Hector Perez - CSC 475

Assignment 1 Re-Submission

Question 1.

Fixed:

Implementation of DFT:

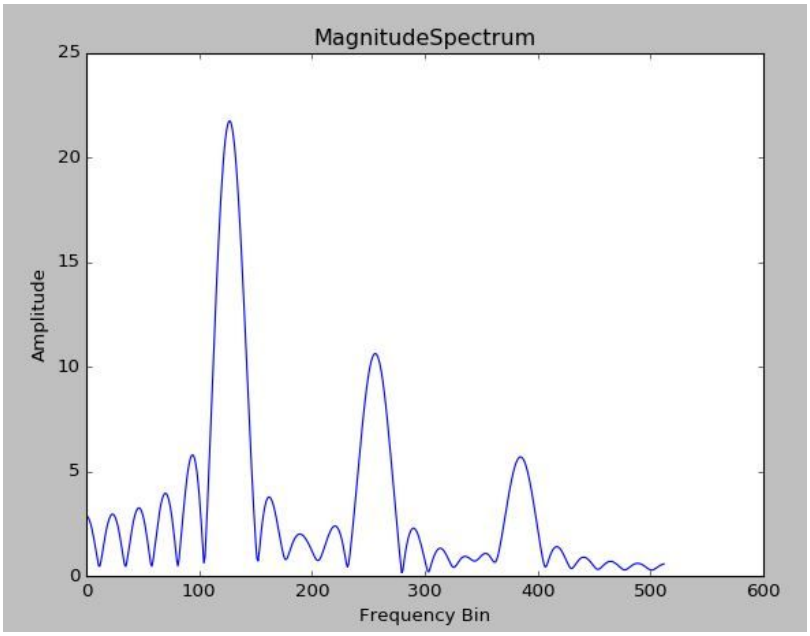
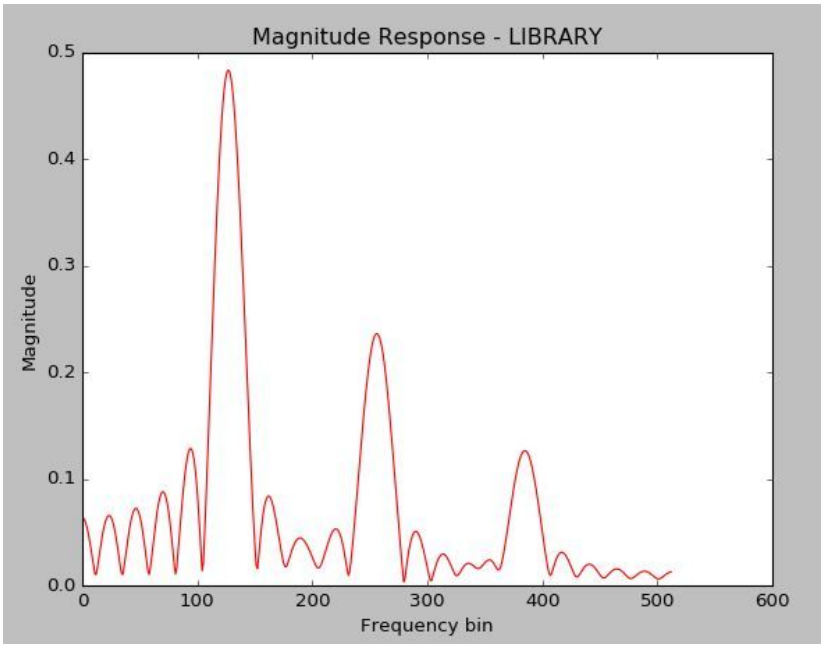
- Checked the code where the cosine is created and made sure that it is not a sine.
- Reduced number of bins so that it is easily visible when a frequency is between bins.
- Phase looks better.
- Showed the calculation of the size of a bin in Hz so that it depends on Fs and N points.
- Researched about implementation of DFT and made sure that correlation is used, not pointwise multiplication.
- For the last part of the question, made sure that the mixture of the harmonics is what is used to multiply pointwise with the basis functions.

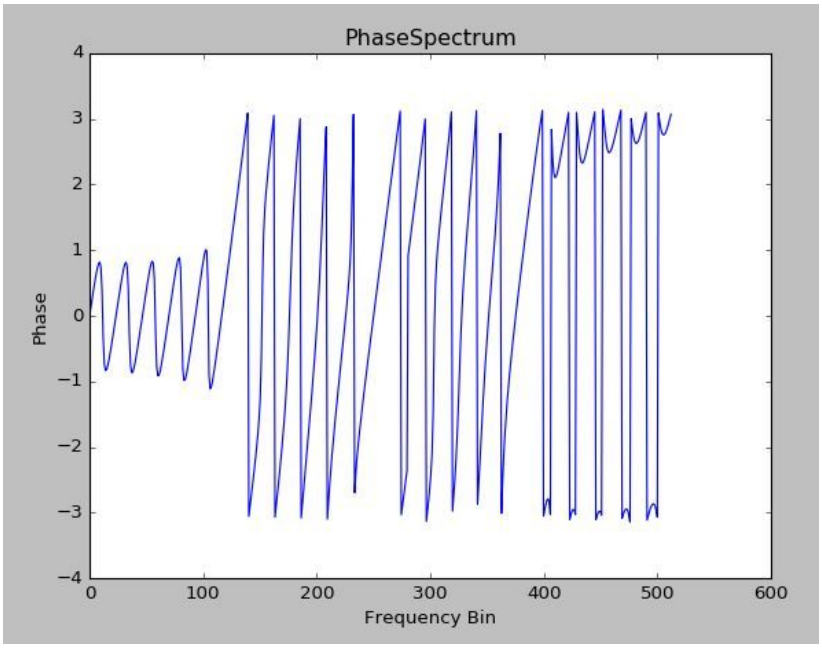
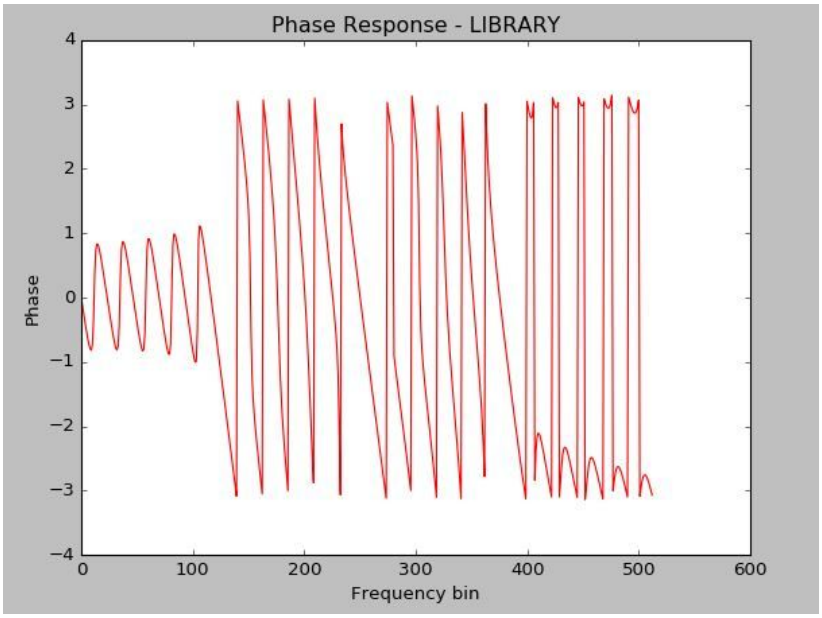
1A. DFT implementation in Python:

I include here only the main dft implementation. The rest of the code is in Appendix A.

```
def dftByHector(sizeOfFFT,sign,sizeOfBin):
        re_in = sign.data;
        n = len(re_in);
        re_out = np.zeros(sizeOfFFT,);
        im_out = np.zeros(sizeOfFFT,);
        for k in range(0,sizeOfFFT):
                sumReal = 0;
                sumImag = 0;
                for t in range(0,n):
                        #to be more intuitive, we could have done
                        #angle = float(2.0*np.pi*t*(k*sizeOfBin)/(sizeOfFFT*sizeOfBin)
                        #but cancellation occurs.
                        angle = float(2.0*np.pi*(t*k)/sizeOfFFT);
                        sumReal += re_in[t] * np.cos(angle);
                        sumImag += re_in[t] * np.sin(angle);
                re_out[k] = sumReal;
                im_out[k] = sumImag;
        non_Aliasing_Points = sizeOfFFT/2;
        plotDFT(non_Aliasing_Points, sign,re_out,im_out);
        return 0;
```

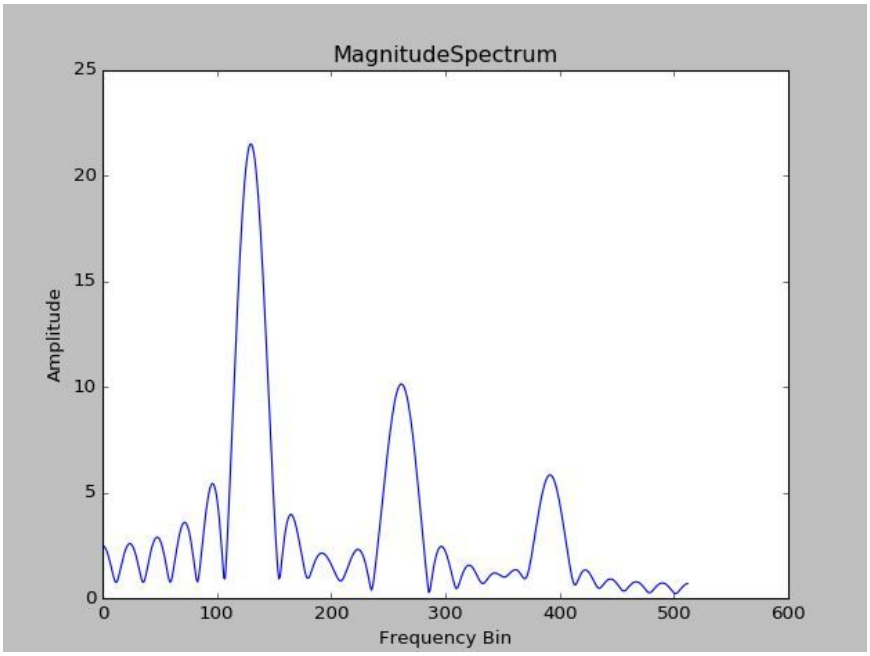
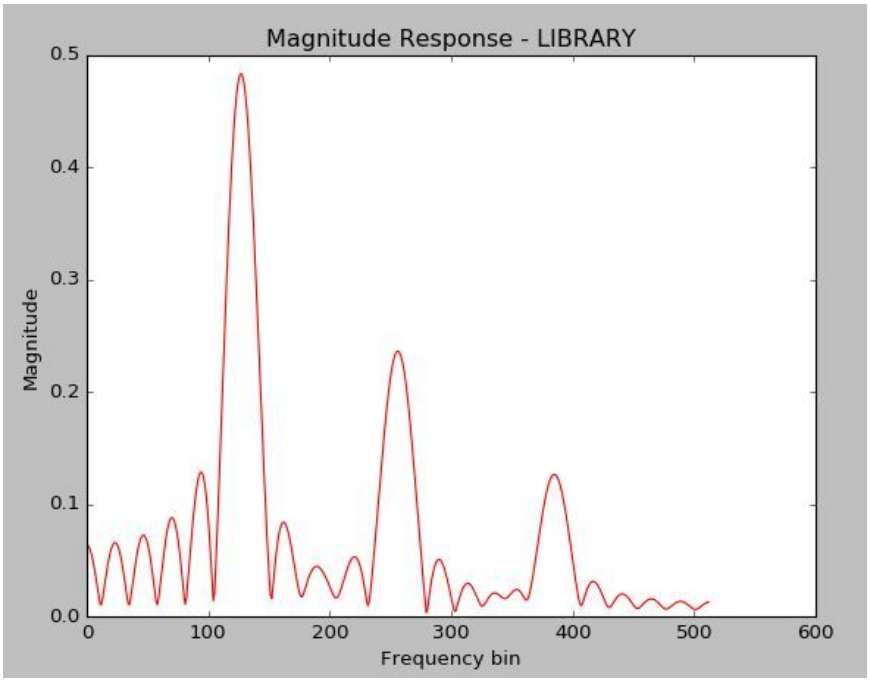
1B. Test FFT implementation with 3 harmonically related sinusoids (PLOT).
B.1 Exact fundamental. Used bin 128 = 5512.5 Hz. Magnitude and phase plots compared.
Fs= 44100 and Number of Points in DFT = 1024. Size of Freq Bin is Fs/NumPoints = 43.07 Hz.

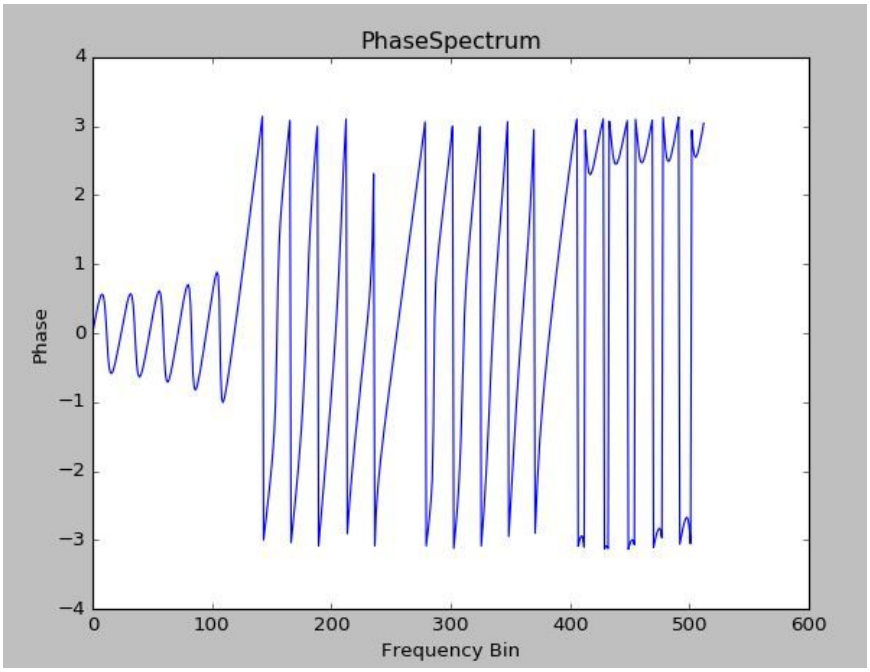
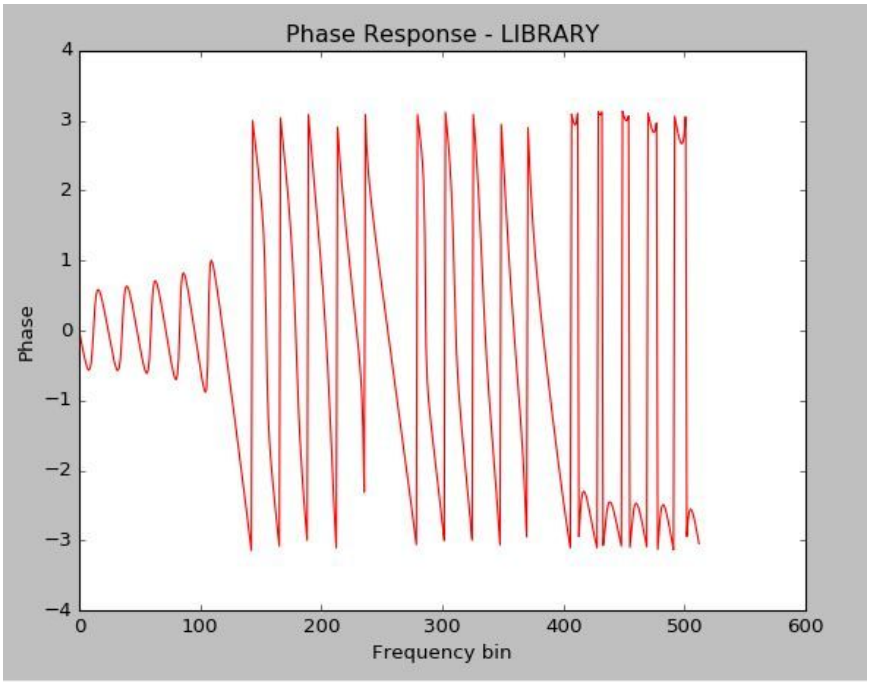
	Magnitude spectrum plotted from implementation above.
	Magnitude spectrum plotted from library FFT function.

	Phase response for DFT implementation
	Phase response for FFT Library implementation.

1.B.2. Fundamental between bins.

We pick a fundamental that equals the frequency of bin 130.5. This is ~ 5,620 Hz.

	Magn Spectr. for DFT implementation
	Magn Spectr. for FFT library function

	Phase Resp for DFT impl.
	Phase Resp. for FFT function Library.

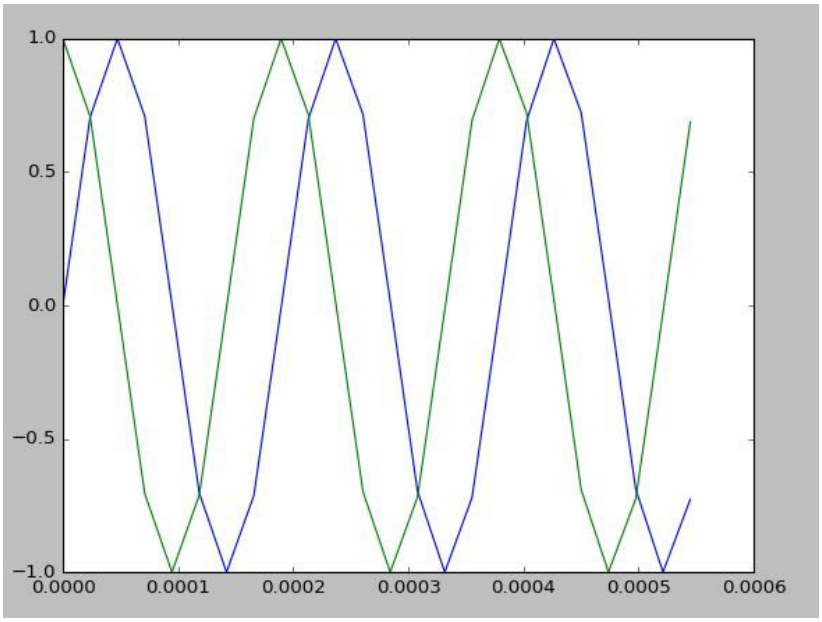
Discussion: The FFT for the signal with an exact bin frequence should be at the bin. Then, the one that is between bins should show frequency leaking (because it's not exactly at a bin, the energy for it will be 'split' by the closest bins.)

C. Plot basis functions for a particular DFT bin - The number of points for my DFT implementation is 256 and Fs is 44100 Hz. The basis functions for bin 30 are plotted here. Fix note: I am checking that the sine and cosine are created correctly.

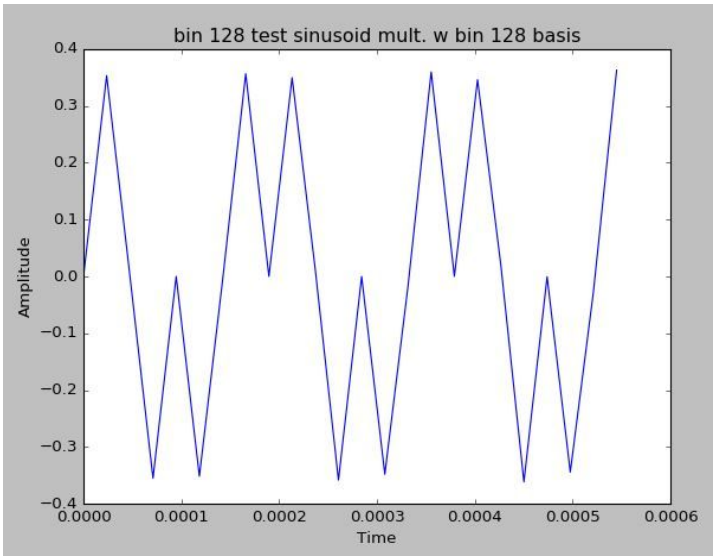
Code: **Appendix B:** *BasisFunctions.py*

D. Pointwise multiplying.

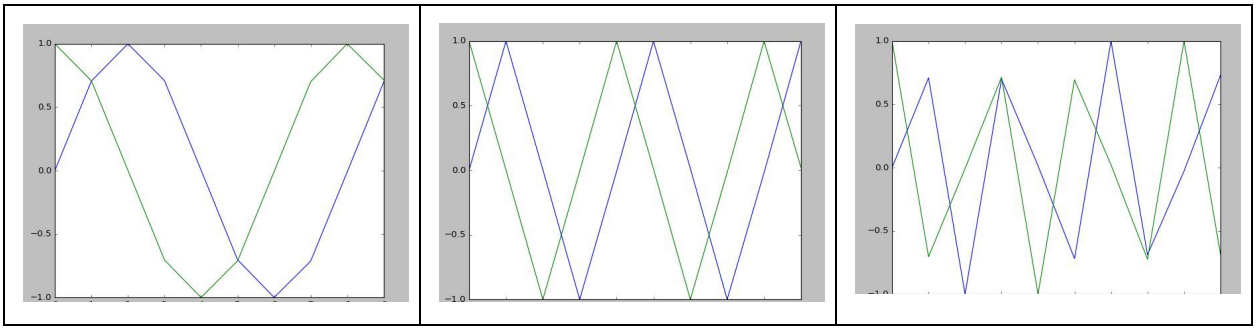
D.1. Input fundamental multiplied with basis functions corresponding to closest bin. The corresponding bins to the first fundamental is bin 128. Its basis functions are:

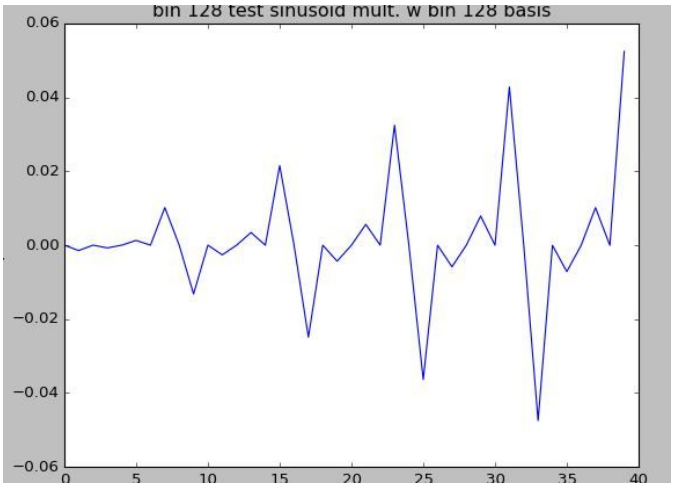


Result of pointwise multiplication with the basis functions.



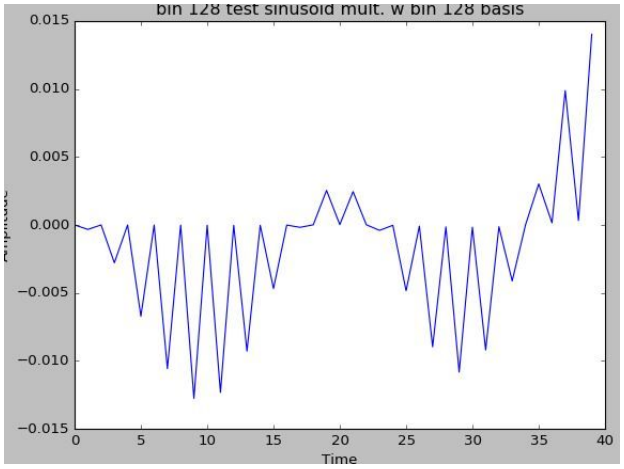
D.1. REPEATED: MULTIPLYING WITH THE ‘THREE HARMONICS SIGNAL’ and the two basis functions for each of those signals. Code is in the Appendix B.





The fundamentals were tested with a mixture of sinusoids that correspond to the basis functions shown here.

D.2 Input mixture signal multiplied with all basis functions corresponding to unrelated bins. The fundamental for the mixture will be Bin 15 and Bin 128 will be used for unrelated basis functions.



D.3 Discussion.
In class it was discussed that when multiplying a sinusoid by an unrelated bin basis, the correlation should be close to zero. This means that the sum of pointwise multiplying (above) all should be close to zero for an unrelated bin. If the result is nonzero, this means there was correlation between the input and the basis functions. If a test sinusoid was out of phase with the input, then we might get lower results. Using a mixture of a sine and a cosine helps solve the phase problem.

Question 2:

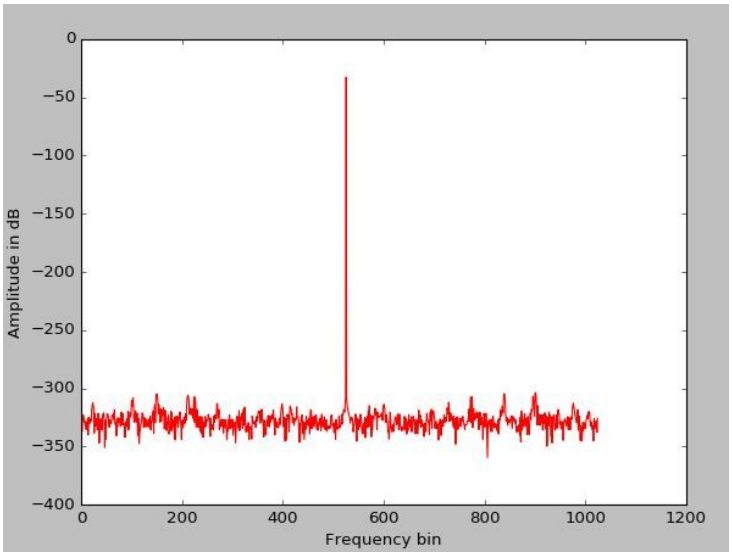
Fixed:

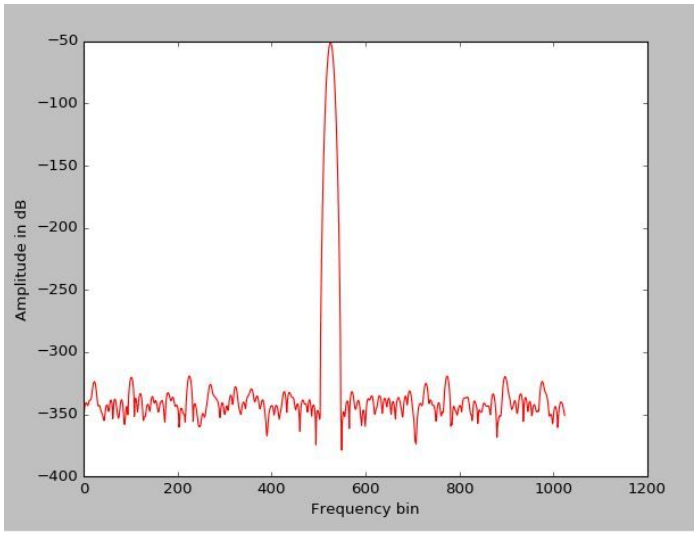
Improved the plots and was more precise on labeling the axes.

2.A: Plot magnitude response in dB of a sine wave that has frequency equal to bin 525 using an existing FFT implementation. Use number of points as 2048 for the FFT. I didn't overlap the plots, but i put them one right after the other.

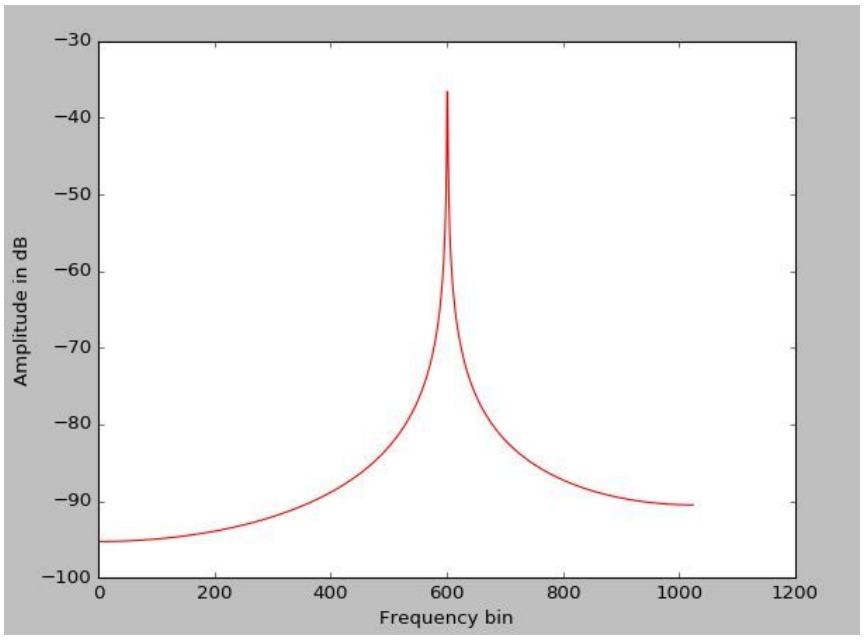
Size of bin in Hz = Fs/Number of points = 44100/2048 = 21.53 Hz.
Frequency of bin 525 = 11304.93 Hz

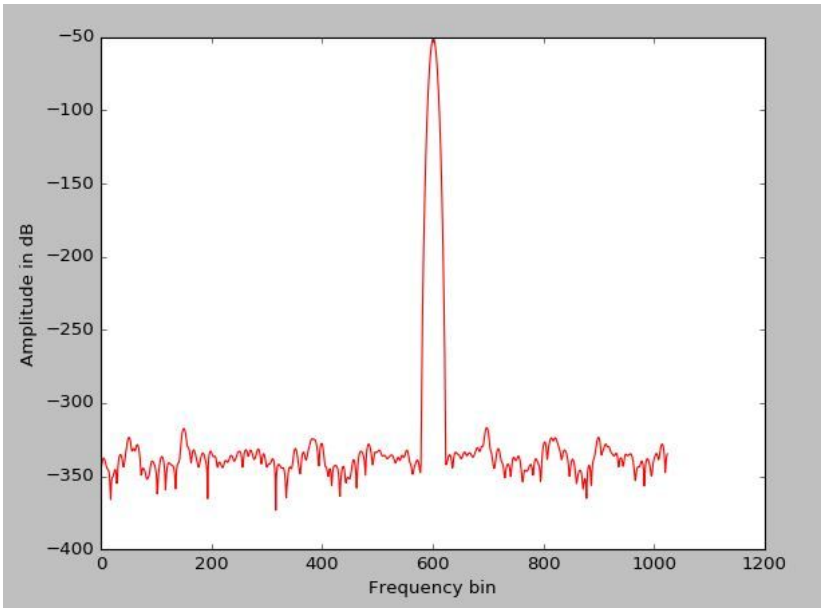
Code: **Appendix C: *question2.py*** and ***question2functions.py***

Windowed Bin 525	

Not Windowed Bin 525	

Magnitude response in dB of a sine wave with freq corr. to bin 600.5 = 12930.69Hz

Windowed Bin 600.5	

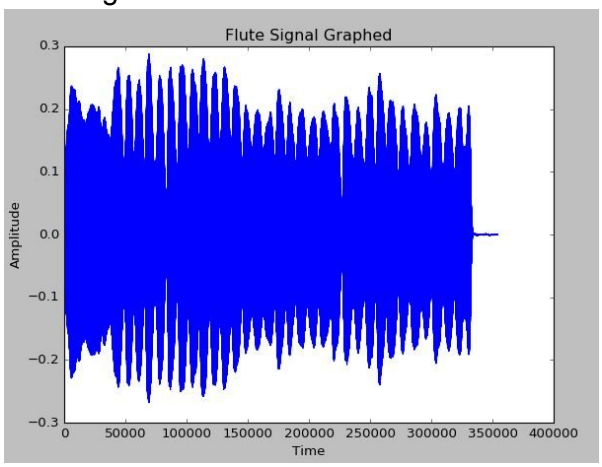
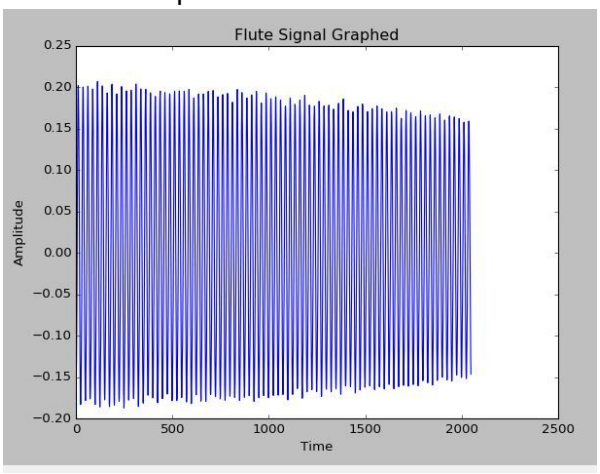
Not Windowed Bin 600.5	

Discussion Question 2 A: All the plots show that windowing the function before performing the FFT (multiply by hanning window in the time domain) will ‘clean up’ the magnitude response. The frequencies that are initially found when not windowing are ‘filtered out’. If a signal has discontinuities, then the FFT without windowing tries to ‘recreate’ it by adding a lot of unneeded frequency elements. But with windowing these discontinuities are reduced, and so frequency leaking doesn’t occur. All the non-windowed FFT here have frequency leaks. The dB plots really show how the spectrum is contaminated when the signal isn’t windowed. When it is windowed, other frequencies are just ‘lessened’.

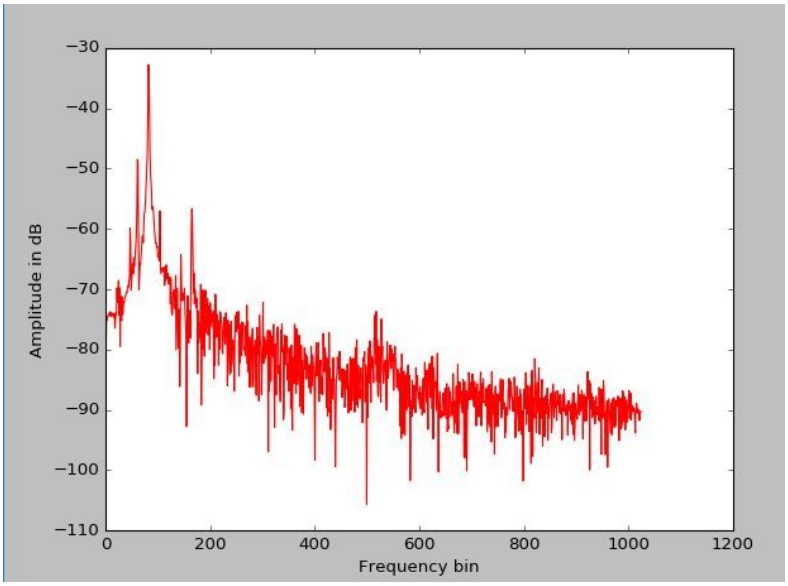
Question 2 : 2.B. Code: Appendix D : Question2_b.py

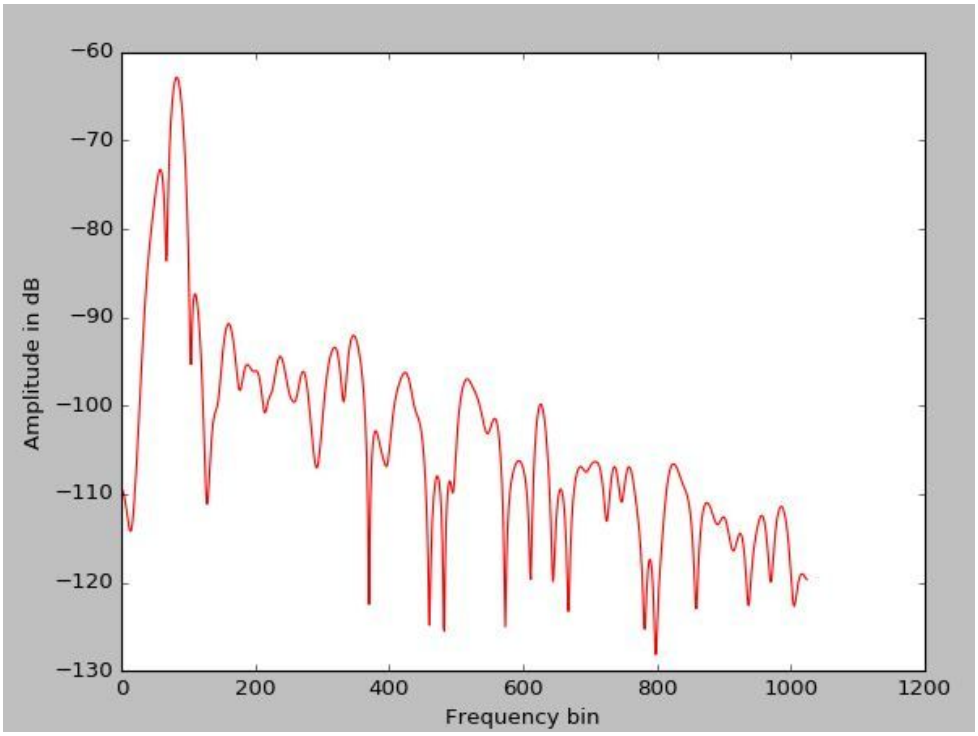
a. Find single note of an instrument - select 2048 samples of audio from steady state of sound. (Violin, flute,organ … etc). Window as before, and Perform FFT on it.

The waveform for this flute note is this. Posteriorly, 2048 samples were selected. The FFT showed a peak at 1764 Hz ~approx Bin 76.

Initial Signal	Selected samples
	

Plot of Magnitude Spectrum

Not windowed	

Windowed	

Discussion: The effect on the plots is that it is easier to see which frequency is predominant. Noise in the spectrum caused by discontinuities in the signal are reduced by windowing. ‘Lobes’ appear.

Question 3: Fixed: Removed some sections that weren’t answering the question.

A. Characterization of a music piece.

I decided to characterize ‘Walking on the moon’ by Sting and the police.

Orchestration: Guitar, drum kit, 1 voice.

Timbre: The instruments used in the song are modified. The drum kit which has a very short delay and the guitar and vocals have a lot of reverb applied to them. There are changes in dynamics mostly from the vocals and the drums (for emphasis). The guitar plays full chords and not individual notes.

Acoustics: There is no perceivable background noise, but the perceivable size of the room changes due to the reverberation added.

Rhythm: The tempo is 146 Bpm. The time signature is 4/4.

Melody: The vocals carry the melody. The register is mostly the middle range. For emphasis, sometimes the voice hits higher range.
.
Harmony: It seems that the first chord progressions are in D minor, and then for the choruses there is modulation to the relative major (F major). This is all played by the guitar.

Verse: repeats Dm7 and C
Chorus: repeats B (flat), F, C and G minor.

Structure: Large scope (what parts does the music have). Do they repeat?
If we call verses A, choruses B and bridge C, The structure of the song is AABABA. No bridge.

B. Information needs for different users.

Later, different types of users and their information needs are discussed. When applied to 'Walking on the Moon', these are some possible information needs.

1. Casual User needs: find similar sounding songs to it. Suggest other songs I might like. Cluster it with other similar songs.
2. Professional Users: Retrieve musical works that have characteristics similar to this one (Being able to choose if rhythm, melody, harmony,orchestration are relevant). Highlight which dimensions are relevant.
3. Music theorists. Musicologists. Music scholars, and musicians. Don't care that much about the music itself. They want for example to compare the performance differences between this and another performance. So maybe compare the original and covers of it.

C. What is the difference between query-by-humming and query-by-example
(1 paragraphs) ? (4/3 points)
In query by example, a section of the actual content of a piece of music is used for the query. In query by humming, a vocal interpretation/performance by the user is used for the query. The difference is how the user interacts with the system.

D. How is MusicXML different from MIDI (1 paragraph) ? (4/3 points)

Music XML - Leverages the properties of XML for information exchange. It tries to represent western musical notation. For example can differentiate between note durations explicitly(half note, whole note).

MIDI (Musical Instrument Digital Interface) - Represents each event in time but doesn't care about the durations of the events (can't relate them back to a duration such as a half note or whole note). Each event is an 'X' sized word (There is a set number of bits to represent it. MIDI is oriented toward capturing information for a particular digital 'keyboard performance'.It is also used for data exchange.

E. What part of the article was the most novel/interesting from your personal perspective ? Explain it to someone who has not read the article in your own words. (4/3 points)

I am interested in detecting the key/tonality for the group project, so I paid attention to the section of the article which explained how chord sequences could be extracted from the content of an audio signal. The article said this could be extended to tonality extraction.

A musical chromatic scale is used (made up of 12 equidistant notes within an octave). The first step is to recognize which frequencies are present for one of the tones of the chromatic scale. It's interesting that each note in the scale will NOT map to a single specific frequency,but to a group of frequencies.This is because instruments have a certain quality/timbre (the sound of a single instrument note is made up of not only one frequency).

The second step is to create chord templates. A chord template uses the data we had per note in the chromatic scale, and tries to 'profile' what it would look like if those notes' frequencies were combined to create a chord.

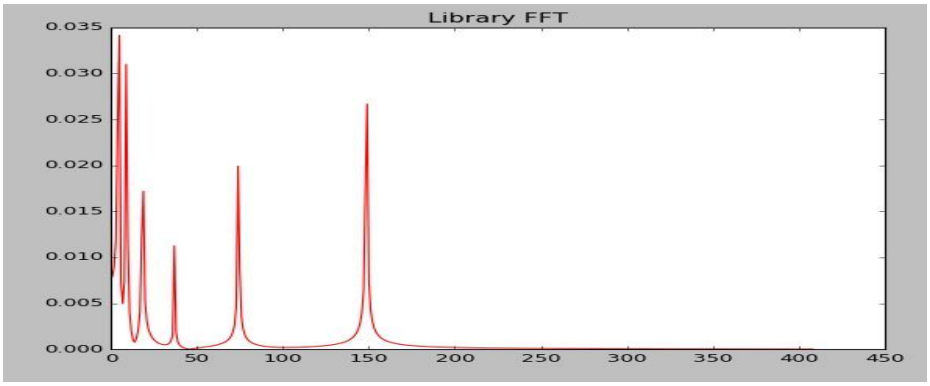
Lastly, to recognize the chords, the incoming signal is compared with each of these chord templates. I am not so clear about how many profiles to make, because there are many different kinds of chords (thinking about 7th chords, which are almost like the regular chords, but with one extra note).

Question #4.

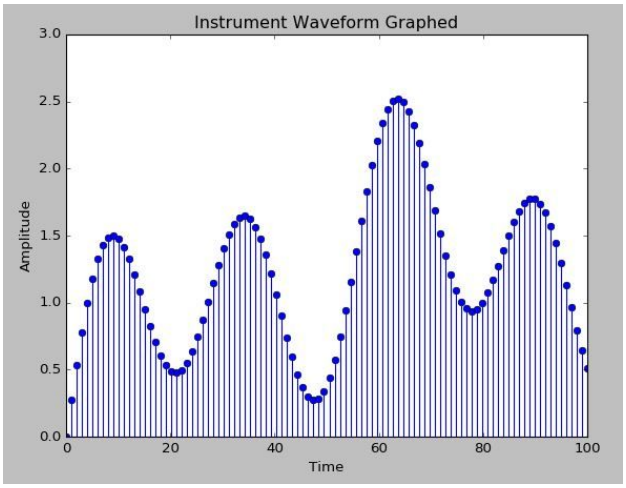
Fixed:

- Added discussion about how the peaks were chosen from the Spectrogram in Audacity.
- Used the synthesized sound for the last section of the question.

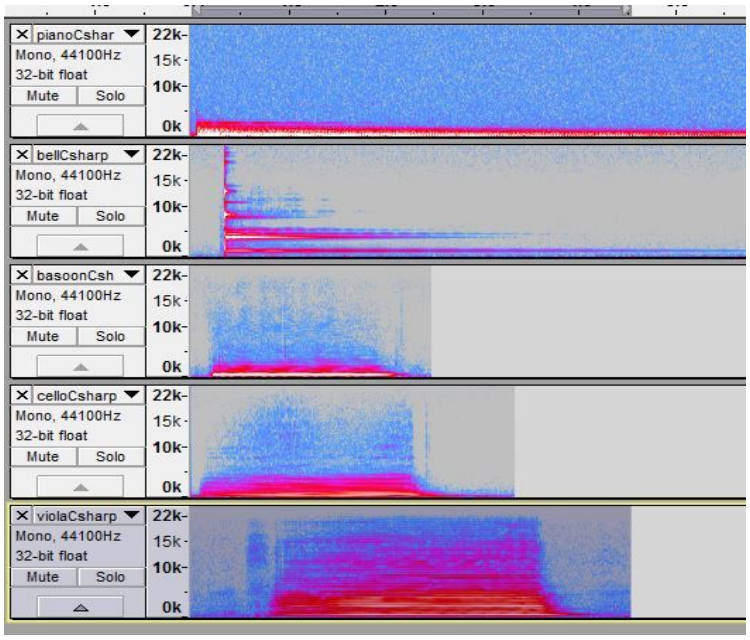
Code: **Appendix E** - ***AdditiveSynthesis.py.***
I created a temp array with these frequencies and amplitudes (num1 is freq, num2 is amp): [[50,1],[100,0.75],[200,0.5],[400,0.25],[800,0.5],[1600,0.75]]; Here is a plot of the DFT of the generated instrument.



The peaks were chosen as instructed, examining their spectrum in Audacity and selecting the highest peaks. In Audacity the spectrogram is color coded, and if red or white is the strongest.



B. Either find online or record yourself using Audacity two different sounds of the same pitch (for example violin and a flute). (5 points)

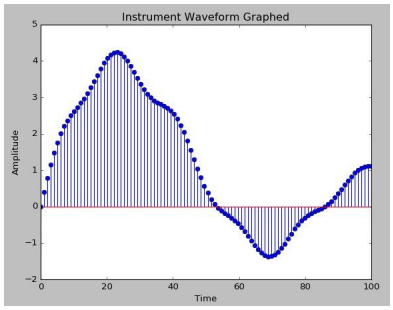


I found five sounds of the same pitch. Piano, bassoon, viola, cello,and bells. The bells spectrogram shows widely spaced harmonics.

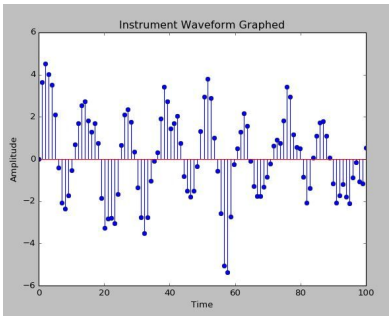
What is missing in the synthesized sounds is attack delay sustain and release.

I plotted the waveforms for the different instruments.

Piano

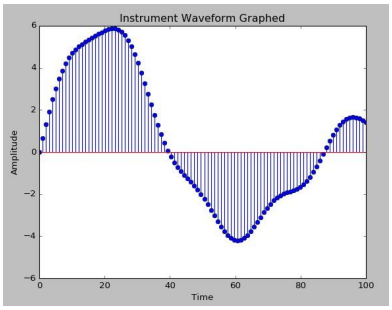


Bell

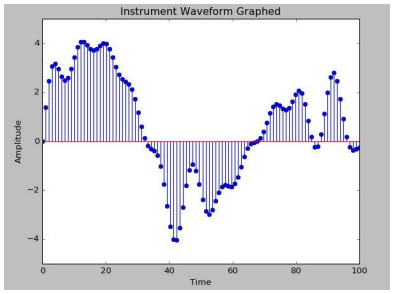


Basoon

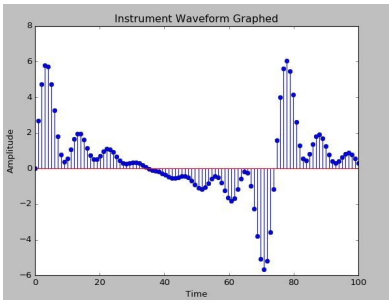
Looks very similar to Piano.



Cello



Viola



Peaks chosen (in Hz), per instrument: (Amplitudes were chosen for each peak as well).

Piano	Bell	Basoon	Cello	Viola
400	1300	500	750	700
500	19000	600	650	650
600	13000	400	700	4800
1000	10700	700	5000	4200
300	8400	550	4000	3600

1200	5300	450	3000	3000
2300	4500	800	2000	2400
2400	3800	1000	1000	1800
2500	3700	2000	500	1200

C. Read about the equal temperament tuning system and MIDI. Write code that takes a list of MIDI note numbers (not note names) and synthesizes the corresponding melody using your additive synthesis instrument. Create an audio file with a well known melody using your system and plot the corresponding spectrogram in Audacity (10 points).

Code **Appendix F: *Melody.py*** Recreated the Mario Bros. theme.
Explanation: the previously found peaks were used, and the melody is created by picking the midi note given as the fundamental, and then calculating the corresponding harmonics that would be present in the sound by a ratio from harmonic to fundamental.

Midi uses 69 as concert A = 440Hz. And concert C = number 60 in Midi.

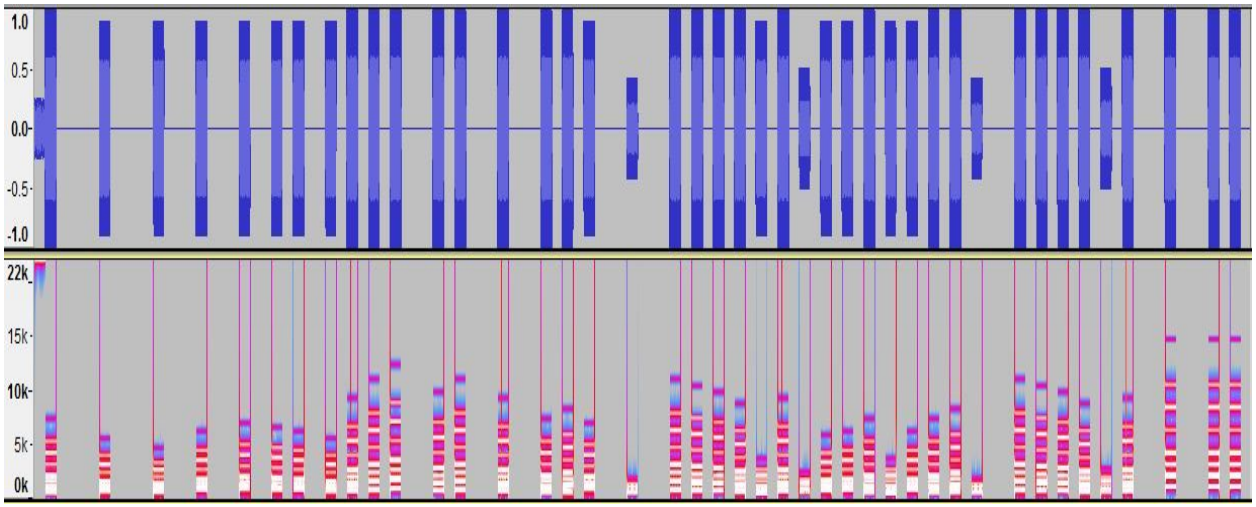
A formula to calculate other frequencies for ‘d’ note value in MIDI is:

$$f = 2^{(d-69)/12} \cdot 440\ \text{Hz}$$

Although this tells us how to calculate a single frequency, it doesn’t tell us how, if we have this ‘mixture’ of sinusoids, to deal with the other frequencies that make up the sound.

Discussion: The results were good. The melody is recognizable.

The waveform and spectrogram look like this:



For different midi notes, the harmonics for a single note were based on the ratios that existed on the single note on which the additive synthesis was based. For example, if in the synthesized signal the frequency for a harmonic is A and the frequency for the fundamental is F, then a ratio can be calculated, as A/F, and then multiplied by the new fundamental for a different midi note. In the figure you can see how the different harmonics included in the sound are visible: different 'white' lines that each represents a harmonic calculated in the way explained.

Question #5.
Fixed: Clearly show that the input is being buffered in MATLAB.
Q5A.Forwards and Backwards FFT with no loss of audio.

Buffering occurs in the code in q5.py in the appendix G - each iteration, up to the number of frames that we have, the current samples selected are:

```
curr_input_samples = inputSamples(i*windowSize:(i+1)*windowSize);
```

With a windowSize of 2048 samples, as instructed.

Code: **Appendix G: *q5.py*** (Question 5 first part)

Q5B. Replacing the phase with random numbers by converting imaginary FFT data to polar form and back to rectangular form.convert back to real and imaginary coefficients and perform the inverse FFT. Try different window sizes.

Code: **Appendix H: *q5randomphases.py***

Discussion:

An input signal can be decomposed into a linear combination of sinusoids with certain phase and amplitude. When the phase is randomized, the sound gets 'blurred' and is not as clear. This is because to decompose the signal into the sinusoids, a matching phase sinusoid must be 'present' in the signal. If before reconstructing the signal with IFFT we changed that phase, then the linear combination is changed and the signal composed from the sinusoids shifted in phase is not clear anymore - it's not the same signal.
The sound heard here is not 'continuous' - it 'comes in spurts', and there is an audible 'tone'. I think this happens because the change in phases from window to window is not smooth, as it would normally be. When experimenting and changing the size of the window, the distance between the 'discontinuities' changed accordingly. The original fft data wouldn't have phases that change so drastically or randomly. Therefore it is noticeable that there are artifacts.

C. Select the 4 highest peaks in the magnitude spectrum from polar form and set all the other magnitudes to zero while retaining the phases. Perform the inverse FFT and listen/comment on effect on the audio.

Code: **Appendix I: *q5mostlyphases.py***

Discussion: The sound heard is as if the pitch is not stable - it keeps changing. The sound is clear, but for example I had a violin single note, yet the sound that comes out is varying in pitch.

One possibility is that the phases corresponding to no magnitude might be affecting neighboring frequencies' phases. Another consideration is that the values for the real and imaginary parts (cartesian complex number) would have to be such that the magnitude would be zero and the phase would be non-zero. For a+jb, there aren't such values.

Appendix A : Code for DFT Implementation. `I skip the imports.`

```
class mainProg1():
	def main():
		#-----IMPLEMENTED----#
		Fs = 44100;
		sizeOfFFT = 1024; #size of fft ( k = N , in this case)
		sizeOfBin = Fs/sizeOfFFT;
		fundamental1 = 128*sizeOfBin;
		fundamental2 = 130.5*sizeOfBin;
		#PART A - fundamental at freq bin
		sine1 = mir.Sinusoid(amp = 1,freq = fundamental1,phase = 0, duration = 0.001);
sine2 = mir.Sinusoid(amp = 0.5,freq = 2*fundamental1, phase = np.pi/4, duration = 0.001);
sine3 = mir.Sinusoid(amp = 0.25,freq = 3*fundamental1, phase = np.pi/8, duration = 0.001);
		signal1 = mir.Mixture(sine1,sine2,sine3);
		methodsForFFT.dftByHector(sizeOfFFT,signal1,sizeOfBin);
		#PART A - fundamental at freq bin
		#PART B - fundamental between freq bin
		sine4 = mir.Sinusoid(amp = 1,freq = fundamental2,phase = 0, duration = 0.001);
sine5 = mir.Sinusoid(amp = 0.5,freq = fundamental2*2, phase = np.pi/4, duration = 0.001);
sine6 = mir.Sinusoid(amp = 0.25,freq = fundamental2*3, phase = np.pi/8, duration = 0.001);
		signal2 = mir.Mixture(sine4,sine5,sine6);
		methodsForFFT.dftByHector(sizeOfFFT,signal2,sizeOfBin);
		#PART B - fundamental between freq bin
		#-----LIBRARY----#
		methodsForFFT.plotWithLibraryFFT(sizeOfFFT, signal1);
		methodsForFFT.plotWithLibraryFFT(sizeOfFFT, signal2);
		return 0;

	if __name__ == "__main__": main()

def dftByHector(sizeOfFFT,sign,sizeOfBin):
	re_in = sign.data;
	n = len(re_in);
	re_out = np.zeros(sizeOfFFT,);
	im_out = np.zeros(sizeOfFFT,);
	for k in range(0,sizeOfFFT):
		sumReal = 0;
		sumImag = 0;
		for t in range(0,n):
			angle = float(2.0*np.pi*(t*k)/sizeOfFFT);
			sumReal += re_in[t] * np.cos(angle);
			sumImag += re_in[t] * np.sin(angle);
		re_out[k] = sumReal;
```

```
                im_out[k] = sumImag;
        non_Aliasing_Points = sizeOfFFT/2;
        plotDFT(non_Aliasing_Points, sign,re_out,im_out);
        return 0;


def plotMagnitudeSpectrum(non_Aliasing_Points, sign,re_out,im_out):
        magnitude_spectrum = np.zeros(non_Aliasing_Points,);
        print len(magnitude_spectrum);
        for i in range(0,non_Aliasing_Points):
            magnitude_spectrum[i] = np.sqrt(im_out[i]**2+re_out[i]**2);
        arr2 = magnitude_spectrum;
        arr1 = np.linspace(0,non_Aliasing_Points,non_Aliasing_Points);
        plt.ylabel('Amplitude');
        plt.xlabel('Frequency Bin')
        plt.title('MagnitudeSpectrum');
        plt.plot(arr1,arr2);
        return 0;
def plotPhaseSpectrum(non_Aliasing_Points, sign,re_out,im_out):
        phase_spectrum = np.zeros(non_Aliasing_Points,);
        for i in range (0,non_Aliasing_Points):
                phase_spectrum[i] = np.arctan2(im_out[i],re_out[i]);
        arr4 = phase_spectrum;
        arr3 = np.linspace(0,non_Aliasing_Points,non_Aliasing_Points);
        plt.title('PhaseSpectrum');
        plt.ylabel('Phase');
        plt.xlabel('Frequency Bin');
        plt.plot(arr3,arr4);
        return 0;
def plotDFT(non_Aliasing_Points,sign,real_part, imaginary_part):
        plotMagnitudeSpectrum(non_Aliasing_Points,sign,real_part, imaginary_part);
        plt.show();
        plotPhaseSpectrum(non_Aliasing_Points,sign,real_part,imaginary_part);
        plt.show();
        return 0;
def plotWithLibraryFFT(sizeOfFFT, sign):
        non_Aliasing_Points = sizeOfFFT/2;
        freq = np.fft.fft(sign.data,sizeOfFFT)/len(sign.data);
        real_part = np.real(freq);
        imag_part = np.imag(freq);
        magn_spectrum = np.absolute(freq);
        phase_spectrum = np.zeros(non_Aliasing_Points,);
        for i in range (0,non_Aliasing_Points):
                phase_spectrum[i] = np.arctan2(imag_part[i],real_part[i]);
        lin_space = np.linspace(0,non_Aliasing_Points,non_Aliasing_Points);
        plt.title('Magnitude Response - LIBRARY');
        plt.ylabel('Magnitude');
        plt.xlabel('Frequency bin');
        plt.plot(lin_space, magn_spectrum[:non_Aliasing_Points],'r');
        plt.show();
```

```
plt.title('Phase Response - LIBRARY');
plt.ylabel('Phase');
plt.xlabel('Frequency bin');
plt.plot(lin_space, phase_spectrum[:non_Aliasing_Points],'r');
plt.show();
```

Appendix B: *BasisFunctions.py* - I skip the imports.

```
class BasisFunctions():

	def main():
		Fs = 44100;
		TotalNumBin = 1024;
		FundamentalBin = 128;
		SizeOfBin = Fs/TotalNumBin;
		FundamentalFreq = FundamentalBin*SizeOfBin;
		BasisFreq = FundamentalFreq;
		sine1 = mir.Sinusoid(freq = FundamentalFreq);
		cosine1 = mir.Sinusoid(freq = FundamentalFreq, phase = np.pi/2);
		sine2 = mir.Sinusoid(freq = 2*FundamentalFreq);
		cosine2 = mir.Sinusoid(freq = 2*FundamentalFreq, phase = np.pi/2);
		sine3 = mir.Sinusoid(freq = 3*FundamentalFreq);
		cosine3 = mir.Sinusoid(freq = 3*FundamentalFreq, phase = np.pi/2);
		mixture = mir.Mixture(sine1,sine2,sine3);
		plt.plot(sine1.data[:10]);
		plt.plot(cosine1.data[:10]);
		plt.show();
		plt.plot(sine2.data[:10]);
		plt.plot(cosine2.data[:10]);
		plt.show();
		plt.plot(sine3.data[:10]);
		plt.plot(cosine3.data[:10]);
		plt.show();
		#multiply everything.
		arr4 = np.multiply(mixture.data, sine1.data);
		arr5 = np.multiply(arr4, cosine1.data);
		arr6 = np.multiply(arr5, sine2.data);
		arr7 = np.multiply(arr6, cosine2.data);
		arr8 = np.multiply(arr7,sine3.data);
		arr9 = np.multiply(arr8, cosine3.data);
		plt.title('bin 128 test sinusoid mult. w bin 128 basis');
		plt.ylabel('Amplitude');
		plt.xlabel('Time');
		plt.plot(arr9[:40]);
		plt.show();
		return 0;
	if __name__ == "__main__": main()
```

Appendix C: ***question2.py*** and ***question2functions.py*** I skip the imports.

```
class question2():
	def main():
		sign = mir.Signal();
		relative_path = 'wav_files/flutehighMono.wav'
		sign.wav_read(relative_path);

		sizeOfFFT = 2048;
		sizeOfBin = 44100/float(sizeOfFFT);

		sine_a = mir.Sinusoid(freq = 525.0* sizeOfBin);
		sine_b = mir.Sinusoid(freq = 600.5* sizeOfBin);
		sine_c = mir.Sinusoid(freq = 525.5* sizeOfBin);

		hanning_window = np.hanning(2048.0);

question2functions.plotWindowedAndNonWindowed(sign,sizeOfFFT,hanning_window);
question2functions.plotWindowedAndNonWindowed(sine_a,sizeOfFFT,hanning_window);#525
question2functions.plotWindowedAndNonWindowed(sine_b,sizeOfFFT,hanning_window);#600.5
question2functions.plotWindowedAndNonWindowed(sine_c,sizeOfFFT,hanning_window);#525.5

		return 0;

	if __name__ == "__main__": main()
```

Q2 functions. Py:

```
def windowSignal(signal,windowingFunction):
		numberOfWindows = int(len(signal.data)/len(windowingFunction));
		windowSize = len(windowingFunction);
		for i in range (0, numberOfWindows):
			currentSamples = signal.data[:windowSize];
			signal.data[:windowSize] =
np.multiply(currentSamples,windowingFunction);

def plotMagnitudeSpectrum(signal,sizeOfFFT):
	freq = np.fft.fft(signal.data,sizeOfFFT)/len(signal.data);
	magn_spectrum = np.abs(freq);
	#loudest = np.amax(magn_spectrum);
	magn_spectrum = 20*np.log10(magn_spectrum);
	lin_space = np.linspace(0,sizeOfFFT/2,sizeOfFFT/2);
	plt.ylabel('Amplitude in dB');
	plt.xlabel('Frequency bin')
	#plt.title('MagnitudeSpectrum');
	plt.plot(lin_space, magn_spectrum[:sizeOfFFT/2],'r');
	plt.show();
```

```
def plotWindowedAndNonWindowed(signal,sizeOfFFT,hanningWindow):
	plotMagnitudeSpectrum(signal,sizeOfFFT);

	windowSignal(signal,hanningWindow);
	plotMagnitudeSpectrum(signal,sizeOfFFT);
```

Appendix D : Question2_b.py

```
class question2_b():
	def main():
		sign = mir.Signal();
		relative_path = 'wav_files/flutehighMono.wav'
		sign.wav_read(relative_path);
		plt.plot(sign.data);
		plt.show();
		sizeOfFFT = 2048;
		sizeOfBin = 44100/float(sizeOfFFT);
		hanning_window = np.hanning(2048.0);
		question2functions.plotMagnitudeSpectrumReduced(sign,sizeOfFFT);
		plt.show();
		question2functions.windowSignal(sign,hanning_window);
		question2functions.plotMagnitudeSpectrumReduced(sign,sizeOfFFT);
		plt.show();
		return 0;
	if __name__ == "__main__": main()
```

Note: The 2048 initial samples were selected from sign by doing sign.data[:2048] inside Magnitude Spectrum reduced.

Appendix E: Q4 : AdditiveSynthesis.py.

```
		list_of_frequencies_and_amplitudes =
[[50,1],[100,0.75],[200,0.5],[400,0.25],[800,0.5],[1600,0.75]];
		instrumentData = np.zeros(44100,);
		for sublist in list_of_frequencies_and_amplitudes:
			sinTemp = mir.Sinusoid(freq = sublist[0], amp = sublist[1]);
			instrumentData = np.add(instrumentData,sinTemp.data);
		outputSignal = mir.Signal(data = instrumentData);
		outputSignal.wav_write('outInstrument.wav');
		inputSamples = outputSignal.data;
		sizeOfFFT = 4096;
		numberOfBins = sizeOfFFT;
		freq = np.fft.fft(inputSamples,sizeOfFFT);
		real_part = np.real(freq);
		imag_part = np.imag(freq);
		magn_spectrum = np.absolute(freq)/len(inputSamples);
		lin_space = np.linspace(0,sizeOfFFT,sizeOfFFT);
		plt.title('Library FFT');
		print len(lin_space);
		print len(magn_spectrum);
```

```
                plt.plot(lin_space[:sizeOfFFT/10], magn_spectrum[:sizeOfFFT/10],'r');

                plt.show();
```

Appendix F. Synthesis of Melody. Melody.py. - I skip the imports.

```
class Melody():

        def main():
                        list_of_MIDI_values = [72, 12,12, 20, 24, 67, 12,12, 20, 24, 64,
12, 20, 24, 69, 12, 20, 12, 71, 12, 20, 70, 12, 69, 20, 12, 67, 16, 76, 16, 79, 16, 81,
12, 20, 12, 77, 12, 79, 12, 20, 12, 76, 12, 20, 12, 72, 12, 74, 12, 71, 12, 20, 24];
                f_and_amp =
[[1300,1],[3600,1],[3700,1],[3800,1],[4500,0.5],[5300,1],[8400,0.25],[10700,0.5],[13000
,0.25],[19000,0.01]];
                ratios = np.zeros(len(f_and_amp),);
                for k in range(0,len(f_and_amp)):
                        ratios[k] = f_and_amp[k][0]/f_and_amp[0][0];
#trying to get this played into a file - must use sequence in mir.py file.
                seed = mir.Sinusoid(freq = 0, duration = 0.09);
                sequence = mir.Sequence(seed);
                for i in range(0,len(list_of_MIDI_values)):
                        #calculate freq with the formula
                        #A is 440 Hz
                        difference = (list_of_MIDI_values[i]-69);
                        exponent = difference/12.0;
                        f = 440.0*(2.0**(exponent)); #we can see use 12th root of 2
                        seed = mir.Sinusoid(freq = 0, duration = 0.09);
                        mixture = mir.Mixture(seed);
                        for k in range(0,len(ratios)):
                                if ((ratios[k]*f) >= 500.0):
                                        sinTemp = mir.Sinusoid(freq = ratios[k]*f,
duration = 0.09, amp = 0.5*f_and_amp[k][1]);
                                else:
                                        sinTemp = mir.Sinusoid(freq = 0, duration = 0.09,
amp = 0);
                                mixture = mir.Mixture(mixture,sinTemp);
                        sequence = mir.Sequence(sequence,mixture);
                outputSignal = sequence;
                outputSignal.wav_write('outMelody.wav');
                return 0;

        if __name__ == "__main__": main()
```

Appendix G: **q5.py -** BUFFERED INPUT in Matlab.

```
clc;
clear;

filename = 'violaCsharp.wav';
[inputSamples,Fs] = audioread(filename);
L = length(inputSamples);
windowSize = 2048;
num_Avail_Windows = floor(L/windowSize);
sizeOfFFT = 2048;
fftData = zeros(sizeOfFFT,1);
inversefftData = zeros(windowSize,1);
inversefftSamples = zeros(1,1);

for i = 1:num_Avail_Windows-1

    curr_input_samples = inputSamples(i*windowSize:(i+1)*windowSize);

    Y = fft(curr_input_samples,sizeOfFFT);
    fftData = fftData + Y; %real and imaginary

    inversefftData = ifft(Y,windowSize);

    inversefftSamples = [inversefftSamples; inversefftData];

end

disp('sounding');
soundsc((1/L).*abs(inversefftSamples),Fs);
disp('sounded');
audiowrite('outInverseFFT.wav',(1/L).*abs(inversefftSamples),Fs);

magn_spectrum = abs(real(fftData)/L);
magn_spectrum = magn_spectrum(1:sizeOfFFT/2);

f = (Fs/sizeOfFFT).*(0:sizeOfFFT/2-1);

plot(f,magn_spectrum);

title 'Magnitude Spectrum'
ylabel 'DFT magnitude'
xlabel('f (Hz)')
ylabel('|Magnitude|')
```

Appendix H: q5mostlyphases.py

```
clc;
clear;
filename = 'bellCsharp.wav';
[inputSamples,Fs] = audioread(filename);
L = length(inputSamples);
windowSize = 2048;
num_Avail_Windows = floor(L/windowSize);
sizeOfFFT = 2048;
fftData = zeros(sizeOfFFT,1);
inversefftData = zeros(windowSize,1);
inversefftSamples = zeros(1,1);
for i = 1:num_Avail_Windows-1
    curr_input_samples = inputSamples(i*windowSize:(i+1)*windowSize);
    Y = fft(curr_input_samples,sizeOfFFT);
    %transform to polar and randomize phases
    [theta, ro] = cart2pol(real(Y),imag(Y));
    %find the 4 highest peaks
    n = 4;
    [sortedX,sortingIndices] = sort(ro,'descend');
    maxValues = sortedX(1:n);
    maxValueIndices = sortingIndices(1:n);

    %set to zero and re insert the maxValues
    ro = zeros(length(ro),1);

    for i = 1:length(maxValues)
        ro(maxValueIndices(i,1)) = maxValues(i,1);
    end
    %transform back to cartesian
    [re,im] = pol2cart(theta,ro);
    Y = complex(re,im);
    fftData = fftData + Y; %real and imaginary
    inversefftData = ifft(Y,windowSize);
    inversefftSamples = [inversefftSamples; inversefftData];
end
disp('sounding');
soundsc((1/L).*abs(inversefftSamples),Fs);
disp('sounded');
audiowrite('outInverseFFT.wav',(1/L).*abs(inversefftSamples),Fs);
magn_spectrum = abs(real(fftData)/L);
magn_spectrum = magn_spectrum(1:sizeOfFFT/2);
f = (Fs/sizeOfFFT).*(0:sizeOfFFT/2-1);
```

```
plot(f,magn_spectrum);
title 'Magnitude Spectrum'
ylabel 'DFT magnitude'
xlabel('f (Hz)')
ylabel('|Magnitude|')
```