Hector Perez - CSC 475

Assignment 1

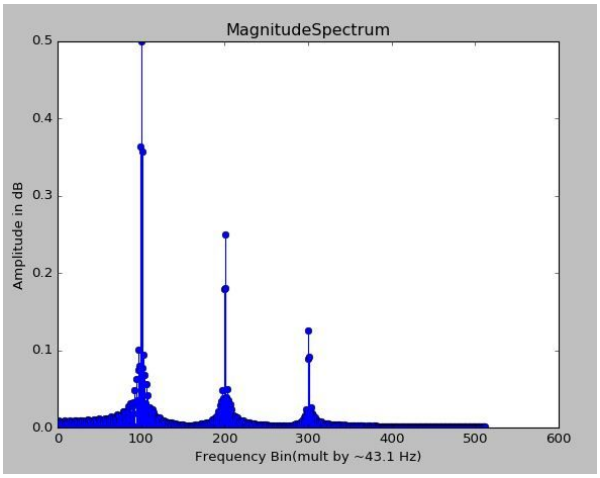
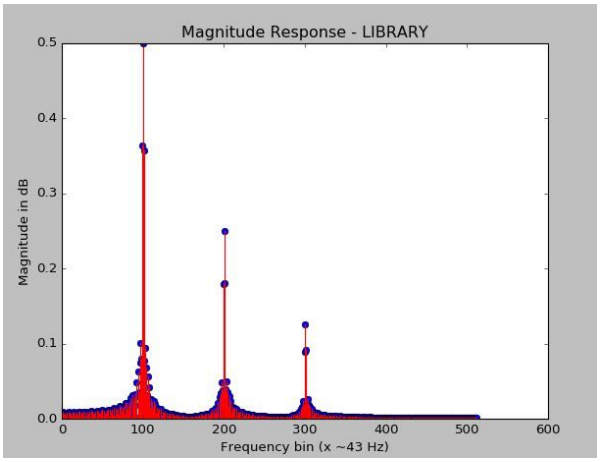
Question 1.

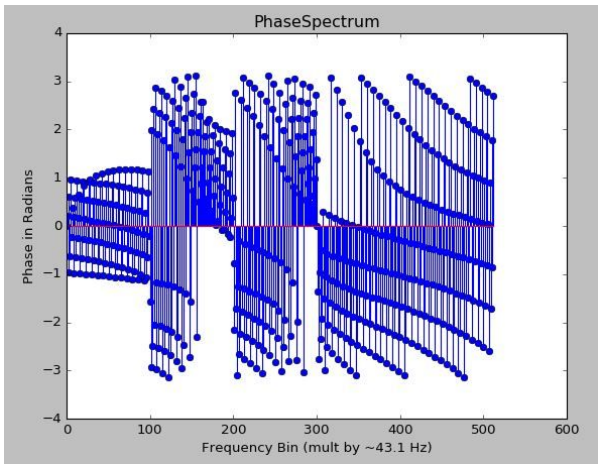
1A. Listing of Code and FFT performed with programmed routine and with the library function. Code is in Python: included in **Appendix A.** ***- Q1 - mainProg1.py and methodsForFFT.py***

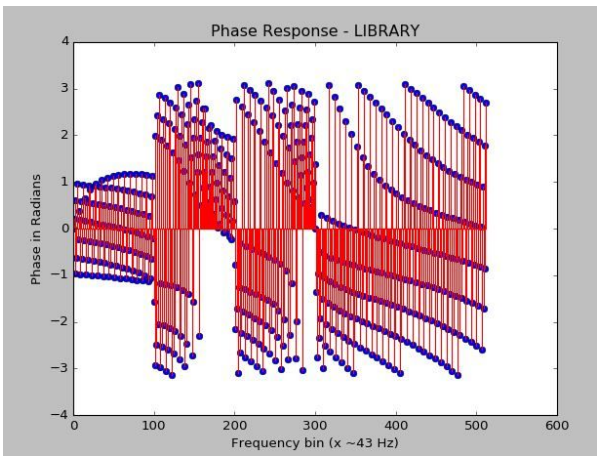
EDIT: In **Appendix I,** I include a plot with the 'unwrapped' according to source shared. And modified code for the phase response.

Note:The np.fft.fft function plot matches the plot produced by the code, the phase response is noisy.

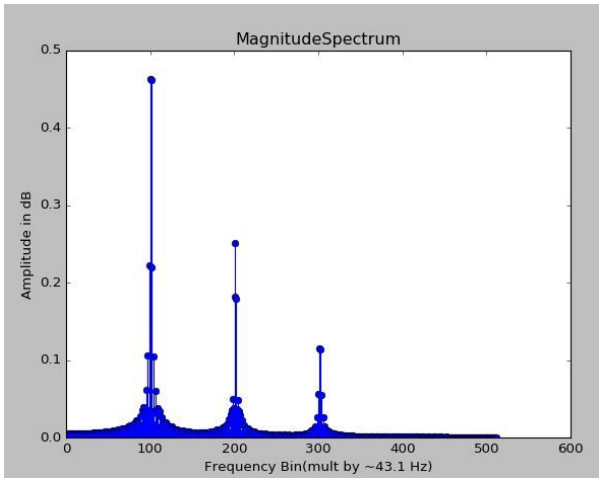
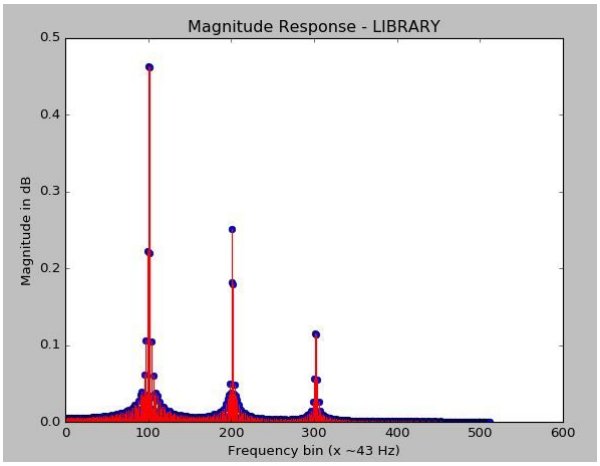
1B. Test FFT implementation with 3 harmonically related sinusoids (PLOT).
B.1 Exact fundamental = 100*43.1. Magnitude and phase plots compared.
The fs = 44100 and DFT size = 1024. Size of Freq Bin is ~43.1 HZ.

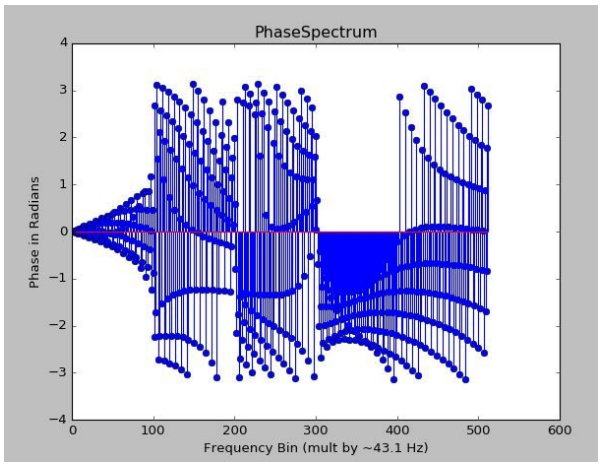
Programmed FFT	Library Function
	

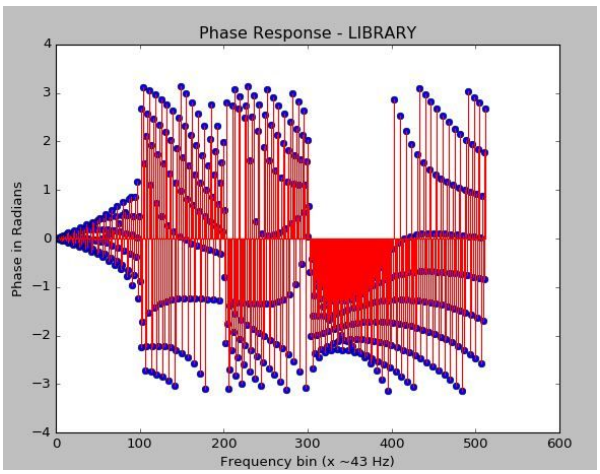




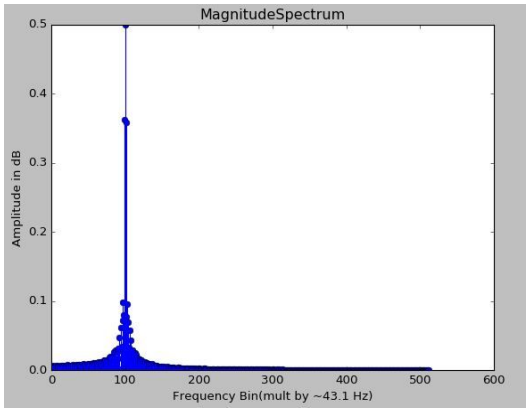
B.2. Fundamental between bins.
Sinusoids had frequencies corresponding to bins 50.5, 103.34 and 206.7 (2227, 4454.1 and 8908.2Hz).

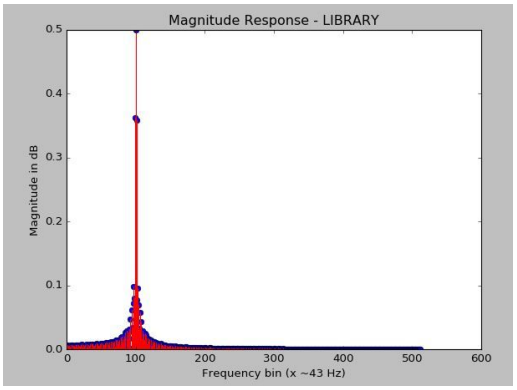
Programmed FFT	Library Function
	

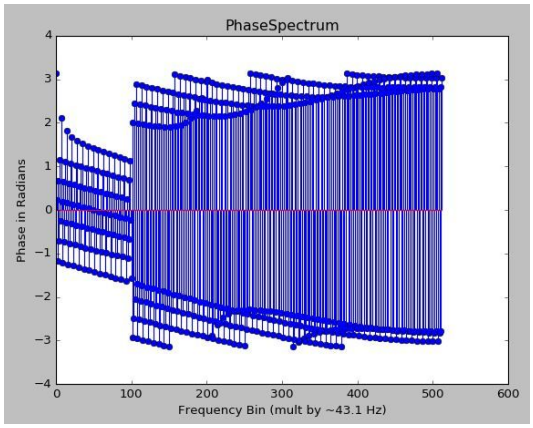


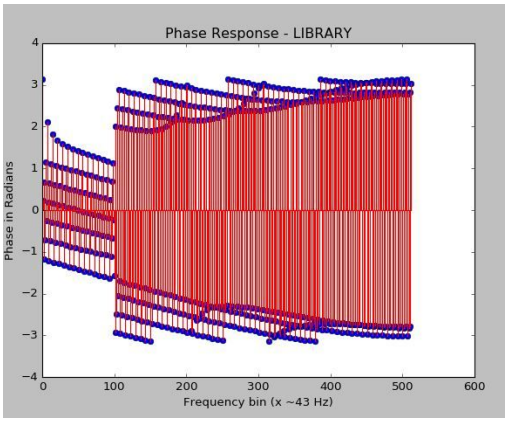


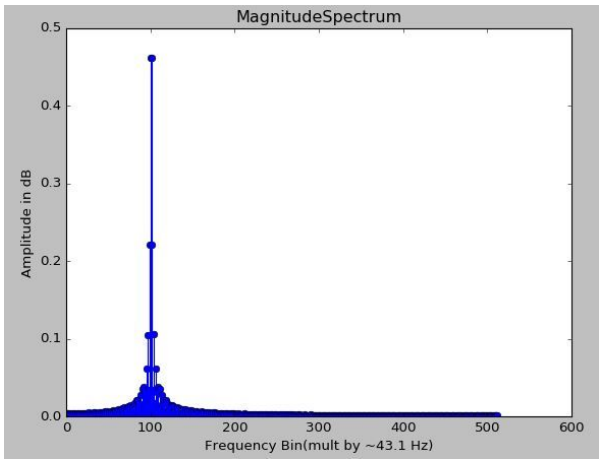
Testing with the fundamentals only for the exact 100 bin and the 100.5 bin:

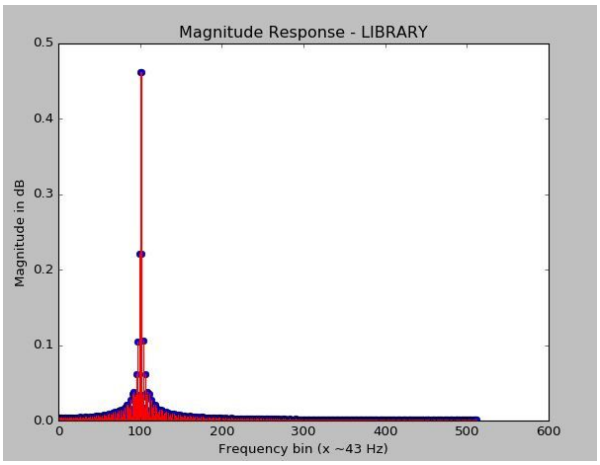


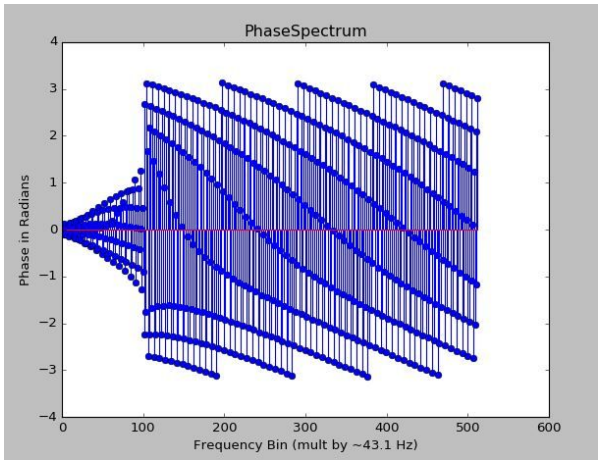


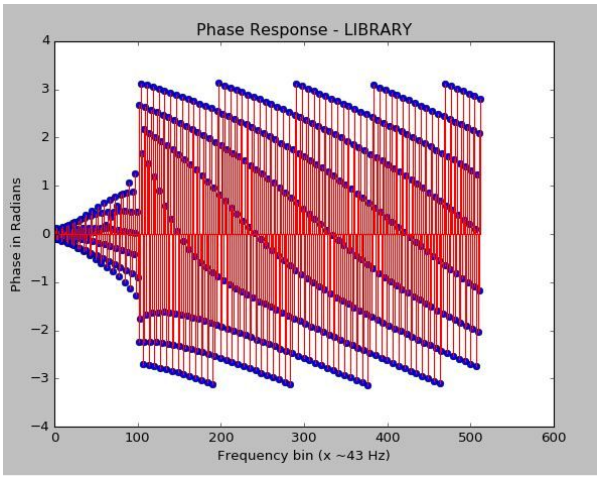








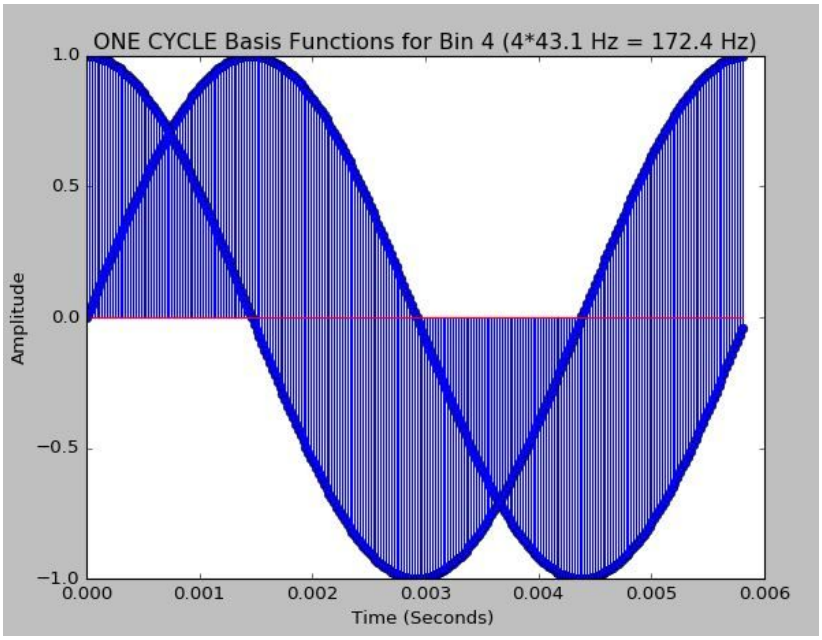




Discussion: I have had experience with this issue before. I could not reproduce the effect expected. The FFT for the signal with an exact bin frequence should be at the bin. Then, the one that is between bins should show frequency leaking (because it's not exactly at a bin, the

energy for it will be ‘split’ by the closest bins. I also believe something is wrong with my phase plots.

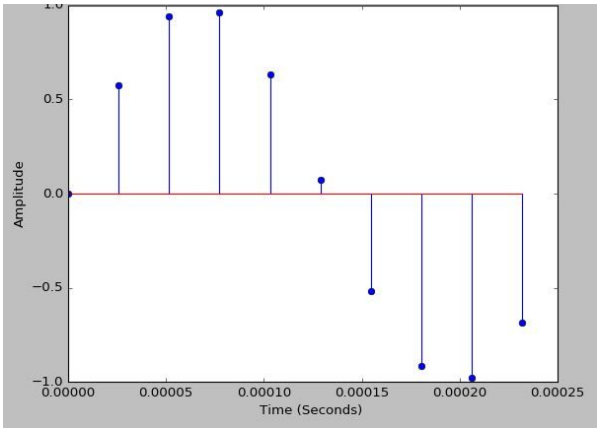
C. Plot basis functions for a particular DFT bin - The size of my FFT is 256. The basis functions for bin 4 are plotted here.
The code for this is in **Appendix B:** *BasisFunctions.py*

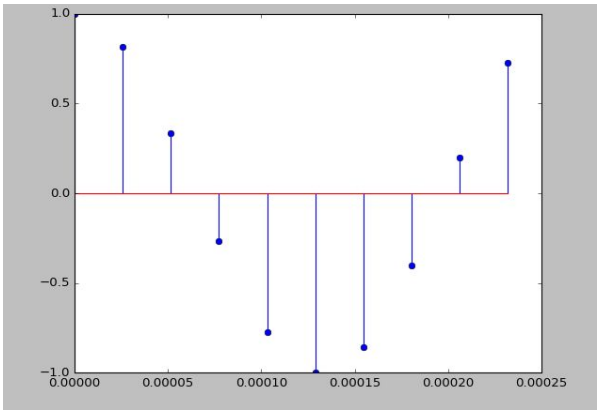


This is the two basis functions plotted for 1 cycle. T= 1/172.4 = 0.0058 seconds, and the number of samples is 44100 per second.
This means that there are 255 samples per cycle.

D. Pointwise multiplying.

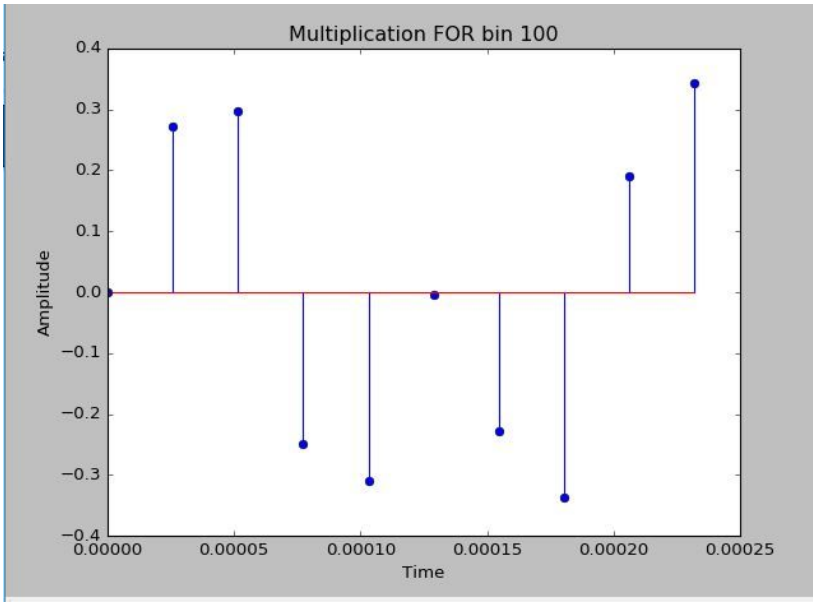
D.1. Input fundamental multiplied with basis functions corresponding to closest bin.
The corresponding bins to the first fundamental is bin 100. Its basis functions are:



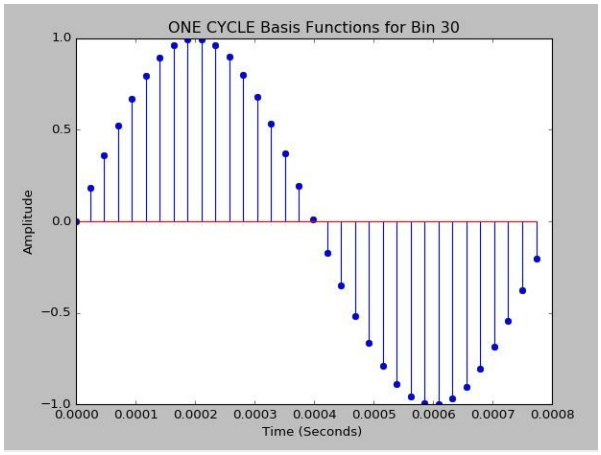


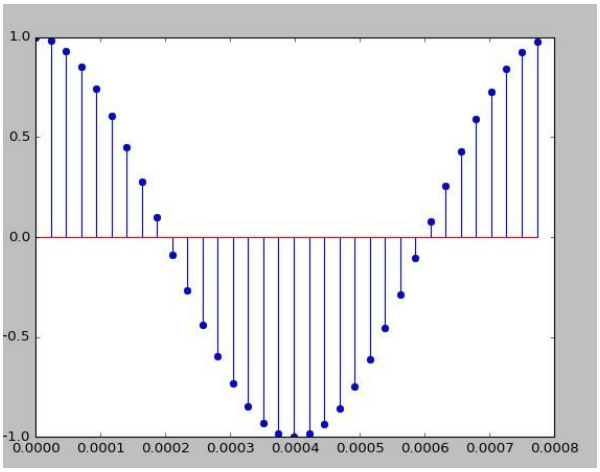
The frequency is 100*43.1 and the period is 0.0002 seconds. Number of samples per one cycle is 10. (I decided to plot a single cycle).

The pointwise multiplication with a sinusoid of 4310H Hz = bin 100 with bin 100 basis functions.

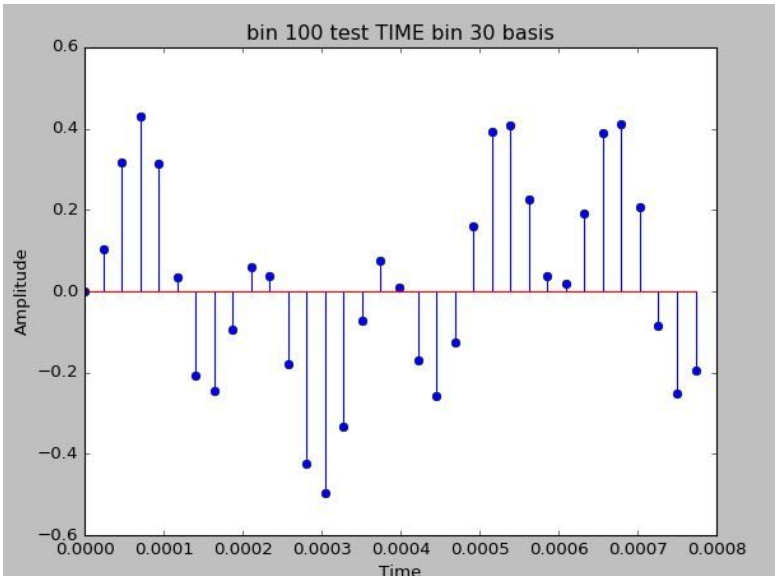


D.2 Input fundamental multiplied with basis functions corresponding to unrelated bin.





The pointwise multiplication with a sinusoid of 4310H Hz = bin 100 with bin 30 basis functions. The cycle of bin 100 is smaller than the cycle for bin 30.



D.3 Discussion.

In class it was discussed that when multiplying a sinusoid by an unrelated bin basis, the inner product should be close to zero. This mean 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 has phase, and it were to be multiplied with only one the basis functions (sine or cosine) then we might get low results, because the correlation doesn't show that they would have matched, had the sinusoid not been phased. Multiplying by both a sinusoid and a cosine at the same time helps balance this, and gives us information about the phase.

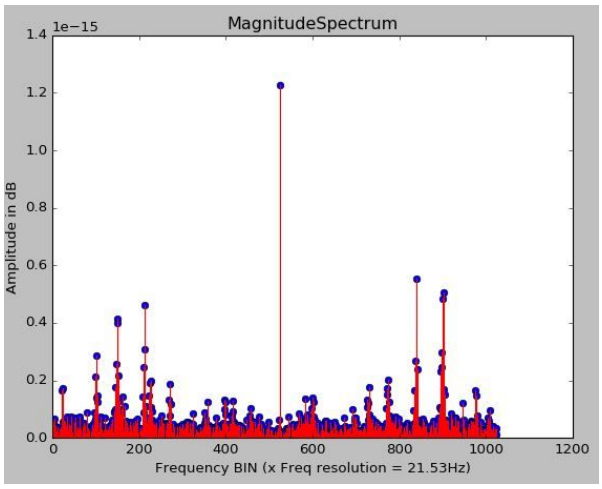
Question 2:

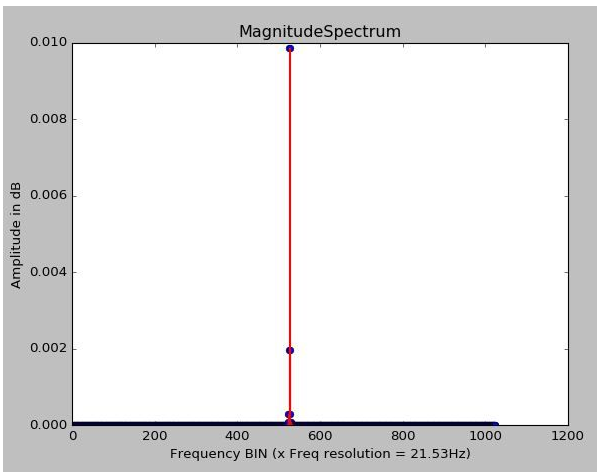
First Part (A) : Using an existing FFT implementation, plot: Magnitude response in dB. Use DFT size 2048 of a sine wave that has frequency corresponding to the bin 525.

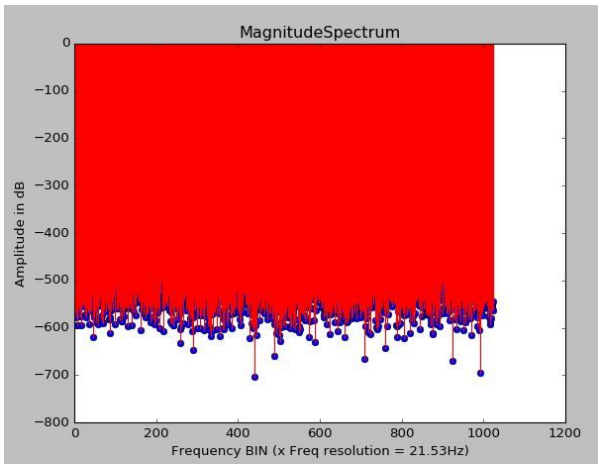
Code: **Appendix H:** question2.py and question2functions.py

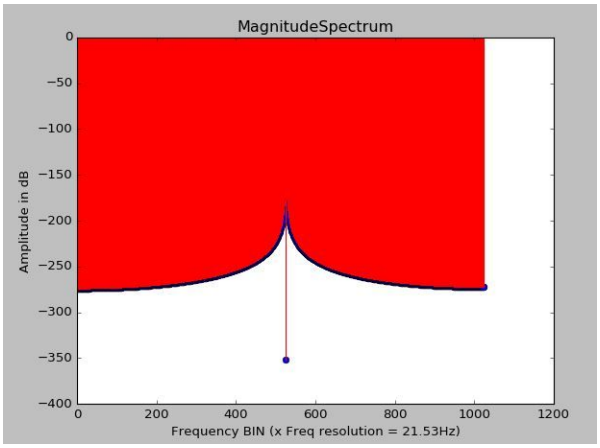
Size of bin is roughly 21.53 Hz so the 525th bin would be frequency 11,303.35 Hz approx.

Non windowed (NOT dB)	Windowed (NOT dB)



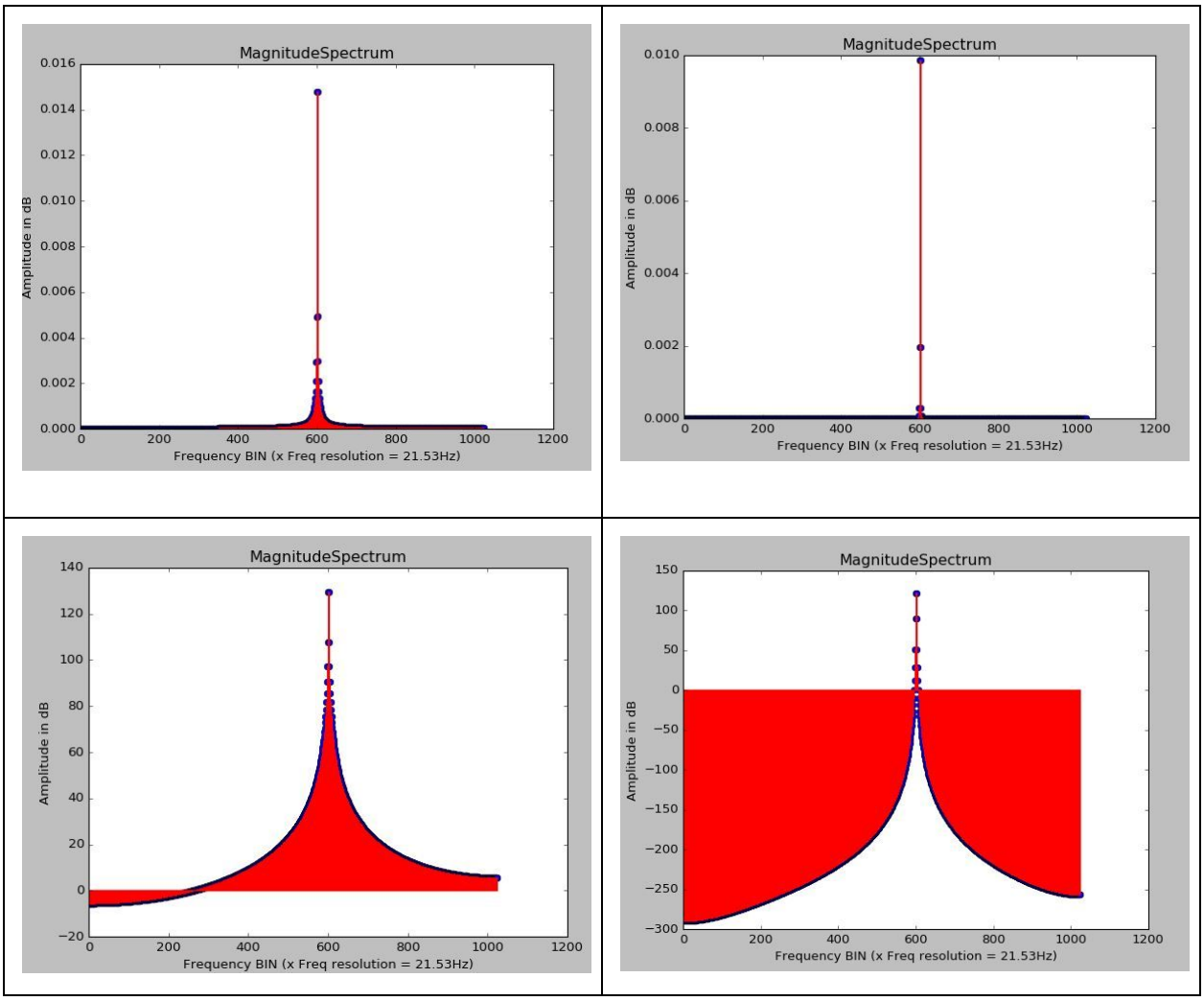






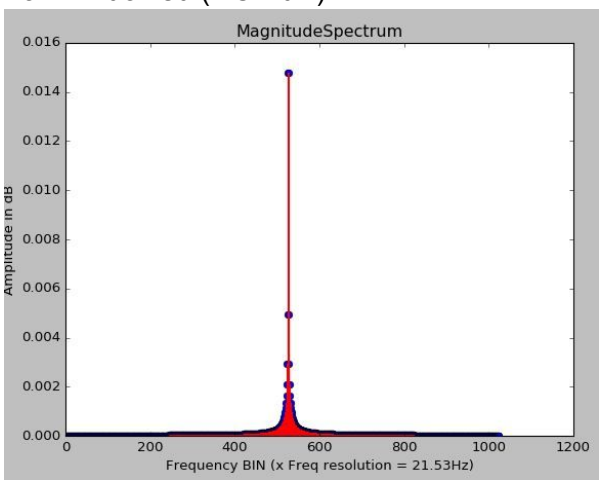
Magnitude response in dB of a sine wave with freq corr. to bin 600.5.
Size of bin is roughly 21.53 Hz so the 525th bin would be frequency 12,928 Hz approx.

Non windowed (NOT dB)	Windowed (NOT dB)

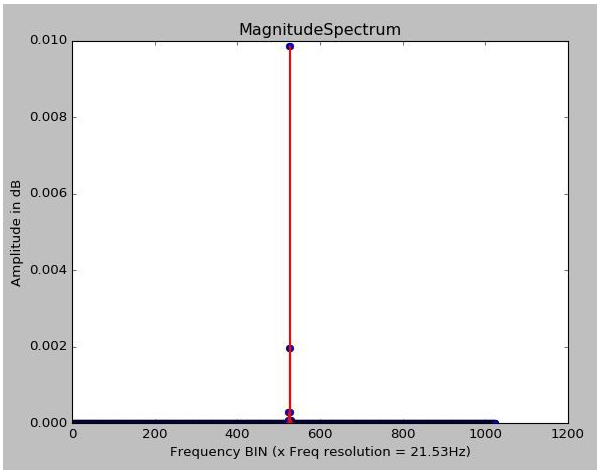


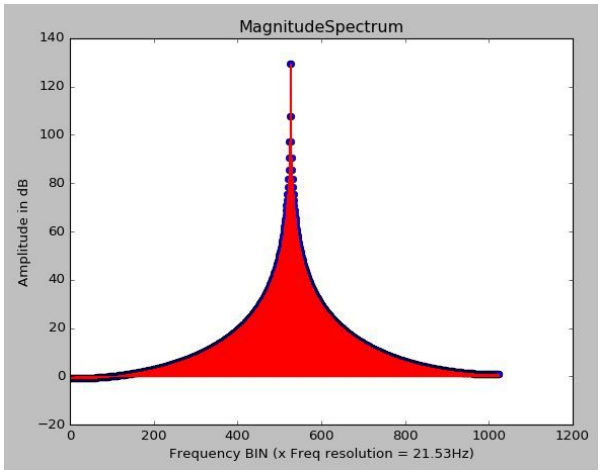
Magn. resp. In dB of a hanning - windowed (size of window 2048) DFT of a sinusoid w/ frequency corr. to bin 525.5.
Size of bin is roughly 21.53 Hz for fs = 44100 so the 525th bin would be frequency 11,314 Hz

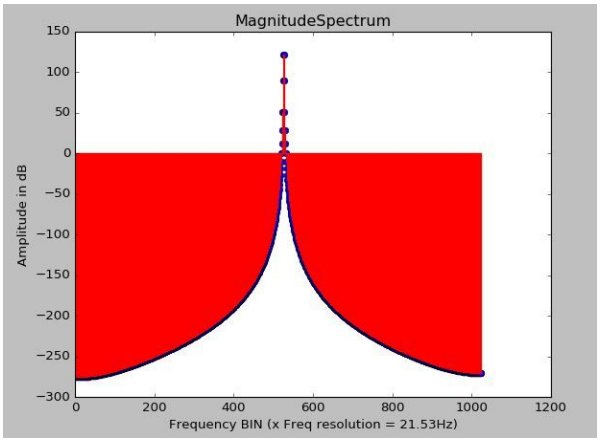
Non windowed (NOT dB)



Windowed (NOT dB)







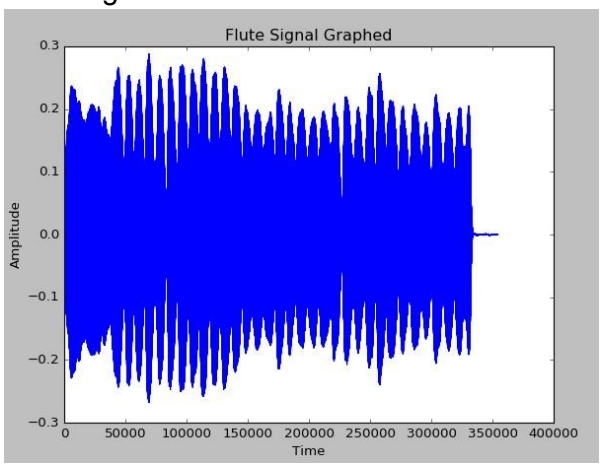
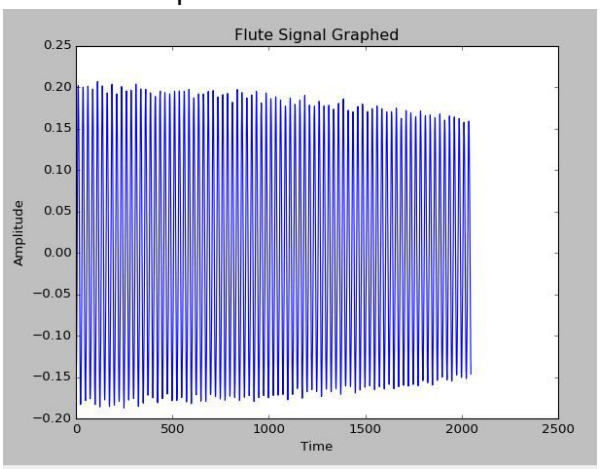
Discussion Question 2 A: All the plots show that windowing the function before performing the FFT (multiply by hanning window in the time domain) will ‘clean up’ the magnitude response. The frequencies that are initially found when not windowing are ‘filtered out’. If a signal has discontinuities, then 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 : Second Part:

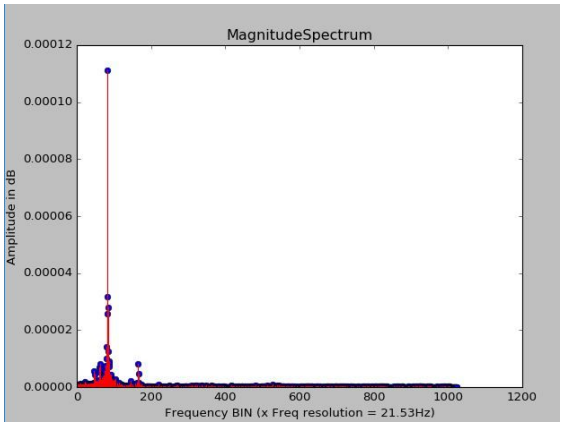
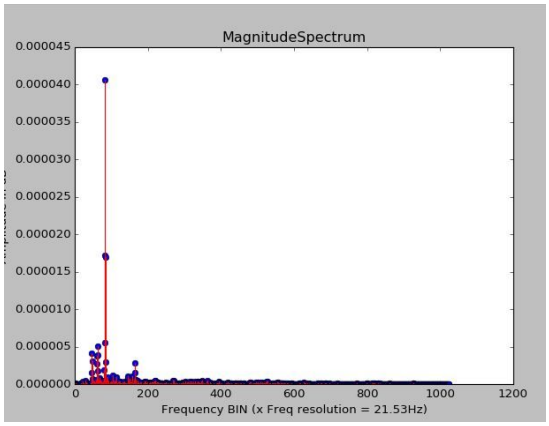
a. Find single not of an instrument - select 2048 samples of audio from steady state of sound. (Violin, flute,organ … etc). Window as before, and Perform FFT on it.

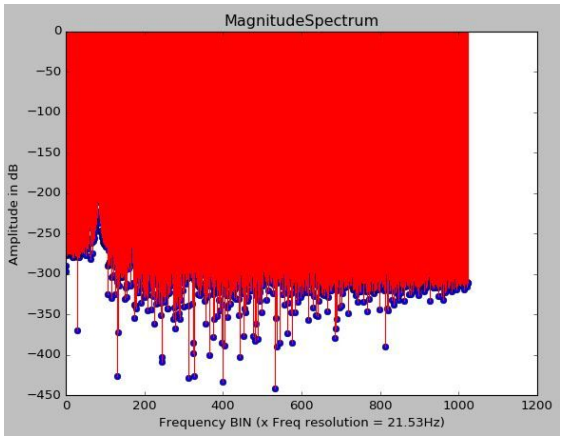
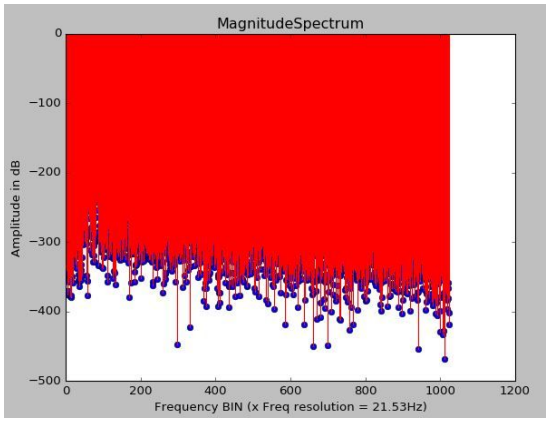
Plot Magnitude response:

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.

	Non Windowed	Windowed
non dB		

dB		

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.

Question 3:
Read the article “Music Retrieval: A Tutorial and Review" by Nicola Orio

A. Pick a piece of music that you like and try to characterize it using the time scales and music dimensions discussed in the paper. Try to be specific about the particular music piece. (4/3 points)

I decided to characterize ‘Walking on the moon’ by Sting and the police.
http://www.e-chords.com/tabs/the-police/walking-on-the-moon
http://stewartgreenhill.com/ukulele/WalkingOnTheMoon.html

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. For the same piece of music describe specifically what types of information needs the different users mentioned in the MIR overview would be interested in. (4/3 points)

The article mentions more specific musical information that could be extracted from scores and performances. However, I believe other information could be useful (the name of the song, name of performer, genre). The article also mentions that the information is useful in different degrees to people with different musical expertise. For example, an experienced musician/performer will get more out of a score).

Information that can come from scores:
General Parameters: Main tonality. Modulations. Time signatures. Tempi, musical form and repetitions.
Local Parameters: Specific events that must be played by instruments - duration of events and intensity of events. Extract Melody, Harmony and Rhythm.

Information retrievable from performances:
Differences caused by the individual's interpretation of a score. Musicologists, music theorists, musicians can use this.
Differences in orchestration for alternate performances (different instrumentation/ arrangement).

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.

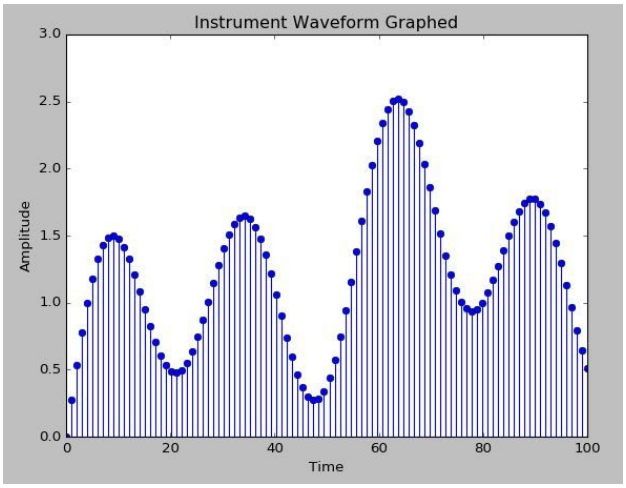
Lastly, to recognize the chords, the incoming signal is compared with each of these chord templates. I am not so clear about how many profiles to make, because there are many different kinds of chords (thinking about 7th chords, which are almost like the regular chords, but with one extra note).

KEEP GOING (Question 4 below).

Question #4.

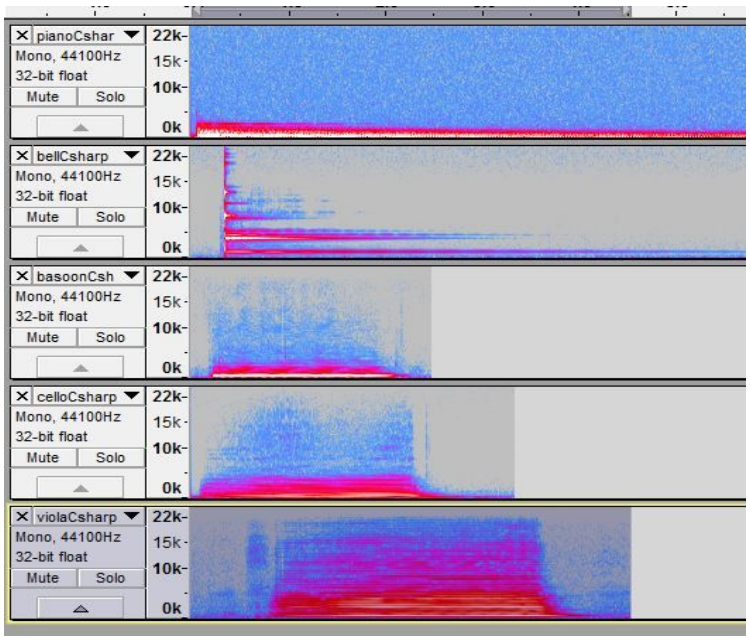
A.Code for additive synthesis instrument (5 points).

Included in ***Appendix C,*** as Code for *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]]; Then I added the sinusoids and generated this plot for the first 100 samples of the 44100 samples result (1 sec audio) and the DFT.



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

Code is a variation on the AdditiveSynthesis.py. (In appendix C) I just changed the frequencies and amplitudes of the sinusoids to be added.

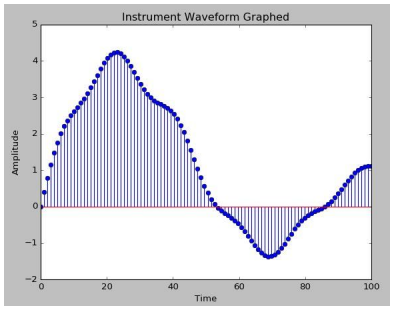


I found that the spectrums for the piano, bassoon, viola and cello look similar. The one for the bells looks like it has more harmonics. Easy to distinguish between the bells and everything else, but not between the others.

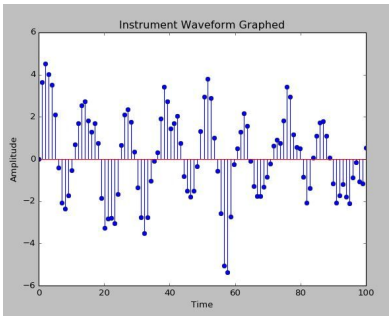
What would be missing probably is the variation in energy over time for the different frequencies that make up the sound. Attack,delay,sustain,release would make the sound more realistic.

Plots for resulting waveforms and list of frequencies and amplitudes:

Piano

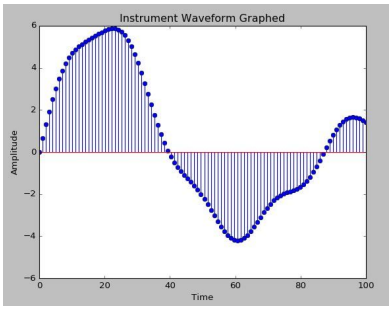


Bell

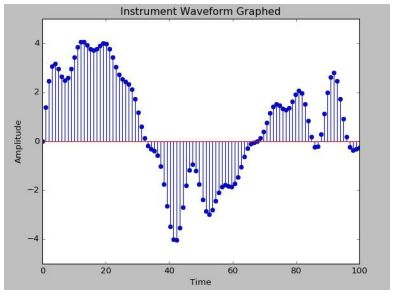


Basoon

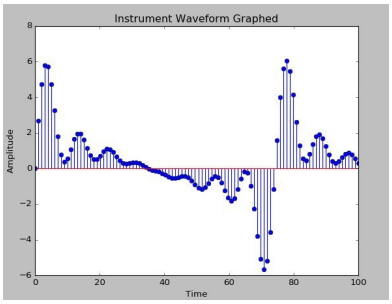
Looks very similar to Piano.



Cello



Viola



For Piano, the list of frequencies and amplitudes is:
[[400,1],[500,1],[600,1],[1000,0.1],[300,1],[1200,0.1],[2300,0.1],[2400,0.1],[2500,0.1],[100,1]];
(You can just put this into AdditiveSynthesis.py and it will generate the signal).
For Bells:
[[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]];
For Basoon:
[[500,1],[600,1],[400,1],[700,1],[550,1],[450,1],[800,0.25],[1000,0.25],[2000,0.25],[2050,0.25]];
For Cello:
[[400,0.25],[500,0.75],[1000,0.5],[2000,0.5],[3000,0.5],[4000,0.5],[5000,0.5],[700,1],[650,1],[750,1]];
For Viola:
[[600,1],[1200,1],[1800,1],[2400,1],[3000,1],[3600,1],[4200,1],[4800,0.5],[600,0.25],[700,0.25]];

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 is in Appendix D: *Melody.py* Recreated the Mario Bros. theme.

Facts used: (I didn't notice this formula was already in mir.py)

Equal temperament uses the 12th root of 2 (because pitch perception is logarithmic) to define the ratio between two adjacent semitones = 1.059463. Each semitone is divided into 100 cents for 12 tone equal temperament.

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:

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

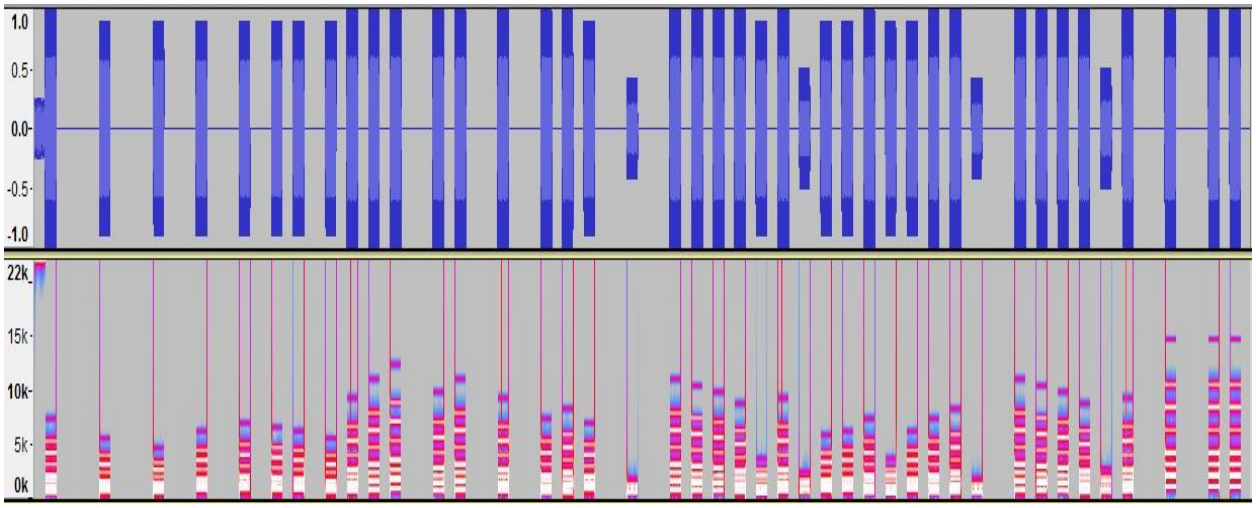
This looks very similar to the formula for calculating a frequency in 12 T.E.T.: (It is the same in fact, because 'a' in this formula refers to the reference frequency!)

$$P_n = P_a(\sqrt[12]{2})^{(n-a)}$$

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:



Question #5.

A. *Write code that reads buffers of data from the audio file, converts them to a complex spectra using the Fast Fourier Transform, converts them back to the time domain using the inverse FFT and writes the output to a file. Confirm that things are working by checking that the output file is identical to the input i.e there is no loss of information from performing the FFT and inverse FFT. (10/5 points)*

The code is in Appendix E: *q5.py* (Question 5 first part) AND *q5Alt2.py*

I wrote this code in MATLAB. There were some issues with the sound recovery through IFFT. I copy my code and code that would normally reproduce the sound identically (the second one doesn't buffer from the input- it was easier in MATLAB to do it this way - and since it did not alter the sound of the input, I used this for the last part). I decided to the one in which the input wasn't buffered to do the FFT so that the sound differences would be noticeable.

B. Convert the complex spectrum from real and imaginary coefficients to polar form i.e magnitude and phase. Replace the phases with random numbers, convert back to real and imaginary coefficients and perform the inverse FFT. What is the effect on the underlying audio ? Experiment with different sizes of window (256; 512; 1024; … ; 32768). Explain what you hear based on your understanding of the DFT in 2 sentences (5/5 points)

The code is in Appendix F: *q5randomphases.py* (Question 5 second part)

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. Perform a similar analysis to above but select the 4 highest peaks in the magnitude spectrum and set all the other magnitudes to zero while retaining the phases. Convert back to real and imaginary coefficients and perform the inverse FFT. What is the effect on the underlying audio ? Explain what you hear based on your understanding of the DFT in 2-3 sentences (5/5 points)

The code is in Appendix G: *q5mostlyphases.py* (Question 5 third part)

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

Appendix A. *- Q1 - mainProg1.py and methodsForFFT.py*

mainProg1.py

(Must_import mir, import numpy as np, import methodsForFFT , import wave ,import os ,import struct ,import matplotlib.pyplot as plt)

```
class mainProg1():
	def main():
		#---------plotting with self implemented DFT-------------------------------#
		#-------------------------------------------------------------------------------------#
		#-------------------------------------------------------------------------------------#
		#FS = 44100 and DFT size = 1024. Size of Freq Bin is ~43.1 HZ.
		#Can represent correctly frequencies of up to 22050 Hz.
		#Bin 50 is 2155Hz.
		sizeOfFFT = 1024; #size of fft ( k = N , in this case)
		non_Aliasing_Points = sizeOfFFT/2;
		sizeOfBin = 44100.0/sizeOfFFT;
		fundamental1 = 100*sizeOfBin;
		fundamental2 = 100.5*sizeOfBin;

		#PART A - fundamental at freq bin
		#sine1 = mir.Sinusoid(amp = 1,freq = fundamental1,phase = 0, duration = 0.01);
		#sine2 = mir.Sinusoid(amp = 0.5,freq = 2*fundamental1, phase = np.pi/4, duration = 0.01);
		#sine3 = mir.Sinusoid(amp = 0.25,freq = 3*fundamental1, phase = np.pi/2, duration =0.01);
		#signal1 = mir.Mixture(sine1,sine2,sine3);
		#inputSamples1 = signal1.data;
		#methodsForFFT.dftByHector(non_Aliasing_Points,signal1,inputSamples1);
		#PART A - fundamental at freq bin

		#PART B - fundamental between freq bin
		#sine1 = mir.Sinusoid(amp = 1,freq = fundamental2,phase = 0, duration = 0.01);
		#sine2 = mir.Sinusoid(amp = 0.5,freq = fundamental2*2, phase = np.pi/4, duration = 0.01);
		#sine3 = mir.Sinusoid(amp = 0.25,freq = fundamental2*3, phase = np.pi/2, duration =0.01);
		#signal2 = mir.Mixture(sine1,sine2,sine3);
		#inputSamples2 = signal2.data;
		#methodsForFFT.dftByHector(non_Aliasing_Points,signal2,inputSamples2);
		#PART B - fundamental between freq bin

		#----------------'plotting with library function for DFT'------------------#
		#-------------------------------------------------------------------------------------#
		#-------------------------------------------------------------------------------------#
		freq = np.fft.fft(inputSamples,sizeOfFFT)/len(inputSamples);
		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 in dB');
		plt.xlabel('Frequency bin (x ~43 Hz)');
		plt.stem(lin_space, magn_spectrum[:non_Aliasing_Points],'r');
		plt.show();
```

```
                plt.title('Phase Response - LIBRARY');
                plt.ylabel('Phase in Radians');
                plt.xlabel('Frequency bin (x ~43 Hz)');
                plt.stem(lin_space, phase_spectrum[:non_Aliasing_Points],'r');
                plt.show();

                return 0;


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

methodsForFFT.py

(Must_import mir, import numpy as np, import wave ,import os ,import struct ,import matplotlib.pyplot as plt)

```
def dftByHector(non_Aliasing_Points,sign,inputSamples):
        re_in = inputSamples;
        im_in = np.zeros(len(inputSamples),);
        real_part = np.zeros (non_Aliasing_Points,);
        imaginary_part = np.zeros (non_Aliasing_Points,);

        sizeOfBinInHertz = 44100.0/1024.0;

        for i in range (0,non_Aliasing_Points):
                temp_cosine = mir.Sinusoid(duration = sign.duration, freq = i*sizeOfBinInHertz, phase = np.pi/2);
                temp_sine = mir.Sinusoid(duration = sign.duration, freq = i*sizeOfBinInHertz , phase = 0);

                for j in range (0,len(inputSamples)):
                        real_part[i] += (re_in[j] * temp_cosine.data[j]) + (im_in[j] * temp_sine.data[j]);
imaginary_part[i] += (-1 * re_in[j] * temp_sine.data[j]) + (im_in[j] * temp_cosine.data[j]);
        plotDFT(non_Aliasing_Points, sign,real_part,imaginary_part);
        return 0;

def plotMagnitudeSpectrum(non_Aliasing_Points, sign,real_part,imaginary_part):
        magnitude_spectrum = np.zeros (non_Aliasing_Points,);
for i in range(0,non_Aliasing_Points):
                magnitude_spectrum[i] = np.sqrt(imaginary_part[i]**2+real_part[i]**2);
                magnitude_spectrum[i] = magnitude_spectrum[i]/(len(sign.data));
        arr2 = magnitude_spectrum;
        arr1 = np.linspace(0,non_Aliasing_Points,non_Aliasing_Points);
        print 'length(arr1)';
        print len(arr1);
        plt.ylabel('Amplitude in dB');
        plt.xlabel('Frequency Bin(mult by ~43.1 Hz)')
        plt.title('MagnitudeSpectrum');
        plt.stem(arr1,arr2);
        return 0;

def plotPhaseSpectrum(non_Aliasing_Points, sign,real_part,imaginary_part):
        phase_spectrum = np.zeros(non_Aliasing_Points,);
        for i in range (0,non_Aliasing_Points):
                phase_spectrum[i] = np.arctan2(imaginary_part[i],real_part[i]);
        arr4 = phase_spectrum;
        arr3 = np.linspace(0,non_Aliasing_Points,non_Aliasing_Points);
        plt.title('PhaseSpectrum');
```

```
        plt.ylabel('Phase in Radians');
        plt.xlabel('Frequency Bin (mult by ~43.1 Hz)');
        plt.stem(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;
```

Appendix B: *BasisFunctions.py*

(Must import mir , import numpy as np,import wave,import os,import struct,import matplotlib.pyplot as plt)

```
class BasisFunctions():
        def main():
                SizeOfBin = 43.1;
                NumBin = 200;
                TestBin = 100;
                BasisFreq  = float(NumBin*SizeOfBin);
                TestFreq = TestBin*SizeOfBin;
                testSine = mir.Sinusoid(freq = TestFreq);
                sine1 = mir.Sinusoid(freq = BasisFreq);
                cosine1 = mir.Sinusoid(freq = BasisFreq, phase = np.pi/2);
                lengthOfCycle = float(1/BasisFreq);
                Fs = 44100.0;
                t = BasisFreq/Fs;
                numSamplesPerCycle = (Fs*lengthOfCycle);
                arr1 = np.linspace(0.0,lengthOfCycle,numSamplesPerCycle);
                arr2 = sine1.data[:numSamplesPerCycle];
                arr3 = cosine1.data[:numSamplesPerCycle];
                arr4 = testSine.data[:numSamplesPerCycle];
                plt.title('ONE CYCLE Basis Functions for Bin 30');
                plt.ylabel('Amplitude');
                plt.xlabel('Time (Seconds) ');
                plt.stem(arr1,arr2);
                plt.show();
                plt.stem(arr1,arr3);
                plt.show();
                arr5 = np.multiply(arr2,arr4);
                arr6 = np.multiply(arr5,arr3);
                plt.title('bin 100 test TIME bin 30 basis');
                plt.ylabel('Amplitude');
                plt.xlabel('Time');
                plt.stem(arr1,arr6);
                plt.show();
                return 0;

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

Included in ***Appendix C,*** as Code for *AdditiveSynthesis.py*.

```
import mir
import numpy as np
import wave
import os
import struct
import matplotlib.pyplot as plt


class AdditiveSynthesis():

	def main():

		#improve - read this list from a file. or receive as system argument. (from console.)
		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');

		arr1 =
np.linspace(0,len(outputSignal.data)/440,len(outputSignal.data)/440);#len(sign.data),len(sign.data)); #x's
		print arr1.shape;

		arr2 = outputSignal.data[:len(outputSignal.data)/440]; #y's
		print arr2.shape;

		plt.title('Instrument Waveform Graphed');
		plt.ylabel('Amplitude');
		plt.xlabel('Time');
		plt.stem(arr1,arr2);
		plt.show();

		#improve - take the DFT of said signal of instrument.

		return 0;

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

Appendix D: *Melody.py*

```
import mir
import methodsForFFT
import numpy as np
import wave
import os
import struct
import matplotlib.pyplot as plt

	class Melody():
		def main():
			#theme for mario bros.
			list_of_MIDI_values = [72, 12, 20, 24, 67, 12, 20, 24, 64, 12, 20, 24, 69, 12,
	20, 12, 71, 12, 20, 12, 70, 12, 69, 12, 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];
			list_of_MIDI_values2 = [48, 12, 20, 12, 79, 12, 78, 12, 77, 12, 75, 12, 60, 12,
	76, 12, 53, 12, 68, 12, 69, 12, 72, 12, 60, 12, 69, 12, 72, 12, 74, 12, 48, 12, 20, 12, 79, 12,
	78, 12, 77, 12, 75, 12, 55, 12, 76, 12, 20, 12, 84, 12, 20, 12, 84, 12, 84, 12];
			list_of_MIDI_values =
	np.insert(list_of_MIDI_values,len(list_of_MIDI_values),list_of_MIDI_values2);
			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 = -22000, duration = 0.000000000001);
			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 12th root of 2

				seed = mir.Sinusoid(freq = 0, duration = 0.09);
				mixture = mir.Mixture(seed);
				for k in range(0,len(ratios)):
					sinTemp = mir.Sinusoid(freq = ratios[k]*f, duration = 0.09, amp
	= f_and_amp[k][1]);
					mixture = mir.Mixture(mixture,sinTemp);

				sequence = mir.Sequence(sequence,mixture);

			outputSignal = sequence;
			outputSignal.wav_write('outMelody.wav');

			return 0;
		if __name__ == "__main__": main()
```

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

```
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 E: *q5alt2.py* (Question 5 first part)

```
clear;
clc;
% THIS DOESN'T BUFFER THE DATA!
%For some reason when you buffer
%the data matlab doens’t playback
%the ifftData directly, because it’s not real
filename = 'battleScene.wav';
[inputSamples,Fs] = audioread(filename);
fftData = fft(inputSamples);
ifftData = ifft(fftData);
soundsc(ifftData,Fs);
```

Appendix F: *q5randomphases.py* (Question 5 second part)

```
clc;
clear;
filename = 'battleScene.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));
    theta = rand(length(theta),1);
    %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|')
```

Appendix G: *q5mostlyphases.py* (Question 5 third part)

```
clc;
clear;
filename = 'pianoCsharp.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|')
```

Appendix H: question2.py and question2functions.py

question2.py (Must `import mir,import question2functions,import numpy as np,import wave,import os,import struct,import matplotlib.pyplot as plt`)

```
class question2():
	def main():
		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);
		sizeSinA = len(sine_a.data);
		sizeSinB = len(sine_b.data);
		sizeSinC = len(sine_c.data);
		hanning_window = np.hanning(2048.0);
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()
```

question2functions.py (Must `import mir,import,import numpy as np,import wave,import os,import struct,import matplotlib.pyplot as plt`)

```
def windowSignal(signal,windowingFunction):
		sizeOfSignal = len(signal.data);
		for i in range (0, sizeOfSignal):
			signal.data[i] = signal.data[i]*windowingFunction[np.mod(i, len(windowingFunction))];
def plotMagnitudeSpectrum(signal,sizeOfFFT):
	freq = np.fft.fft(signal.data,sizeOfFFT)/len(signal.data)*1.0;
	real_part = np.real(freq);
	imag_part = np.imag(freq);
	magn_spectrum = np.abs(real_part,imag_part);
	magn_spectrum = 20*np.log(magn_spectrum);
	lin_space = np.linspace(0,sizeOfFFT/2,sizeOfFFT/2);
	plt.ylabel('Amplitude in dB');
	plt.xlabel('Frequency BIN (x Freq resolution = 21.53Hz)')
	plt.title('MagnitudeSpectrum');
	plt.stem(lin_space, magn_spectrum[:sizeOfFFT/2],'r');
def plotWindowedAndNonWindowed(signal,sizeOfFFT,hanningWindow):
	plt.title("Bin NotWindowed");
	plotMagnitudeSpectrum(signal,sizeOfFFT);
	plt.show();
	plt.title("Bin Windowed")
	windowSignal(signal,hanningWindow);
	plotMagnitudeSpectrum(signal,sizeOfFFT);
	plt.show();
```

Appendix I : Implemented Fix to Phase Response

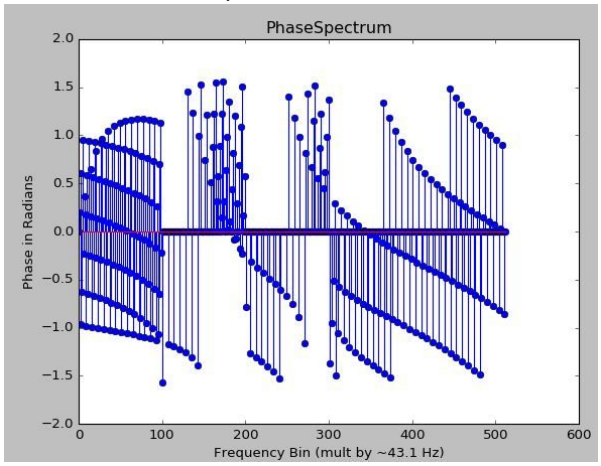
Source:http://www.gaussianwaves.com/2015/11/interpreting-fft-results-obtaining-magnitude-and-phase-information/

This changed inside of my methodsForFFT.py file.

```
def plotPhaseSpectrum(non_Aliasing_Points, sign,real_part,imaginary_part):
        phase_spectrum = np.zeros(non_Aliasing_Points,);
        compl_array = real_part + 1j * imaginary_part; #cheating using complex numbers
        threshold = np.max(np.abs(compl_array))/10000;
        for i in range(0, non_Aliasing_Points):
                if compl_array[i] < threshold:
                        real_part[i] = 0;
                        imaginary_part[i] = 0;
        for i in range (0,non_Aliasing_Points):
                phase_spectrum[i] = np.arctan2(imaginary_part[i],real_part[i]));#*180/np.pi; for degrees
                arr4 = phase_spectrum;
        arr3 = np.linspace(0,non_Aliasing_Points,non_Aliasing_Points);
        plt.title('PhaseSpectrum');
        plt.ylabel('Phase in Radians');
        plt.xlabel('Frequency Bin (mult by ~43.1 Hz)');
        plt.stem(arr3,arr4);
        return 0;
```

The plot looks like this - side by side with previously not ‘denoised’ spectrum. (For Q.1. Exact Fundamental)
There looks like there was some improvement.

Denoised Phase Response acc. To Source



Noisy Phase Response from Before.