noise could be caused by the guitar itself, the amp or the recording equipment. I believe that the 100 Hz spike is caused by the guitar itself vibrating.

Part IV. DFT of Individual Synthetic Guitar Notes

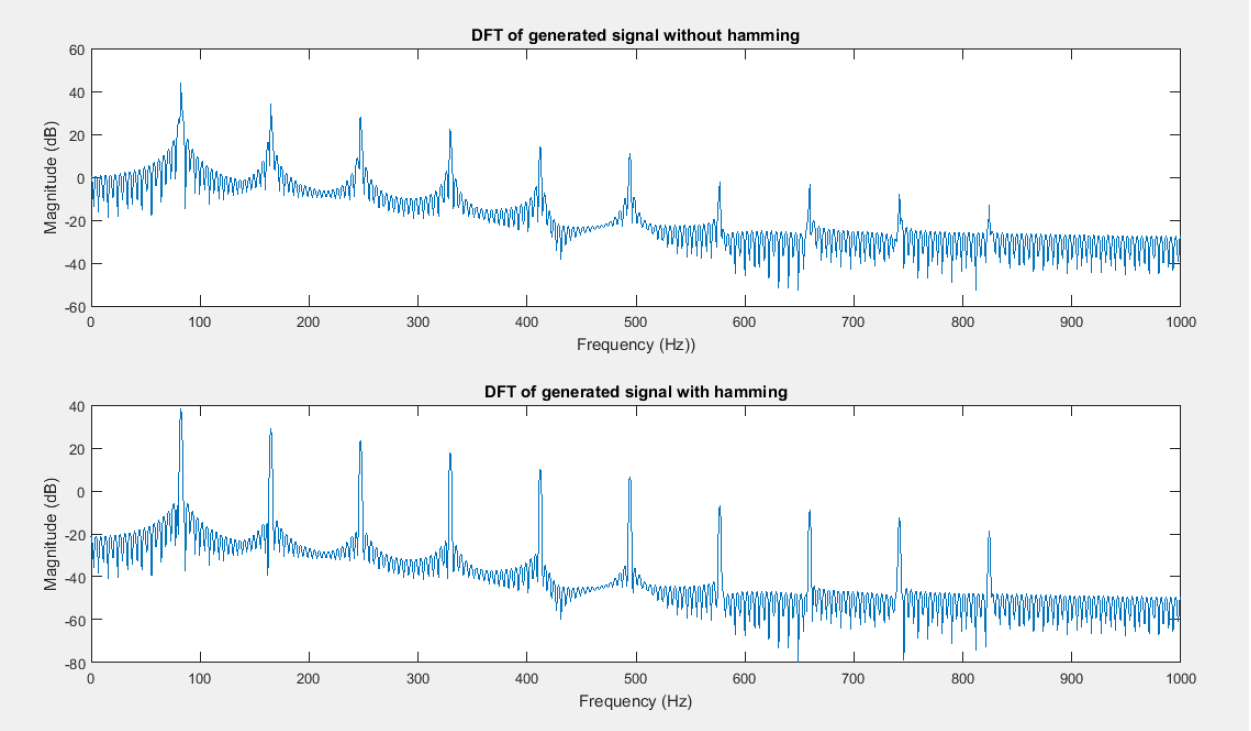


Fig. 16 Correctly tuned, synthetic E note, 82.41 Hz

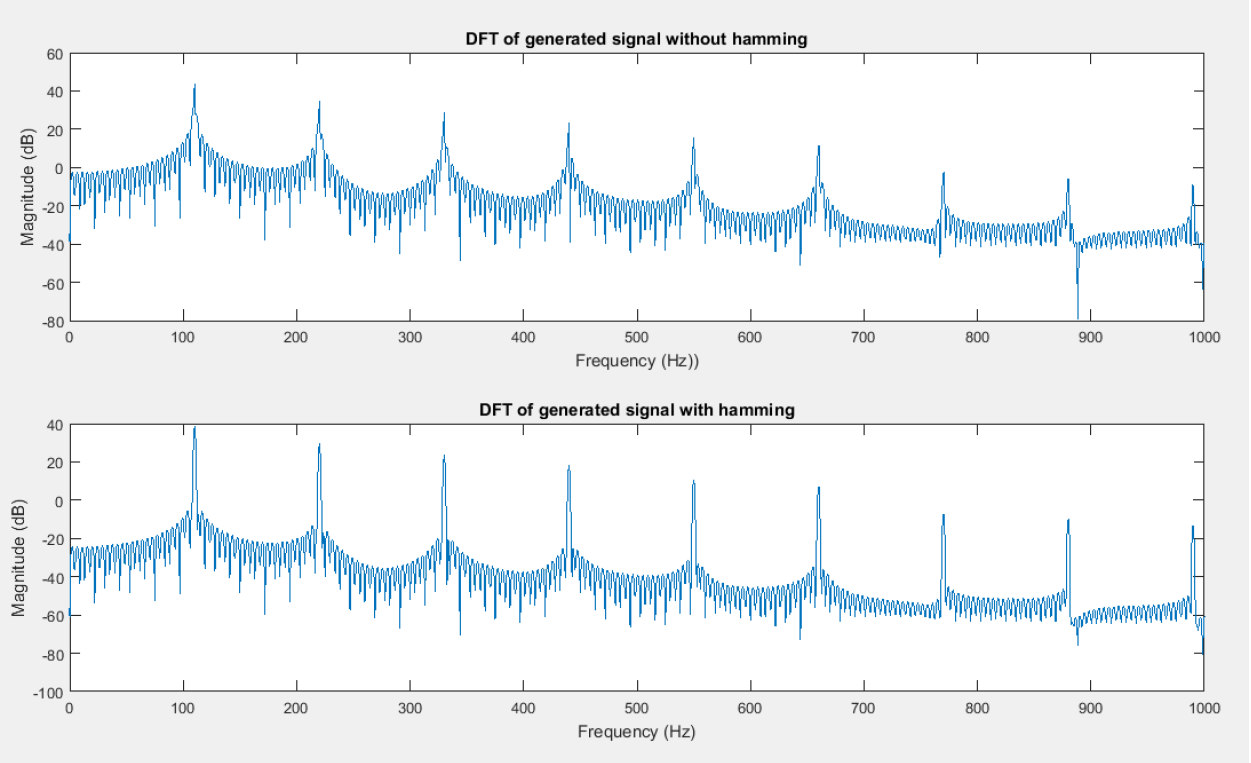


Fig. 17 Correctly tuned, synthetic A note, 110 Hz

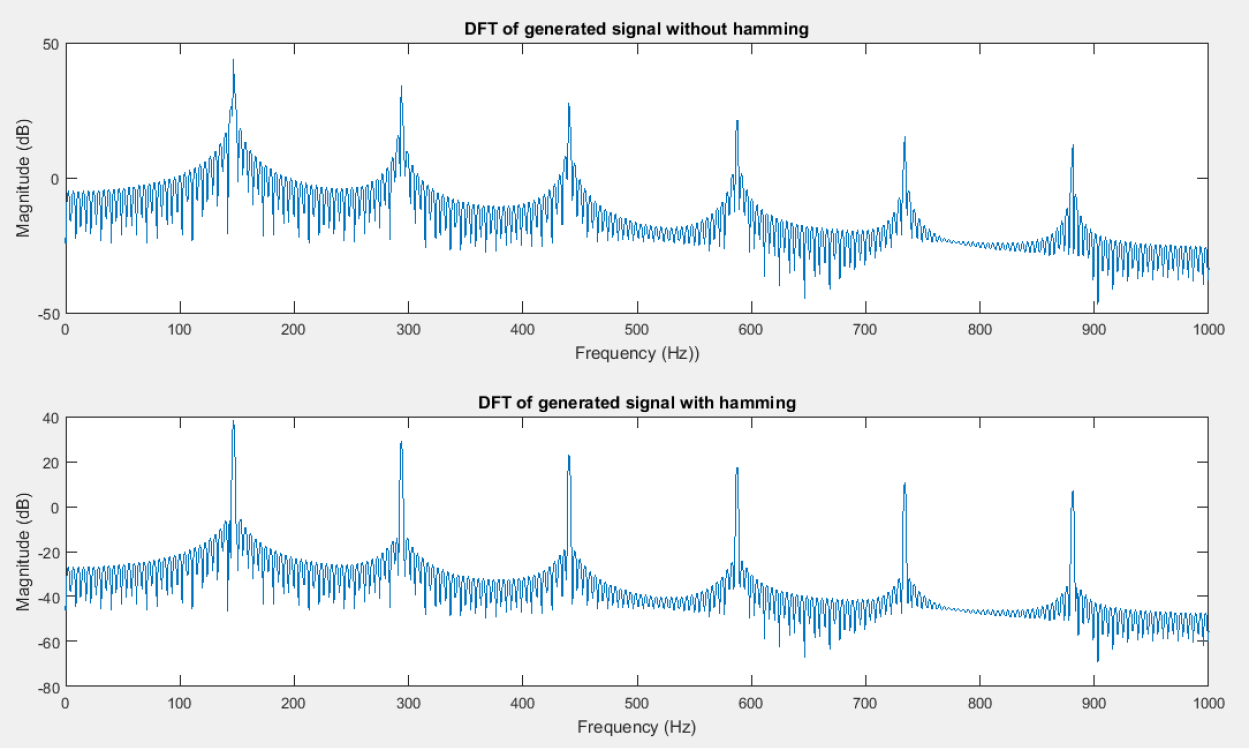


Fig. 18 Correctly tuned, synthetic D note, 146.83 Hz

In Part 4, we created synthetic guitar notes using provided harmonic amplitudes and randomized phases. We used these to create a sum of 10 sinusoids. The first of which had the highest amplitudes and the following ones, the harmonics, had progressively decreasing amplitudes. We then found the DFT and plotted the signal. This gave us a better idea of whether the signal was working correctly.

Comparing Fig. 15 to Fig. 11 shows that they are similar in pattern, with Fig. 15 being less noisy as it is a simulated signal. This shows that our DFT_Synth_Guitar is working correctly and produces accurate results.

The maximum frequency we can accurately create is half of our sampling rate, 4410 Hz, which is 2205 Hz. This is because any higher of a frequency would cause our sampling rate to be under sampled.

Part V. DFT-Based Guitar Tuner

Our part 5 accepts 6 inputs: x, the input sinusoid, b_E,b_A,b_D,lowpass and highpass, which are all filters generated by Tuner_FIRs. It returns 2 outputs: “fo_est”, the estimated frequency, and “band”, the frequency band the signal is inside.
The filters are applied to our signal and the filter that best matches the signal is determined by summing the squares and choosing the largest sum. The signal that matches a filter will be much larger as the filter is designed to pass only values within its range and so if the signals values do not match its passband it will give very small values. Once the band is determined, the program is done for lowpass and highpass bands. These bands return ‘low_freq’ or ‘high_freq’ as appropriate and ‘NaN’ for the estimated frequency. Otherwise the program continues, and estimates the frequency. The frequency is estimated by finding the largest frequency response of the DFT. The use of a hamming window is important for finding accurate results. In our use, we have ideal synthesized signals, so the hamming window does not have as large of an impact as it would on a real world signal. Real signals have noise, and as the hamming window is used to reduce noise, the hamming window would be vital to obtain accurate results with real world signals.

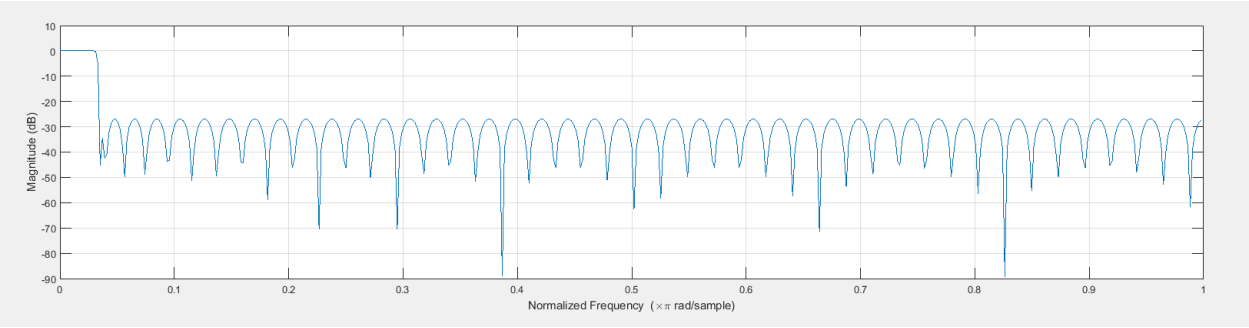


Fig. 19 Frequency response of the lowpass filter

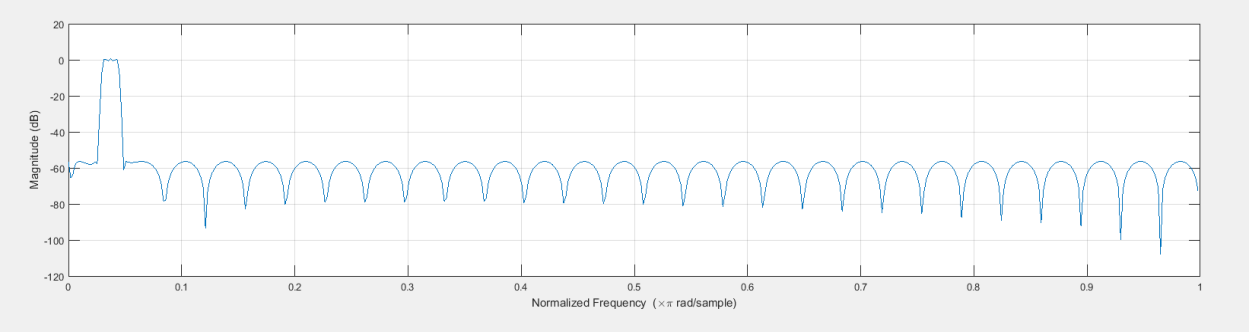


Fig. 20 Frequency response of b_E

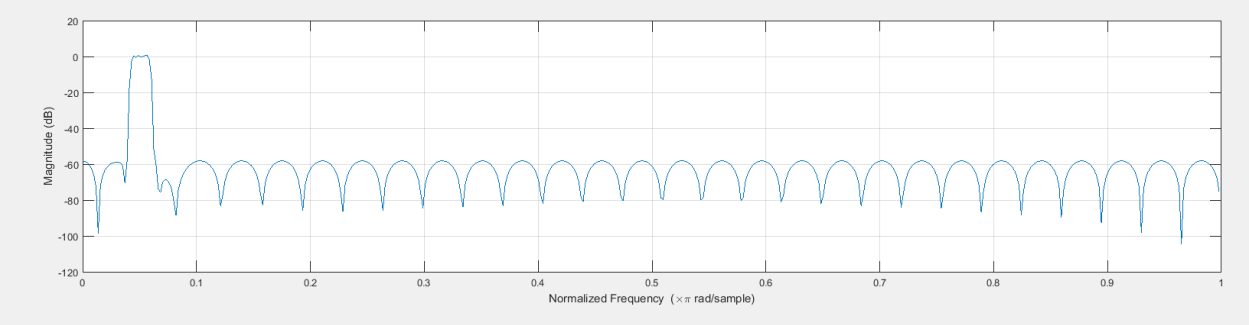


Fig. 21 Frequency response of b_A

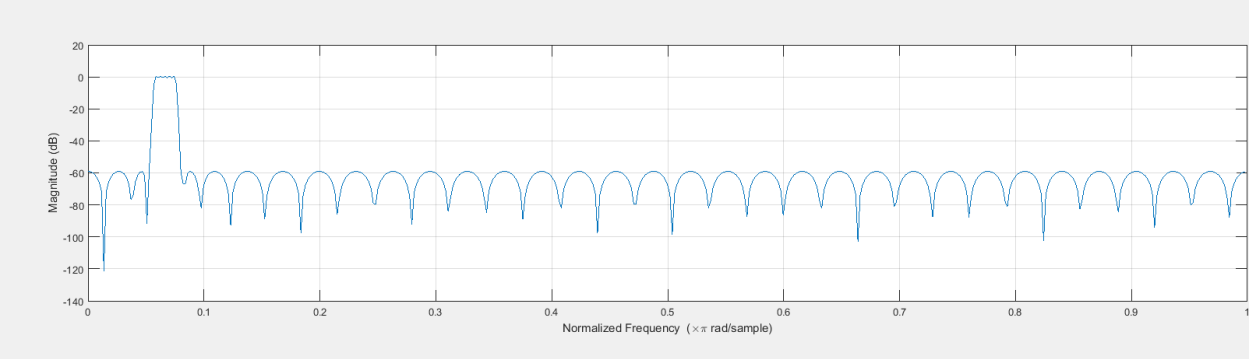


Fig. 22 Frequency response of b_D

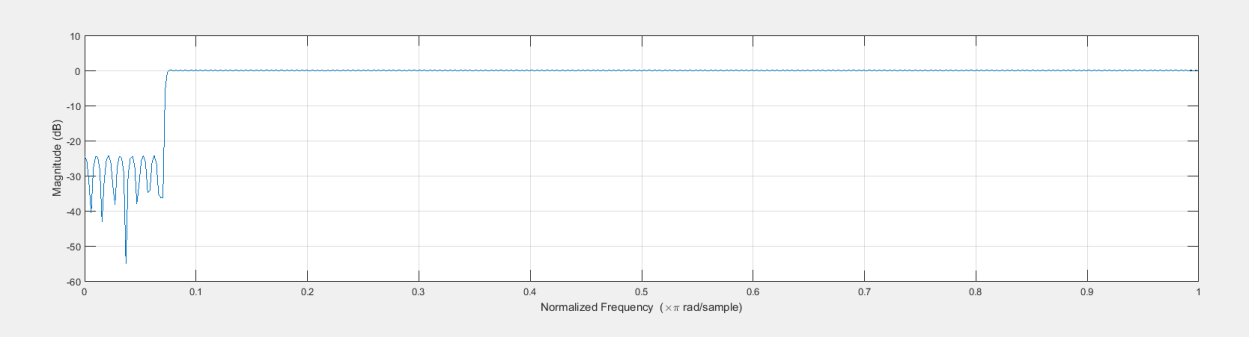


Fig. 23 Frequency response of highpass

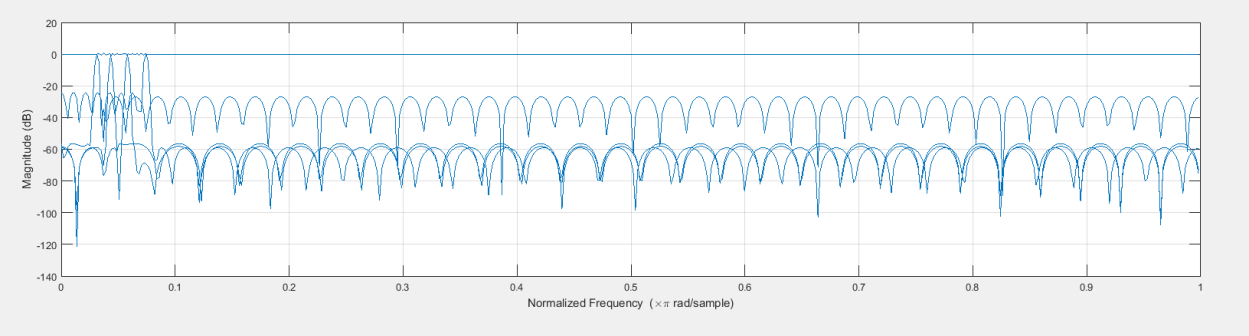


Fig. 24 All filter Frequency responses plotted on one graph

Tuner_FIRs generates the filters: b_E,b_A,b_D,lowpass,highpass. These are generated by feeding specifications of the filters to firpmord which then outputs the correct commands to be fed to firpm. Firpm is then used to create our final filters. As shown in

Fig. 21 plotting all the Filters onto one graph shows how every signal will fall into the passband of one filter. All the filters have less than 1000 coefficients.

The filters we used are FIR, finite impulse response, filters. We used FIR filters because they are simple to design, and always stable. Filters are very important in signal processing. They allow signals of certain frequencies to pass though, while attenuating signals of undesired bands. For example, in a real world guitar tuner, there may be noise in the final signal. This noise will show up on the DFT and could inhibit the tuner's ability to tune the guitar. Filters help to prevent this by attenuating the bands, and therefore noise, in which we are not interested

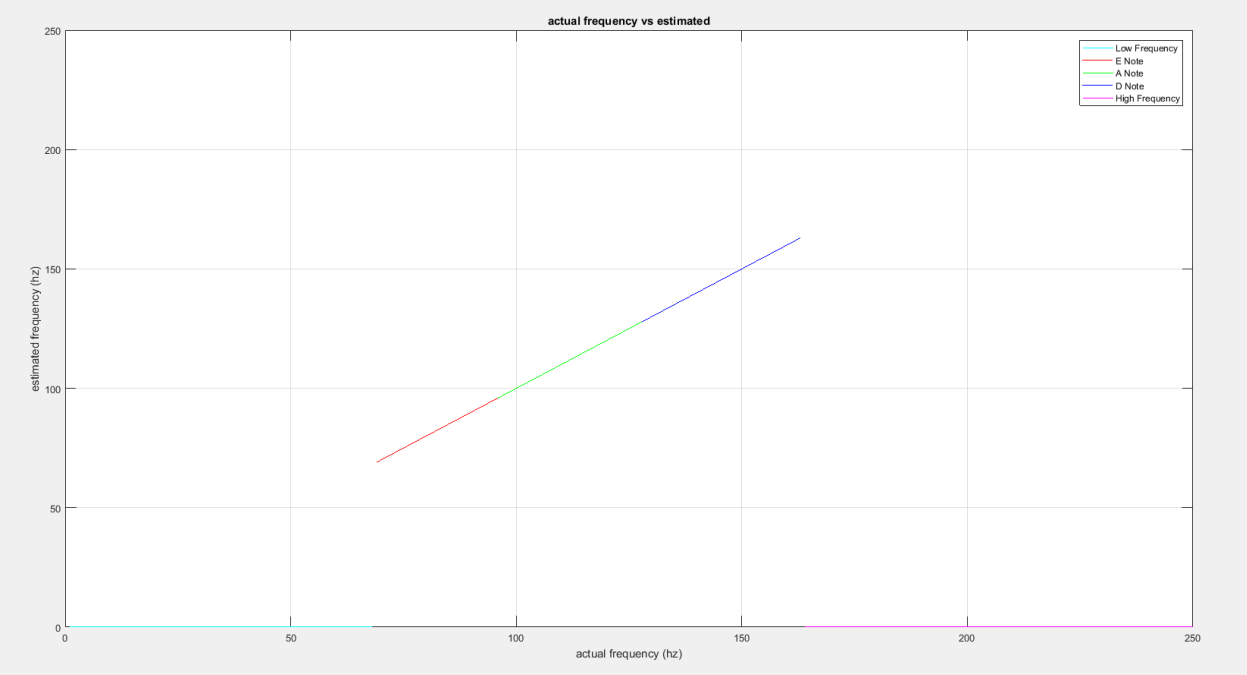


Fig. 25 Shows the results of our test script

To test our code, we used testscript.m. This creates our filters and then calls the DFT_Tuner for all hertz values between 1 and 250. The results of this, both fo_est and band, are stored and used to generate a plot. The plot shows the estimated hertz values versus the actual hertz value, and color codes the line based on which band was returned.

We also tested our code at several different frequencies to ensure it was accurate to 5 cents. As you can see in the chart below, we are well within the limit.

Actual Frequency	Estimated Frequency	Band	Cents

115.375	115.3924	A	0.261
77.125	77.1254	E	0.009
159.856	159.8683	D	0.133
146.3456	146.3524	D	0.080
89.786	89.7966	E	0.204
121.212	121.2211	A	0.130
55.642	NaN	Low Freq	NaN
201.234	NaN	High Freq	NaN

Conclusion

The results we obtained matched up with our expectations for what results we should achieve.

Part 1 sampled a sinusoid and created a DFT which matched our expectations. The use of hamming here increased the accuracy of our DFT, and would be important later to reduce the effects of noise. The DFT created was based off of the inputs of frequency, samples, and amount of zero padding.

Part 2 operated in nearly the same way as Part 1, except now we were operating on 2 separate signals, for one of which we were varying the amplitude. This section gave us an understanding of how hamming can make signals easier to tell apart on a plot. By plotting the 2 signals on the same plot, the effects of having similar frequencies and amplitudes on the plot covering each other became obvious.

Part 3 was our first work with real guitar signals. We took these signals and found the DFTs, which gave us an idea of how later simulated signals should appear. These real signals are noisy, unlike simulated signals. The noise is somewhat reduced by hamming, so these won't match up perfectly with simulated signals generated later on.

Part 4 was the creation of synthesized signals which would be based off of an input frequency. These signals were created through the generation of randomized phases and a preset list of amplitudes. This created the main sinusoid and its harmonics for the relevant frequencies. After the 10th sinusoid, they became irrelevant because further sinusoids would have very small amplitudes. The DFT of this generated signal could be compared to Part 3 to show similarities, the differences caused by noise, confirm that this part is working correctly.

Part 5 took an input sinusoid and some filters, which were generated ahead of time, and estimated the frequency of the input sinusoid and the band that this frequency lies within. It found the band by first filtering the signal with each filter, and then taking the sum of the squares for each band. The largest sum of squares corresponded to the correct band. Only the DFT values from this band were checked to find the largest value, and this value corresponded to the frequency, "fo_est", the estimated frequency.

For testing purposes we created testingscript.m in order to automatically call the DFT_Tuner, Tuner_FIRs and DFT_Synth_Guitar to see what values were output for band and frequency estimate for 1:250 Hz. These values were then plotted against the actual frequency. Frequency estimates of 0 were used for values outside of the supported frequency band. The line was color coded for what band the function detected.