M.I.R.

1. (20 pts)

Code for DFT Implementation in Python is included in Appendix A. (*methodsForFFT.py* and *mainProg.py*).

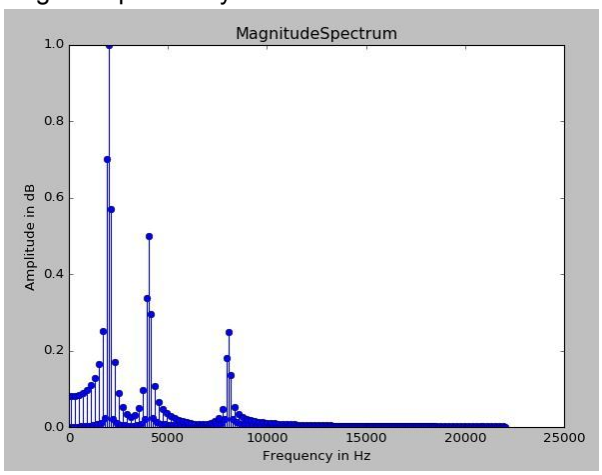
Testing A:

a. Linear combination of 3 harmonically spaced sinusoids (with different amplitudes and phases). + Magnitude and Phase spectrum plots.

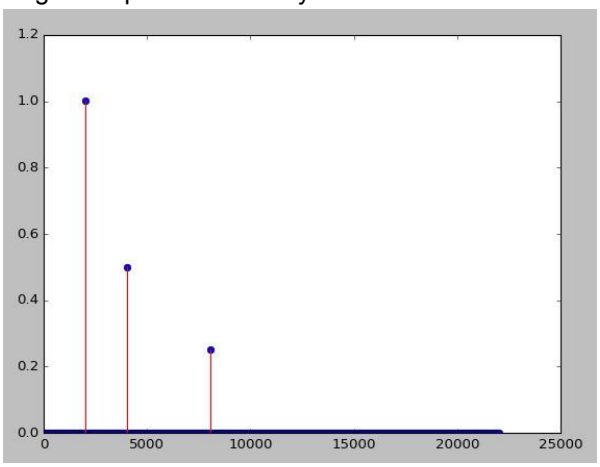
Comparison for the Magnitude Response.

Sine1 has amp = 1,freq = 2000.0,phase = 0;
Sine2 has amp = 0.50,freq = 4000.0, phase = np.pi/4
Sine3 had amp = 0.25,freq = 8000.0, phase = np.pi/2

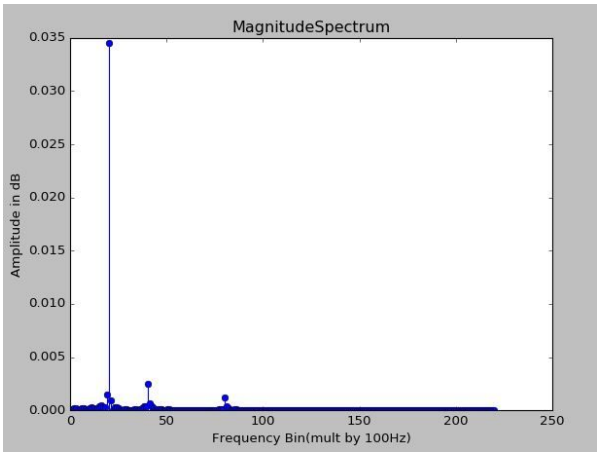
Magn.Resp. DFT by Me



Magn. Resp DFT in Library

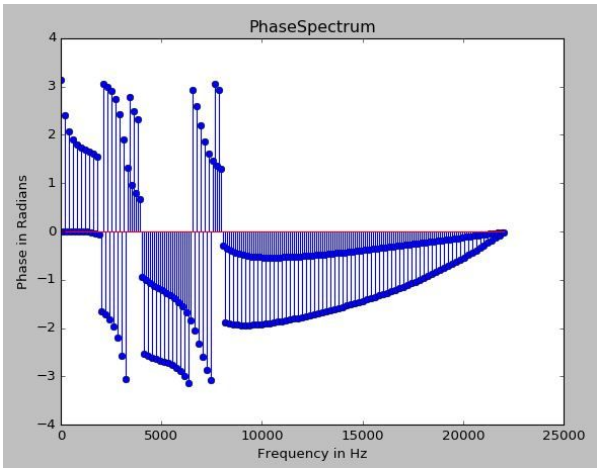
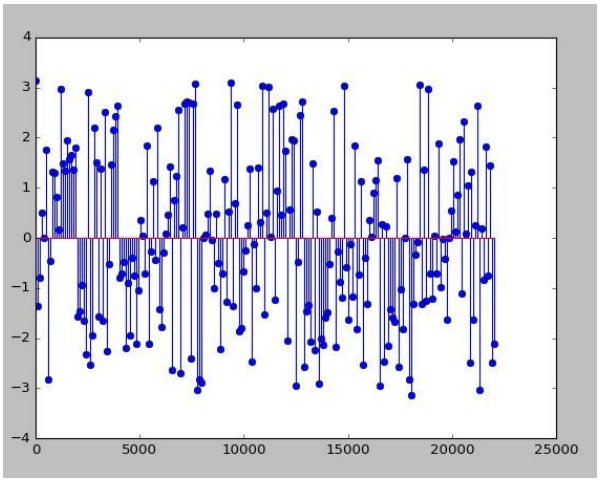


To the right, I show the plot for the same DFT with slightly modified code, that plots against the frequency bins.



Comparison for the Phase Response.

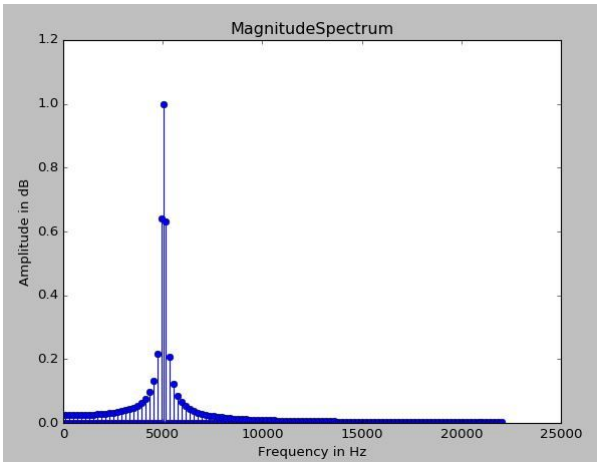
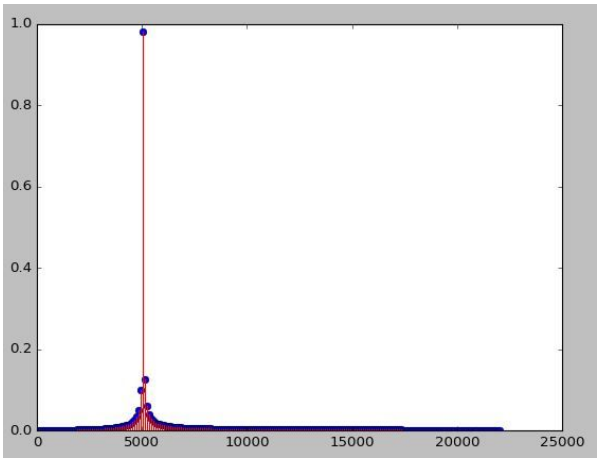
I wasn't able to get a correct phase response from the python implementation.

Phase.Resp. DFT by Me	Phase. Resp DFT in Library
	

b. Testing a frequency that is exactly that of one of the bins. +Magnitude and Phase spectrum plots.

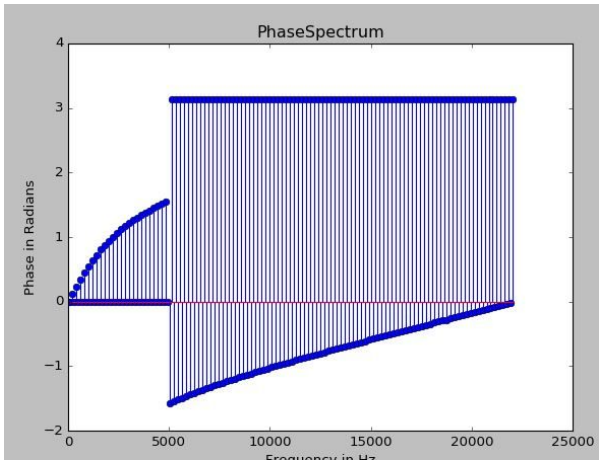
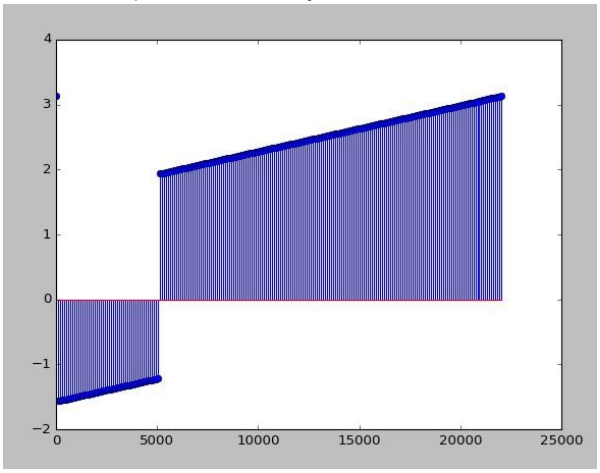
Signal was a Sinusoid at freq = 50.0*100.227272727. 100.227 is bin size in Hz.

Comparison for the Magnitude Response.

Magn.Resp. DFT by Me	Magn. Resp DFT in Library
	

Comparison for the Phase Response.

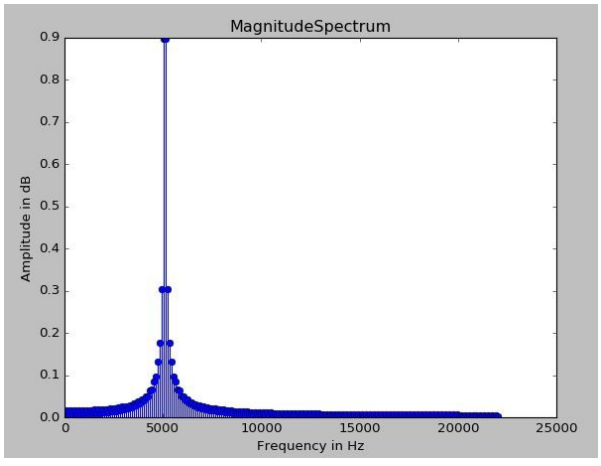
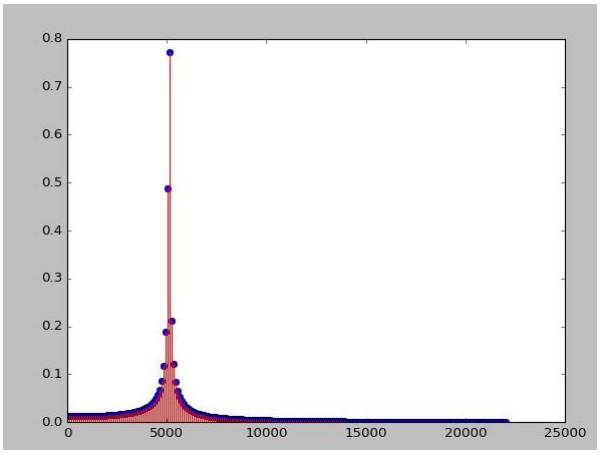
I wasn't able to get a correct phase response from the python implementation. I might be close because the change happened around 5000 Hz. The next phase response (part c) looks closer - why?

Phase.Resp. DFT by Me	Phase. Resp DFT in Library
	

c. Testing a frequency that is between two bins. +Magnitude and Phase spectrum plots.

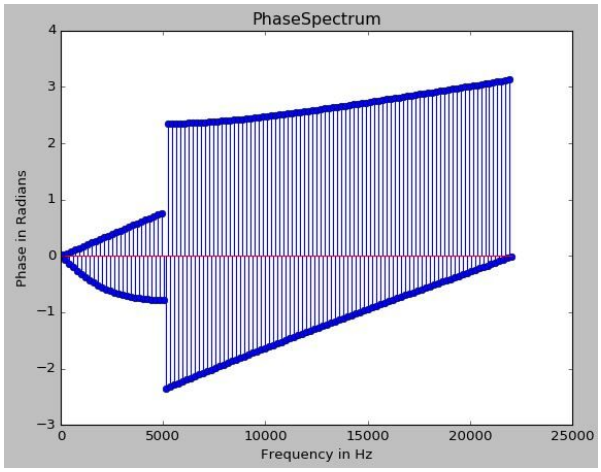
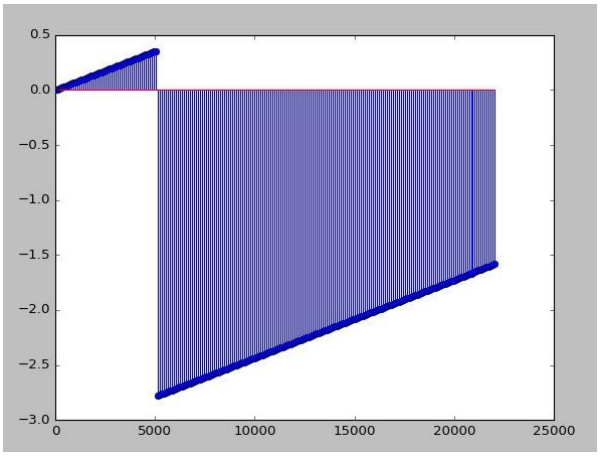
Signal was a Sinusoid at freq = 50.5*100.227272727. 100.227 is bin size in Hz.

Comparison for the Magnitude Response.

Magn.Resp. DFT by Me	Magn. Resp DFT in Library
	

Comparison for the Phase Response.

I wasn't able to get a correct phase response from the python implementation. Ok now I see a resemblance, but I don't understand why that is happening. You can see that the points are there (ascending linearly and then going down and ascending linearly).

Phase.Resp. DFT by Me	Phase. Resp DFT in Library
	

What do you observe?

Discussion :

I notice that in general, windowing the signal helps getting rid of frequency leaks. There are little peaks at the bottom of the greatest peak in the plots. These are reduced by windowing. I have had experience with this, and I know that if you test the FFT with a sinusoid with the frequency of a bin, then you shouldn't get that frequency dispersion at the bottom of the peak. However, my FFT implementation might have some issues because even the signals I tested with the frequency being exactly a bin frequency displayed the frequency leaks.

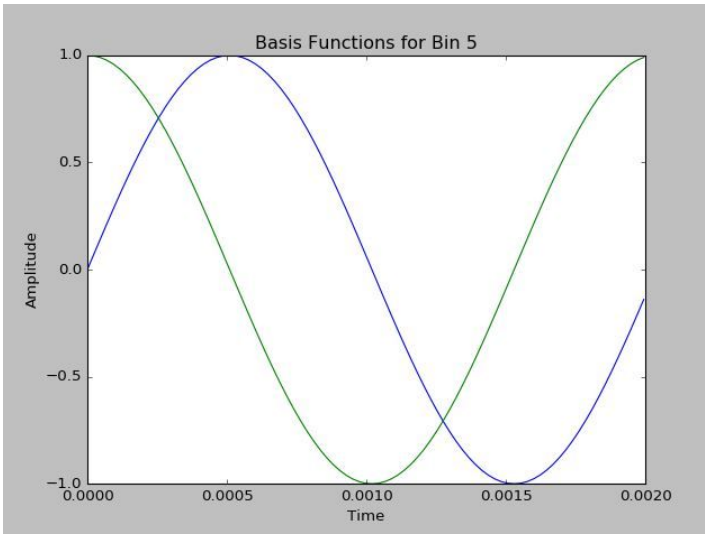
Testing B:

a. Plot basis functions (sine and cosine for a DFT bin). E.g. bin 4 in 256 point DFT.

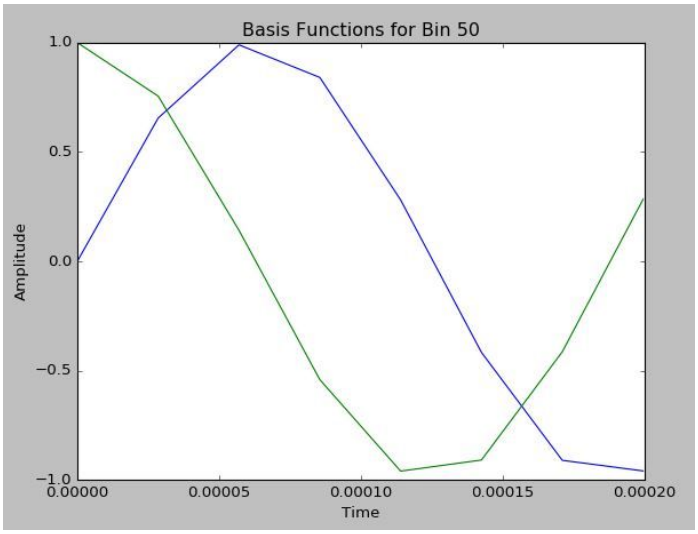
I will plot the basis functions for bin 5. (would be sin(5*100.27) and cos(5*100.27)).

Observation: Code for plot in Appendix B. (*BasisFunctions.py).*

The size of a bin is determined to be 100.27 (I made sizeOfFFT dependant on N. N=41000 for 1 second of audio, and Fs is 44100). I did SizeOfFFT = N/100. So this gives me 441 bins. Each bin is ~ 100 Hz. A cycle should complete in T = 1/freqBin = 1/501.35 = 0.002 seconds.



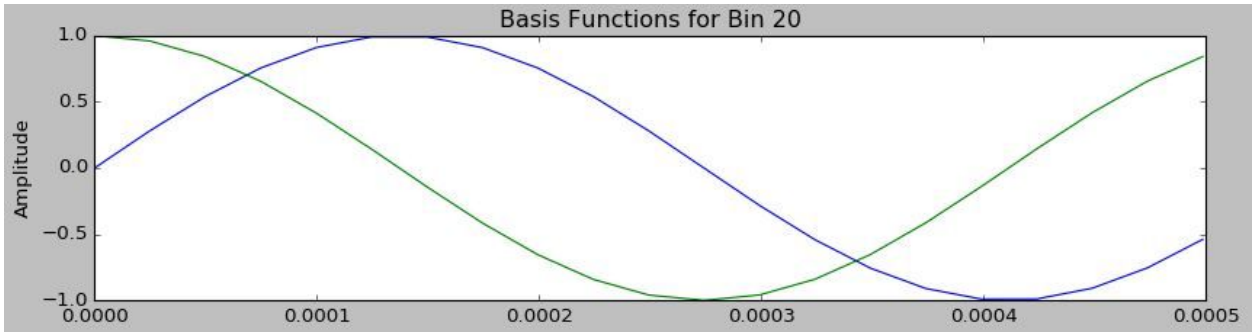
I also plotted the one for bin 50, I noticed that because it is a higher frequency, the number samples is smaller, so it looks ‘jiggy’ (not smooth). (A cycle completes in 0.0002 seconds)



Testing C:
a. Plot the result of pointwise multiplying your input signal from the previous sub question with the three harmonics with the two basis functions that correspond to the closest DFT bin.

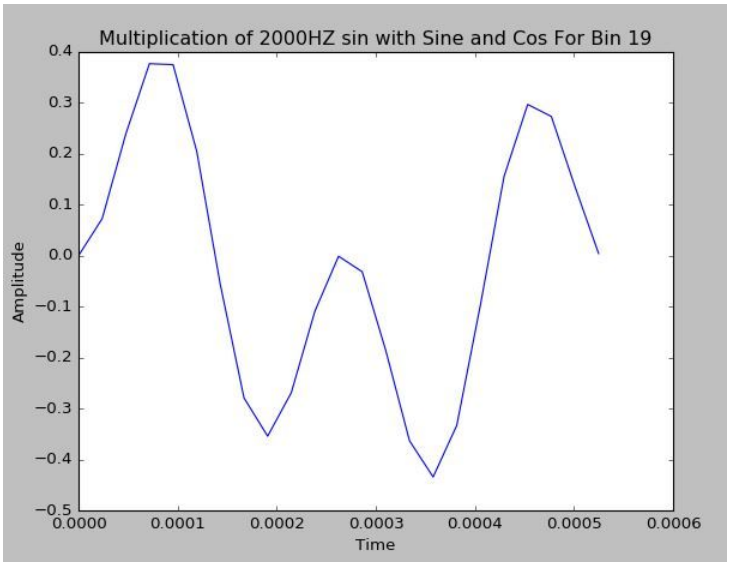
The frequency for the previous sub question fundamental was 2000 Hz. This, divided by the bin Size = 19.94.

So the frequency for that signal (the fundamental) had frequency 2000 Hz. The closest bins to that one are 19 and 20*100,27Hz. I did bin 21 as well 21*100,27 Hz (size of bin is 100,27).

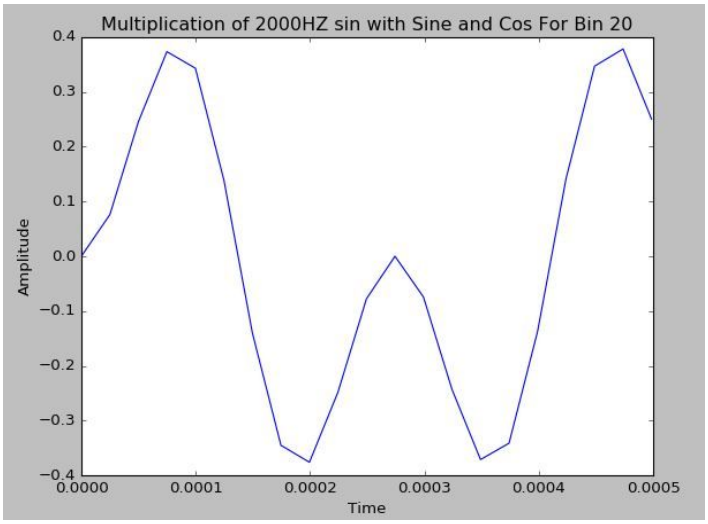


I multiplied pointwise the bin 19 basis functions and the test 2000Hz sinusoid. And repeated for the bin 20 basis functions, and the bin 21 basis functions. These are the results. I display a single ‘cycle’ for the basis functions.

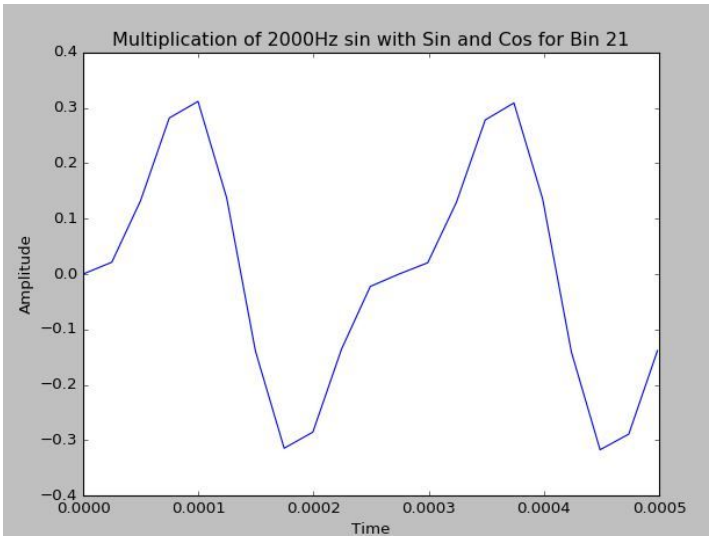
Bin 19 basis functions times test sin



Bin 20 basis functions times test sin



Bin 21 basis functions times test sin

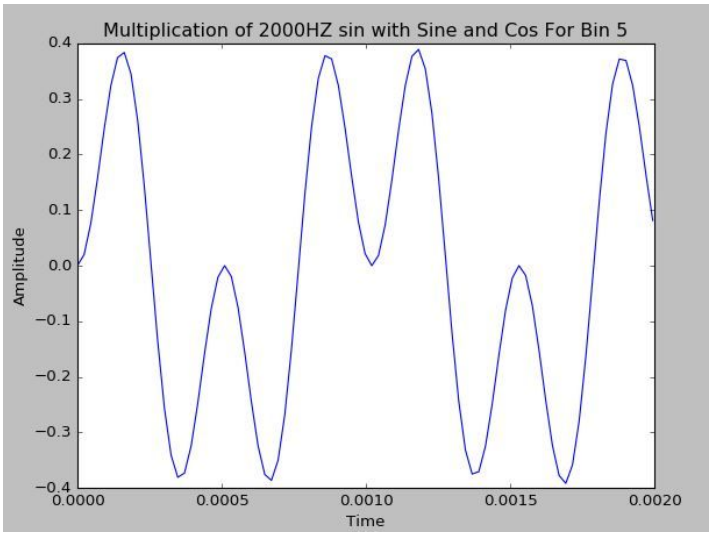


Edit :- inner product is the addition of pointwise multiplications. For a related bin the addition of all the pointwise multiplications should be non-zero. And for an unrelated bin the addition should be close to zero.

In these plots, we see the results would be non-zero if added up all the pointwise multiplications.

b. Also plot the result of pointwise multiplying your input signal with two basis functions that correspond to an unrelated DFT bin. What do you observe ? How do these plots connect with your understanding of the magnitude and phase for a DFT bin?

I chose unrelated bin 5. Frequency is 5*100,27 Hz (size of bin is 100,27). We have seen the basis functions in previous parts of this assignment.



Discussion:
Edit: Pointwise multiplication gives this signal. And inner product is the addition of all the points in it. The result would be zero or close to zero.

It was mentioned in class that multiplying sinusoids of same frequencies (even with different amplitudes and phases) yields a sinusoid of the same frequency. This is important here - if the result of pointwise multiplication has the same number of cycles as the input, that means that the input is close to the basis functions. In our results, we see that for the closest bins (bin 19 and 20) to the input, we get as output of the multiplication, a sinusoid of the same frequency (waveform is changed, but still the cycle has the same duration).

This would indicate higher correlation between input and the bin. If we were to multiply pointwise a single one of the basis functions for the input, there would likely be constructive interference.
However, look at less related bins - bin 21 (even though it's close) generated two cycles instead of just one. This will mean less correlation with the input. Lastly, the less related bin (bin 5) shows also more cycles instead of just one. Therefore if we try to correlate one of the basis functions for these bins with the input, we will get destructive interference (cancellation), and a smaller result than with the more related bins.

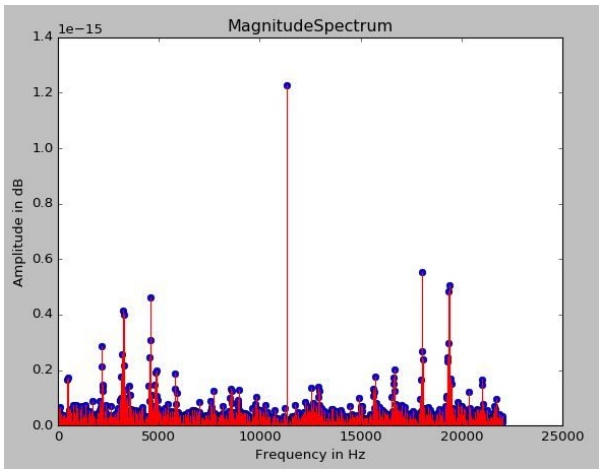
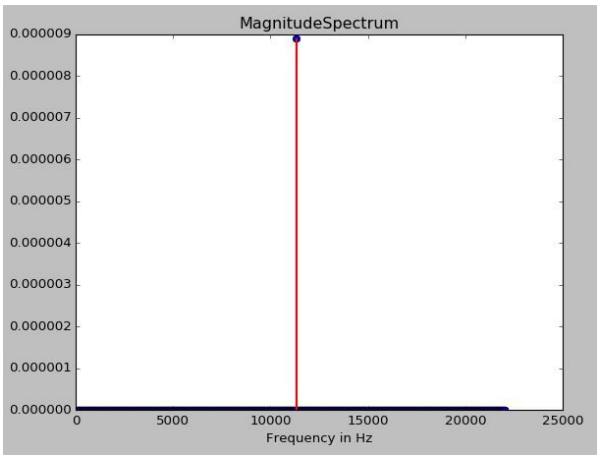
EDIT: Today in class it was discussed that when multiplying the basis functions with an unrelated bin, the inner product is going to be very close to zero.

2. (20 pts).

First Part:

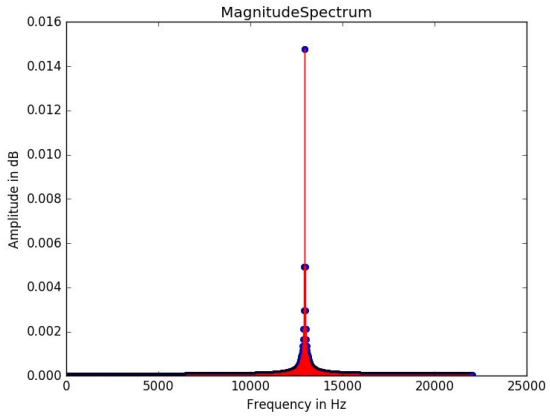
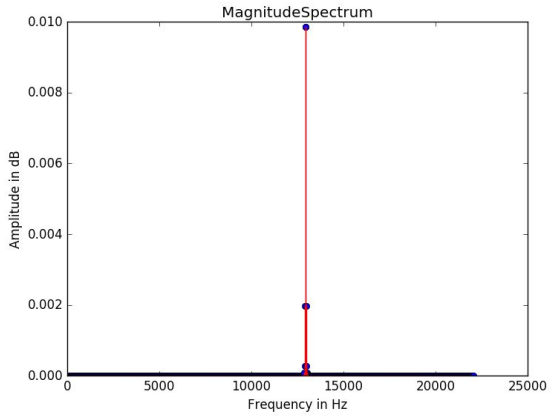
a. Using an existing FFT implementation, plot: Magnitude response in dB. Use DFT size 2048 of a sine wave that has frequency corresponding to the bin 525.

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

Non windowed	Windowed
	

b. Magnitude response in dB of a sine wave with freq corr. to bin 600.5.

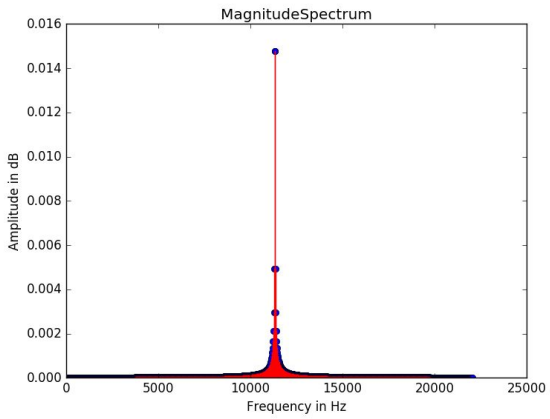
Size of bin is roughly 21.53 Hz so the 525th bin would be frequency 12,928 Hz approx.

Non windowed	Windowed
	

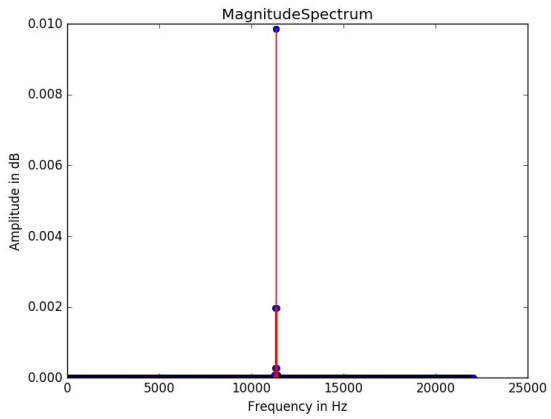
c. Magn. resp. In dB of a hanning - windowed (size of window 2048) DFT of a sinusoid w/ frequency corr. to bin 525.5.

Size of bin is roughly 21.53 Hz so the 525th bin would be frequency 11,314 Hz approx.

Non windowed



Windowed



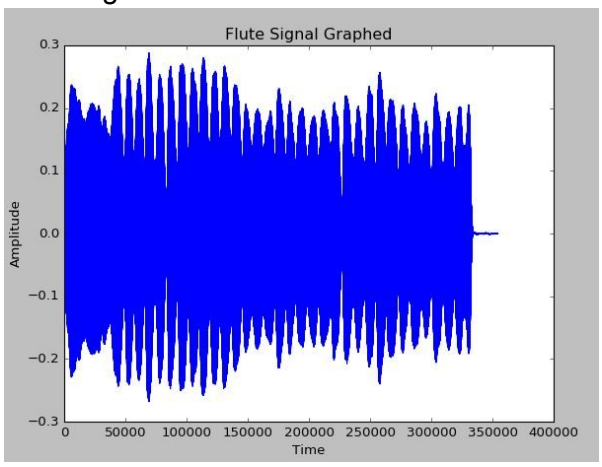
Second Part:

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.

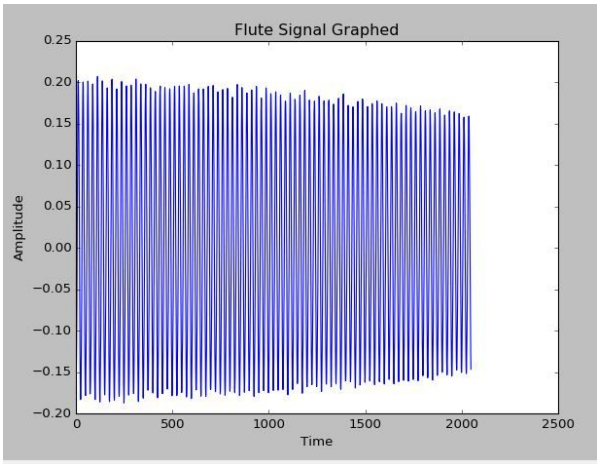
Plot Magnitude response:

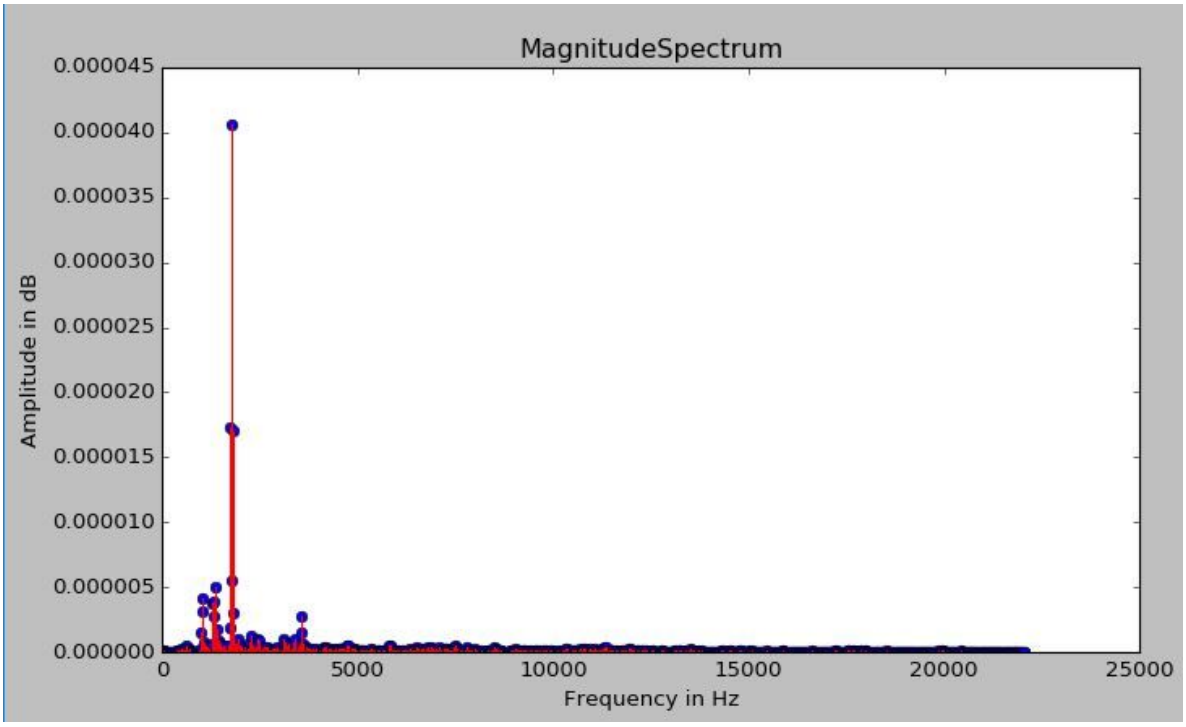
The waveform for this flute note is this. Posteriorly, 2048 samples were selected. The FFT showed a peak at 1764 Hz.

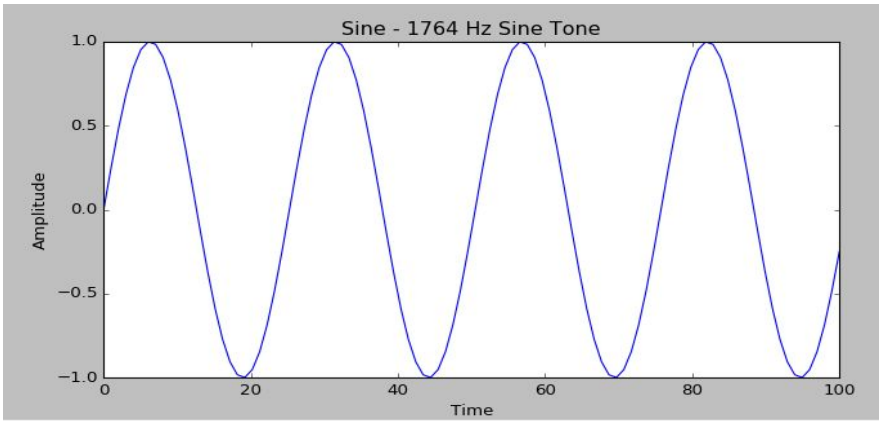
Initial Signal



Selected samples




MagnitudeSpectrum
Amplitude in dB
0.000045
0.000040
0.000035
0.000030
0.000025
0.000020
0.000015
0.000010
0.000005
0.000000
0
5000
10000
15000
20000
25000
Frequency in Hz


Sine - 1764 Hz Sine Tone
Amplitude
1.0
0.5
0.0
−0.5
−1.0
0
20
40
60
80
100
Time

3. (20 points)
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.

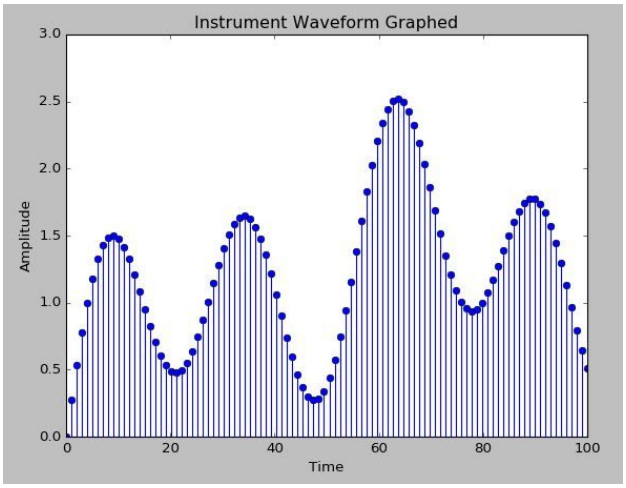
Lastly, to recognize the chords, the incoming signal is compared with each of these chord templates. I am not so clear about how many profiles to make, because there are many different kinds of chords (thinking about 7th chords, which are almost like the regular chords, but with one extra note).

(CONTINUE READING -> Question 4 below).

Question #4.

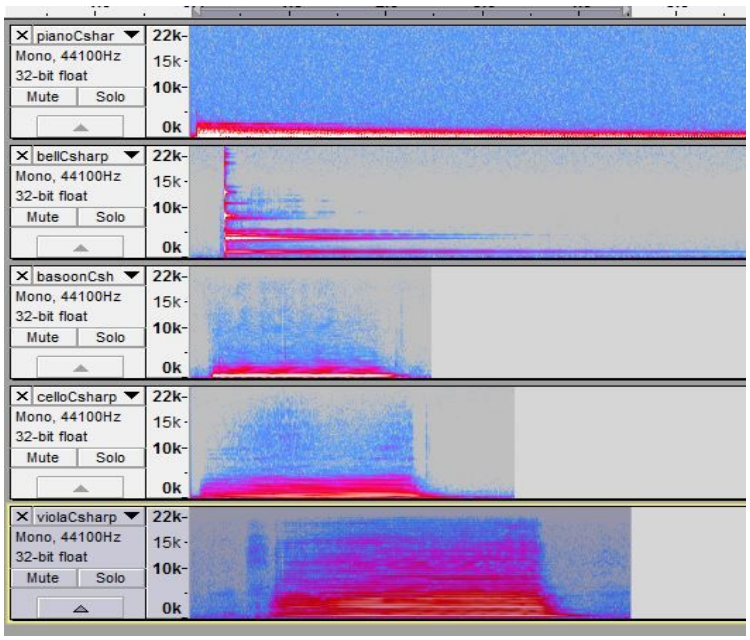
A.Code for additive synthesis instrument (5 points).

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.

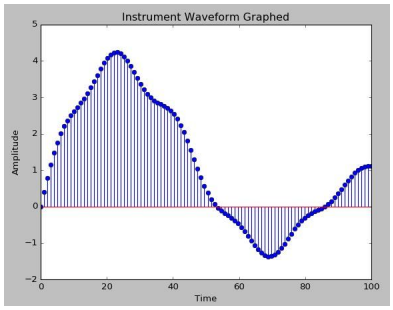


I found that the spectrums for the piano, bassoon, viola and cello look similar. The one for the bells looks like it has more harmonics. Easy to distinguish between the bells and everything else, but not between the others.

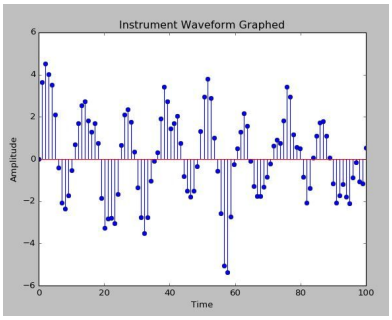
What would be missing probably is the variation in energy over time for the different frequencies that make up the sound. Attack,delay,sustain,release would make the sound more realistic.

Plots for resulting waveforms and list of frequencies and amplitudes:

Piano

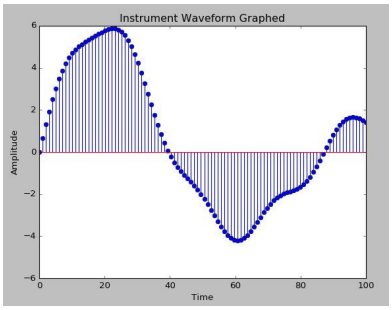


Bell

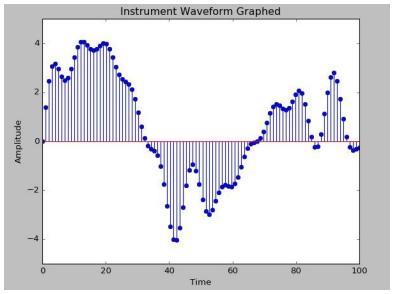


Basoon

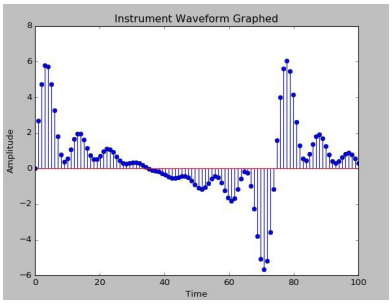
Looks very similar to Piano.



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:

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:

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

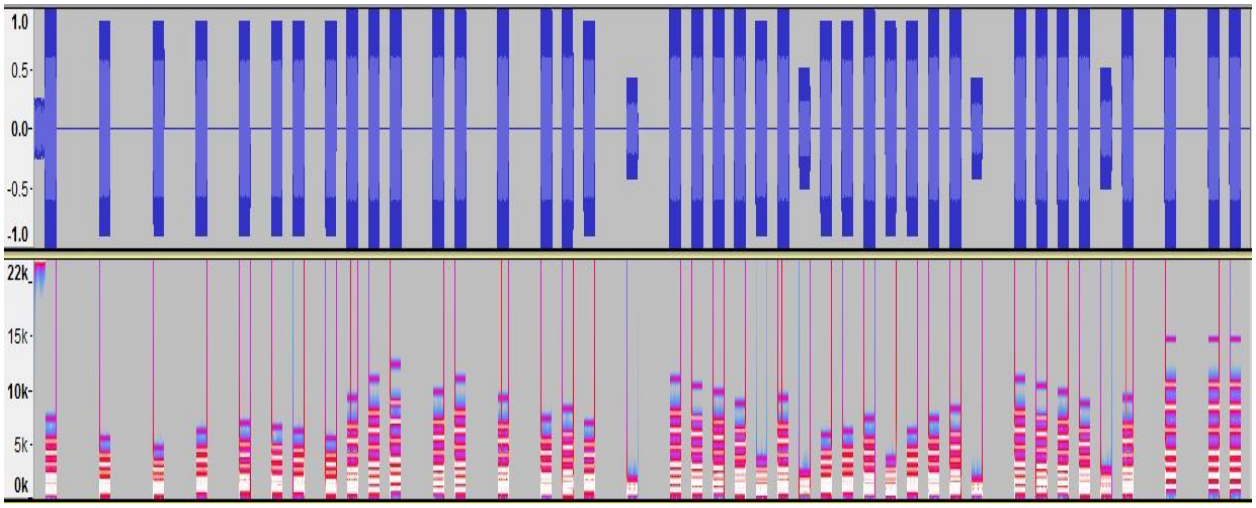
This looks very similar to the formula for calculating a frequency in 12 T.E.T.: (It is the same in fact, because 'a' in this formula refers to the reference frequency!)

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

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 A : Code for DFT Implementation in Python

I created a new file which held all my methods for FFT: (*methodsForFFT.py*)

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


def dftByHector(factor,numberOfBins,sign,inputSamples):

	re_in = inputSamples;
	im_in = np.zeros(len(inputSamples),);

	real_part = np.zeros (numberOfBins,);
	imaginary_part = np.zeros (numberOfBins,);

	sizeOfBinInHertz = (44100/2)/float(numberOfBins);
	print 'sizeOfBinInHertz'
	print sizeOfBinInHertz;

	for i in range (0,numberOfBins):

		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);

		#found reference at
https://www.nayuki.io/page/how-to-implement-the-discrete-fourier-transform

		for j in range (0,len(inputSamples)):
			real_part[i] += (re_in[j] * temp_cosine.data[j]) + (im_in[j]
* temp_sine.data[j]); #calculates the sum per bin k
			imaginary_part[i] += (-1 * re_in[j] * temp_sine.data[j]) +
(im_in[j] * temp_cosine.data[j]);

	plotDFT(factor,numberOfBins, sign,real_part,imaginary_part);

	#--------------
	return 0;
```

```
def plotMagnitudeSpectrum(factor,numberOfBins, sign,real_part,imaginary_part):

	magnitude_spectrum = np.zeros (numberOfBins,);


	for i in range(0,numberOfBins):
		magnitude_spectrum[i] =
np.sqrt(imaginary_part[i]**2+real_part[i]**2);
		magnitude_spectrum[i] = magnitude_spectrum[i]/(numberOfBins/2);

	arr2 = magnitude_spectrum;

	arr1 = np.linspace(0,(44100)/2,numberOfBins);

	plt.ylabel('Amplitude in dB');
	plt.xlabel('Frequency in Hz')
	plt.title('MagnitudeSpectrum');
	plt.stem(arr1,arr2);
	return 0;

def plotPhaseSpectrum(factor,numberOfBins, sign,real_part,imaginary_part):
	phase_spectrum = np.zeros (numberOfBins,);


	for i in range (0,numberOfBins):
		phase_spectrum[i] = np.arctan2(imaginary_part[i],real_part[i]);

	arr4 = phase_spectrum;
	arr3 = np.linspace(0,(44100)/2,numberOfBins);

	plt.title('PhaseSpectrum');
	plt.ylabel('Phase in Radians');
	plt.xlabel('Frequency in Hz');
	plt.stem(arr3,arr4);
	return 0;


def plotDFT(factor,numberOfBins,sign,real_part, imaginary_part):
	plotMagnitudeSpectrum(factor,numberOfBins,sign,real_part,
imaginary_part);
	plotPhaseSpectrum(factor,numberOfBins,sign,real_part,imaginary_part);
	plt.show();
	return 0;
```

This is the main program: (*mainProg1.py*) (just calls on methodsForFFT and mir).

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


class mainProg1():

        def main():

                #---------plotting with self implemented DFT----------------------------------#
                #------------------------------------------------------------------------------#
                #------------------------------------------------------------------------------#

                #PART A - test with harmonically related sinusoids.
                #sine1 = mir.Sinusoid(amp = 1,freq = 2000.0,phase = 0);
                #sine2 = mir.Sinusoid(amp = 0.50,freq = 4000.0, phase = np.pi/4);
                #sine3 = mir.Sinusoid(amp = 0.25,freq = 8000.0, phase = np.pi/2);
                #signal = mir.Mixture(sine1,sine2,sine3);
                #PART A - test with harmonically related sinusoids.

                #PART B - test with sinusoid exactly at a Bin.
                #signal = mir.Sinusoid(freq = 50.0*100.227272727); #100.227 is the calculates size
                                                                   # of a single bin in Hz
                #PART B - test with sinusoid exactly at a Bin.

                #PART C - test with sinusoid between Bins.
                signal = mir.Sinusoid(freq = 50.5*100.227272727); #100.227 is the calculates size
                                                                   # of a single bin in Hz
                #PART C - test with sinusoid between Bins.

                factor = 100; #this reduces the size of the DFT to a fraction of the length of
                                        #the input
                inputSamples = signal.data;

                sizeOfFFT = len(inputSamples)/factor; #size of fft ( k = N , in this case)
                numberOfBins = sizeOfFFT/2;

                #PART A - test with harmonically related sinusoids.
                #methodsForFFT.dftByHector(factor,numberOfBins,signal,inputSamples);
                #PART A - test with harmonically related sinusoids.

                #PART B - test with sinusoid exactly at a Bin.
                #methodsForFFT.dftByHector(factor,numberOfBins,signal,inputSamples);
                #PART B - test with sinusoid exactly at a Bin.

                #PART C - test with sinusoid between Bins.
                #methodsForFFT.dftByHector(factor,numberOfBins,signal,inputSamples);
                #PART C - test with sinusoid between Bins.
```

```
        #----------------'plotting with library function for DFT'----------------------#
        #-------------------------------------------------------------------------------#
        #-------------------------------------------------------------------------------#

        freq = np.fft.fft(inputSamples,sizeOfFFT)/numberOfBins;#/len(inputSamples);
        real_part = np.real(freq);
        imag_part = np.imag(freq);

        magn_spectrum = np.absolute(freq);


        phase_spectrum = np.zeros(numberOfBins,);
        for i in range (0,numberOfBins):
                phase_spectrum[i] = np.arctan2(imag_part[i],real_part[i]);

        lin_space = np.linspace(0,(44100)/2,numberOfBins);

        #plt.stem(lin_space, magn_spectrum[:numberOfBins],'r');
        plt.stem(lin_space, phase_spectrum[:numberOfBins],'b');

        plt.show();

        return 0;


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

Appendix B : *BasisFunctions.py* Used to plot the basis sin and cos for a specific bin. (Using the bin size found for previous DFT). A modification of this was used for the question on multiplying by related and unrelated bins.

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

class BasisFunctions():
        def main():
                SizeOfBin = 100.27;
                NumBin = 50;
                BasisFreq  = float(NumBin*SizeOfBin); #~500
                sine1 = mir.Sinusoid(freq = BasisFreq);#sign);
                cosine1 = mir.Sinusoid(freq = BasisFreq, phase = np.pi/2);#sign);
                lengthOfCycle = float(1/BasisFreq); #seconds 0.002
                Fs = 44100.0;
                t = BasisFreq/Fs;
                numSamplesPerCycle = (Fs*lengthOfCycle);
 #since we sample 44100 times per 1 second, we sample x times for 0.002 seconds
 #44100/1 = x/0.002 =>  x = (44100/1)(0.002) = 44100 * lengthOfCycle ~about 88 samples.;

                arr1 = np.linspace(0.0,lengthOfCycle,numSamplesPerCycle);    #start,stop,n
                print 'size'
                print arr1.shape;
                arr2 = sine1.data[:numSamplesPerCycle];
                print 'size2'
                print arr2.shape;
                arr3 = cosine1.data[:numSamplesPerCycle];
                print 'size3'
                print arr3.shape;

                plt.title('Basis Functions for Bin 50');
                plt.ylabel('Amplitude');
                plt.xlabel('Time');
                plt.plot(arr1,arr2);
                plt.plot(arr1,arr3);
                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|')
```