

## CSC – 575 (Assignment -1)

Question1:

Question2:

Question3:

For purpose of answering the question, I have selected the GTA San Andreas (Theme Song) - <https://www.youtube.com/watch?v=W4VTq0sa9yg>

Part -1

Time Scales – Short, Middle and Long Term

Dimensions – Timber, Orchestration, Acoustics, Rhythm, Melody, Harmony and Structure.

The opening of the music starts with Chimes followed by repetitive drum beats and piano.

**Rhythm** – The music has a time scale – Middle Term, where a sound is being repeating multiple times. The entire Sequence is coming back again.

**Melody** – The music after the initial opening has a middle term melody repetition, that is high and low drum beats.

**Timber** – There is a lot of classification in Timber (Short term) when the opening comes from the Chimes.

**Structure** – When seen for the long term the music has a lot of time scale differences.

Part – 2

**Casual Users** – Since casual users like listening and collecting music their request will be like (Finding the song that sounds like this) (I like this one, what other songs I may admire)(I have to organize my collection)

**Professional Users** – They may be using music in advertisement and production, they need music for an application, their request information will be like (Given a rhythm give me the music that matches it) (I need this kind of soundtrack with this instrument)

**Music Scholars** – These kinds of people are interested in studying music, developing music (analysis of music tools)(I don't want the music but the pitch it was sung in)

Part – 3

**Query By humming** – Humming is a way where in a casual user will use a combination of whistling and singing to produce a melody. Then that melody will be used to search/query the sound/music database to find the full version of the song/music corresponding to the melody.

**Query by Example** – When a part of the music/example of the music is used to query the database to retrieve the full music it is termed as query by example. This will avoid mismatching of the retrieved music.

#### Part- 4

**Music XML**- This is a type of music representation which used XML as portable format for exchange of information. The best part is the industry wide support including the open-source and commercial projects. Since we can write XML in Latex so we can represent this using the latex.

**MIDI**- it stands for Musical Instrument Digital Interface. This is format where we do the data exchange within digital instruments. This requires sequential information transfers. SMF- Standard Midi File will be used for the future information transfer.

#### Part-5

Since this entire document deals with MIR and we are doing MIR for the end users, for me that's the best part of explaining various users that uses MIR. There are three levels of users starting from the most basic going to the most complex ones. The casual users are the one who need music for fun and hobbies. The second level is for the professional student that may need music for their professional purposes, they are commercial users of music. The music scholars are most advanced users; they are one who study music and its composition. It also includes musicians.

#### Part-6

**Paper – 1: Indexing Music Databases Using Automatic Extraction of Frequent Phrases**

In the field of MIR the extraction of music for the database is a crucial and complex task, if we wish to have the offline database then storing the entire indexing file structure take a lot of memory, The best possible way to improve this is using the phrases instead of full content. Repetition is an important property of music, if we utilize that property and only store the most occurring phrase instead of entire file we can save a lot of space. We already have efficient text mining methods that can be applied with minor modifications.

The paper is based of Irish folk music, and concludes with a text-mining based indexing algorithm for music which extracts repeating patterns from the music.

**Paper -2: A Multimodal Framework for Music Inputs**

Music waves have one excellent property of being multimodal, since the previous research in this field have done a lot of work with the indexing and fetching of data but the technique outlined in this paper is essentially for the query. Since when users interact with computer

they want the ease of use, that means that it should be possible for them to query the music database in any form they wish to, example – sing, play or sent symbols.

Since this idea is a new step in terms of the Human Computer Interaction (HCI), which is one of the most emerging topic and further it gives the user a flexibility to interact with the music system.

### Paper - 3: Query By Humming -- Musical Information Retrieval in an Audio Database

If we have all the text in the database, we need to query it using a text, but what if we have a database full of music and audio files, it would be great if we could just hum the melody and fetch the music of our choice. Efficient pattern matching is an important aspect of this paper. It may be possible not to return just one but multiple matching music files.

It is equally essential to have an optimized version of query in the system. Since taking extra time will result in the user losing interest in the system. The authors use pith as the medium of detection between the hummed melody and original sound wave in the database.

### Question4:

#### Part - 1

Additive Synthesis Instrument: The actual concept is to have n number of sin wave generator using the defined frequency and helper function – Mixture ()

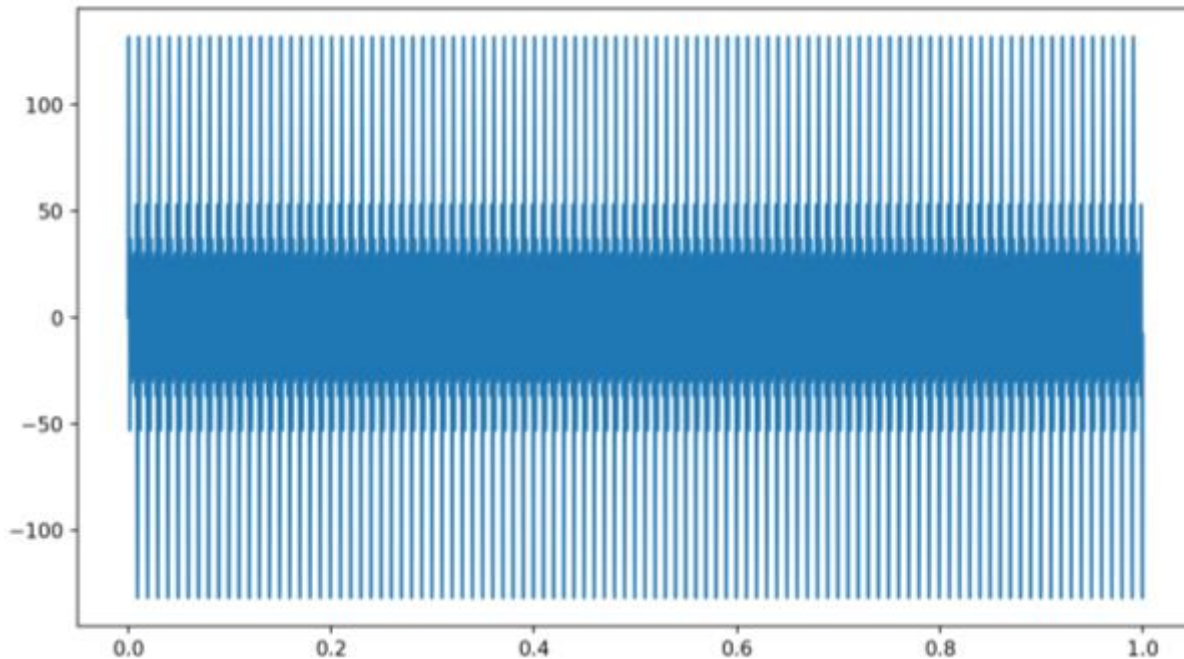
#### Code:

```
import numpy as np
import matplotlib.pyplot as plt
import math
import mir
from mir import Sinusoid
from mir import Signal
from mir import Mixture

#Declare the frequency and amplitude array
freq = [100,200,300,400,500]
amp = [10,20,30,40,50]
#Declare the empty list
list = []
#Making a list of all 5 sinusoids
for n in range (5):
    sin = Sinusoid(amp=amp[n], freq=freq[n])
    list.append(sin)
#Making a mixture using the helper function in mir.py
mix = Mixture(*list)
#Plot the mixture of all 5 waves
mix.plot()
```

#### Plot:

Jagjit Singh (V00865544)



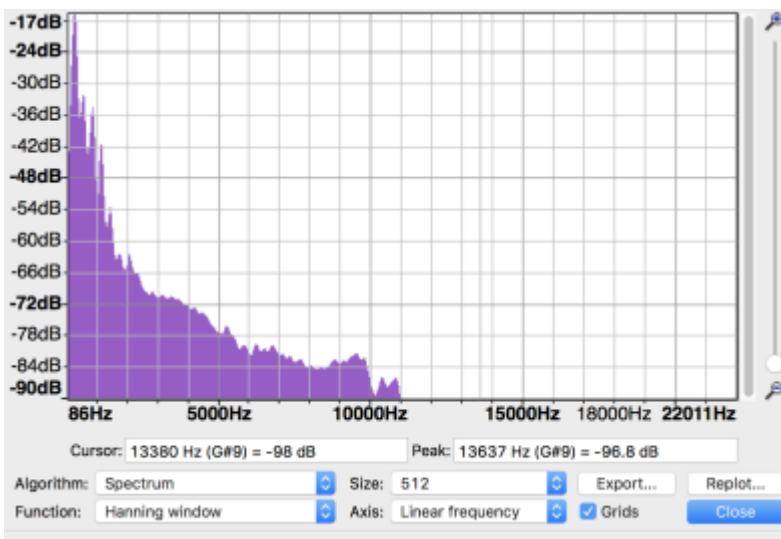
Part - 2

Different Sounds of Same Pitch – passing them to additive Instrument.

Sound 1 = Flute D Tone

Sound 2 = Violin D Tone

Spectrum Analysis Sound 1



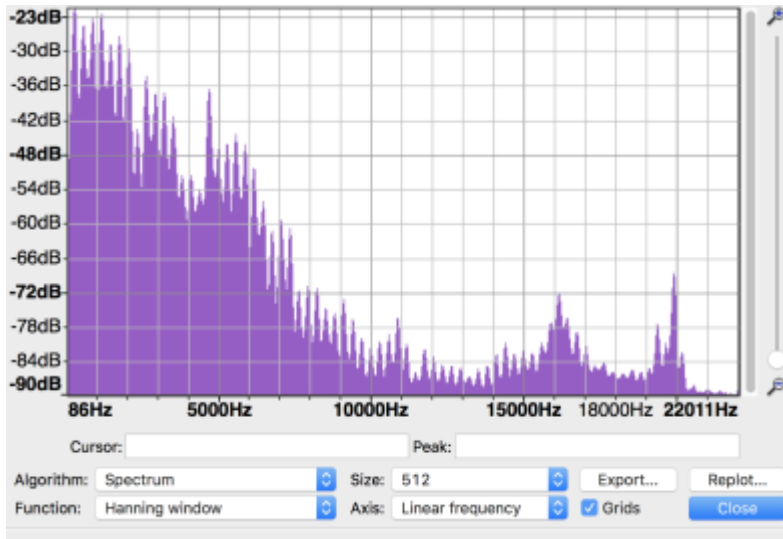
freq = [292,563,869,1160,1439,1767,2064,2828,3159,3447]

amp =

[0.15488166189124816,0.03162277660168379,0.019275249131909367,0.008511380382023767,0.0023173946499684774,0.0007328245331389037,0.0007673614893618193,0.0003388441561392024,0.00031988951096913973,0.00030549211132155157]

Jagjit Singh (V00865544)

## Spectrum Analysis Sound 2



freq = [295,572,882,1165,1460,1764,2052,2343,2633,2931]

amp =

[0.0812830516164099,0.06165950018614822,0.06095368972401691,0.07852356346100718,0.03890451449942807,0.042169650342858224,0.033884415613920256,0.006760829753919818,0.022130947096056376,0.01428893958511103]

### Code:

```
import numpy as np
import matplotlib.pyplot as plt
import math
import mir
from mir import Sinusoid
from mir import Signal
from mir import Mixture
```

*#Frequency and Amplitude for Sound 1*

freq = [292,563,869,1160,1439,1767,2064,2828,3159,3447]

amp =

[0.15488166189124816,0.03162277660168379,0.019275249131909367,0.008511380382023767,0.0023173946499684774,0.0007328245331389037,0.0007673614893618193,0.0003388441561392024,0.00031988951096913973,0.00030549211132155157]

list = []

for n in range (10):

sin = Sinusoid(amp=amp[n], freq=freq[n])

list.append(sin)

*#make signal mixture*

mix1 = Mixture(\*list)

sig1 = Signal(data=mix1.data)

*#output new audio file*

sig1.wav\_write('flu\_new.wav',normalize=False)

mix1.plot()

*#Frequency and Amplitude for Sound 2*

freq = [295,572,882,1165,1460,1764,2052,2343,2633,2931]

amp =

[0.0812830516164099,0.06165950018614822,0.06095368972401691,0.07852356346100718,0.03890451449942807,0.042169650342858224]

Jagjit Singh (V00865544)

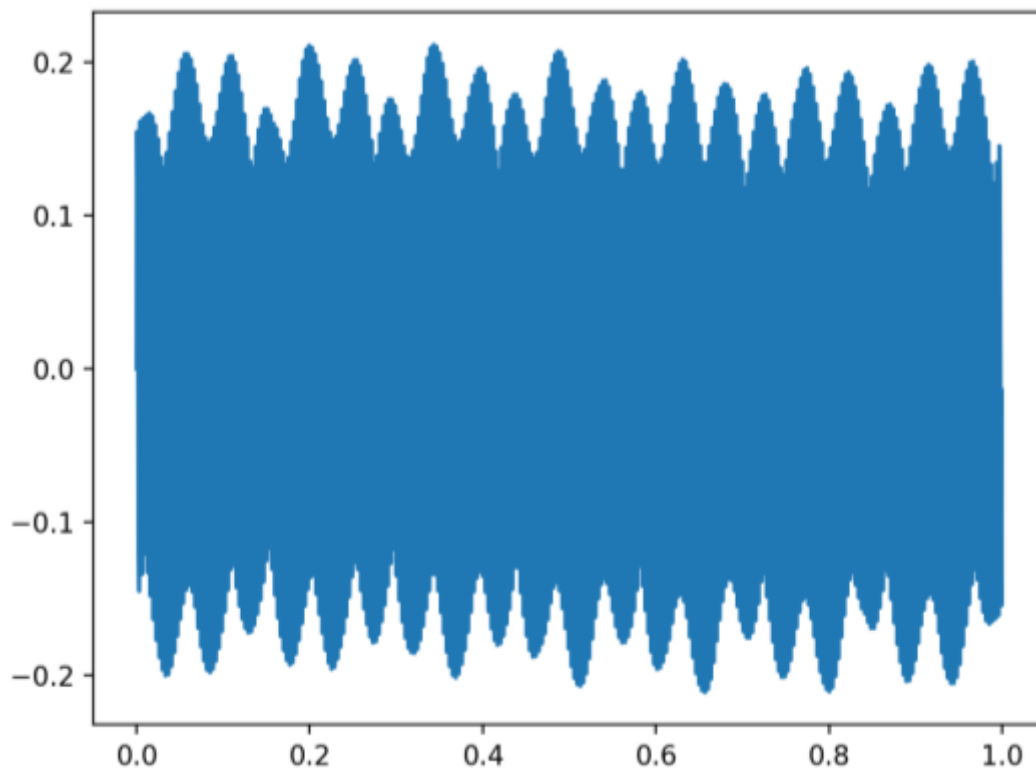
,0.033884415613920256,0.006760829753919818,0.022130947096056376,0.01428893958511103]

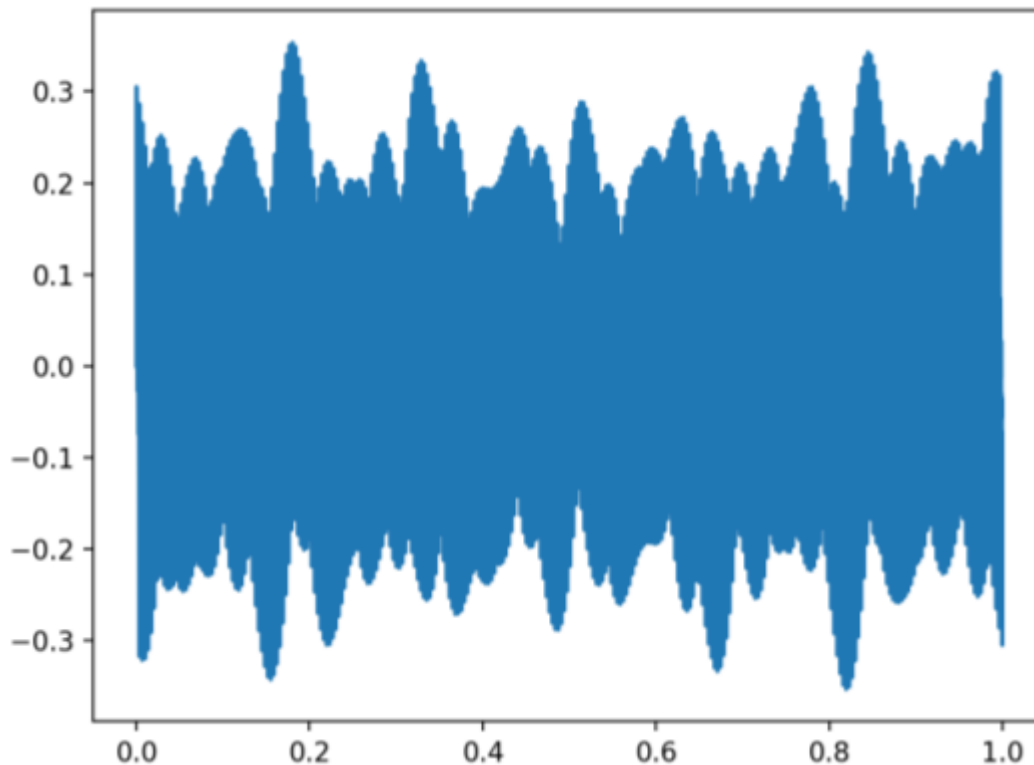
```
list1 = []
for n in range (10):
    sin = Sinusoid(amp=amp[n], freq=freq[n])
    list1.append(sin)
#make signal mixture
mix2 = Mixture(*list1)

sig2 = Signal(data=mix2.data)
#output new audio file
sig2.wav_write('voi_new.wav',normalize=False)

mix2.plot()
```

Plot :





### Part 3

#### Question5:

##### Part – 1

Use the helping function provided in the file to read and write the audio file.

##### Code:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack
import math
import mir
from mir import Sinusoid
from mir import Signal

sig = Signal()
#Read audio File
sig.wav_read('Ques5Input.wav')
wave = sig.data
wavefft = np.fft.fft(wave)
waveifft = np.fft.ifft(wavefft)
#Writing the audio file
sig = Signal(data=waveifft)
sig.wav_write('Ques5Output.wav')
```

Result: Both the audio files are identical.

Jagjit Singh (V00865544)

Input Audio –

<https://github.com/JagjitUvic/MIR/blob/master/Assignment%201/Ques5Input.wav>

Output Audio -

<https://github.com/JagjitUvic/MIR/blob/master/Assignment%201/Ques5Output.wav>

Part – 2

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack
import math
import mir
from mir import Sinusoid
from mir import Signal

sig = Signal()
#Read audio File
sig.wav_read('Ques5Input.wav')
wave = sig.data
wavefft = np.fft.fft(wave)
length = len(wavefft)
magnitude = []
phase = []
for n in range (length):
    magnitude.append(np.absolute(wavefft[n]))
    phase.append(np.angle(wavefft[n]))
#phase goes random
phase = np.random.randn(length)
sigC = []
for n in range (length):
    val1 = magnitude[n]*math.cos(phase[n])
    val2 = magnitude[n]*math.sin(phase[n])
    sigC.append(np.complex(val1,val2))
#find inverse transform
waveifft = np.fft.ifft(sigC)
#write back the signal object
sig = Signal(data=waveifft)
sig.wav_write('Ques5_2Output.wav')
```

Part – 3

Code:

```
import numpy as np
import matplotlib.pyplot as plt
import scipy.fftpack
import math
import mir
from mir import Sinusoid
from mir import Signal

sig = Signal()
#Read audio File
sig.wav_read('Ques5Input.wav')
wave = sig.data
wavefft = np.fft.fft(wave)
```



Jagjit Singh (V00865544)

```
length = len(wavefft)
magnitude = []
phase = []
for n in range (length):
    magnitude.append(np.absolute(wavefft[n]))
    phase.append(np.angle(wavefft[n]))
#fetching 4 high values of magnitude
index = []
value = []
for n in range (4):
    index.append(np.argmax(magnitude))
    value.append(np.amax(magnitude))
    magnitude[np.argmax(magnitude)] = 0
#setting 4 high values rest all 0
for n in range (length):
    if n == index[0]:
        magnitude[n] = value[0]
    elif n == index[1]:
        magnitude[n] = value[1]
    elif n == index[2]:
        magnitude[n] = value[2]
    elif n == index[3]:
        magnitude[n] = value[3]
    else:
        magnitude[n] = 0
sigC = []
for n in range (length):
    val1 = magnitude[n]*math.cos(phase[n])
    val2 = magnitude[n]*math.sin(phase[n])
    sigC.append(np.complex(val1,val2))
#find inverse transform
waveifft = np.fft.ifft(sigC)
#write back the signal object
sig = Signal(data=waveifft)
sig.wav_write('Ques5_3Output.wav')
```

Effects - The output audio has 4 places where it gave different sound than the usual straight sound in the previous cases. There were high nodes at the corresponding high magnitudes that were available.

Output File -

[https://github.com/JagjitUvic/MIR/blob/master/Assignment%201/Ques5\\_3Output.wav](https://github.com/JagjitUvic/MIR/blob/master/Assignment%201/Ques5_3Output.wav)