Fourier Project

Classification And Categorization Of Songs And Gravitational Waves Using Fourier Analysis

By Meet Vyas

AU2040084

Problem

With the increase in audio tracks and artists over the years, identifying a song by ear becomes almost impossible and there is a need to know the song's artist by the track's analysis for the artist to gain popularity and recognition. Apps like Shazam and Google use algorithms to recognize songs based on their frequencies. Similarly, the data obtained from a black hole merger or a neutron merger has a definite frequency which is captured by the LIGO observatories situated at different locations. These mergers produce gravitational waves, the majority of which are recorded by LIGO either in the range of 4000Hz or 16000Hz. To distinguish between these frequencies and analyze these wave signals, the same method is used thus extending the initial problem into another domain of cosmology.

Methodology for solving the problem

Step -1

Removing the noise through Fourier Transform and creating a Constellation Map

The audio signals captured by the microphone contain a lot of noise that needs to be filtered which is where Fourier analysis is very important. Fourier Transform essentially removes the noise from the signal as it maps the relative amount of specific frequency in comparison to other frequencies. After that, we use the scipy library in python extensively for the understanding of the signal and its characteristics. First, we find the peaks of the signal and then use the prominent peak parameter to only map the prominent peaks in the signal. By specifying the minimum distance between prominent peaks and the number of peaks we find the representative peaks for different sections of the audio signal. Now using the short-time Fourier transform we can separate the audio signal into different windows where we can check the peaks for each window and the analysis becomes much richer. Now using these transformed windows we again find the peaks for the signal and now form a constellation map of the peak frequencies by creating a plot with time on the x-axis and frequency on the y-axis. A constellation map will be unique to each song and thus a way of identifying a song has been established.

Code

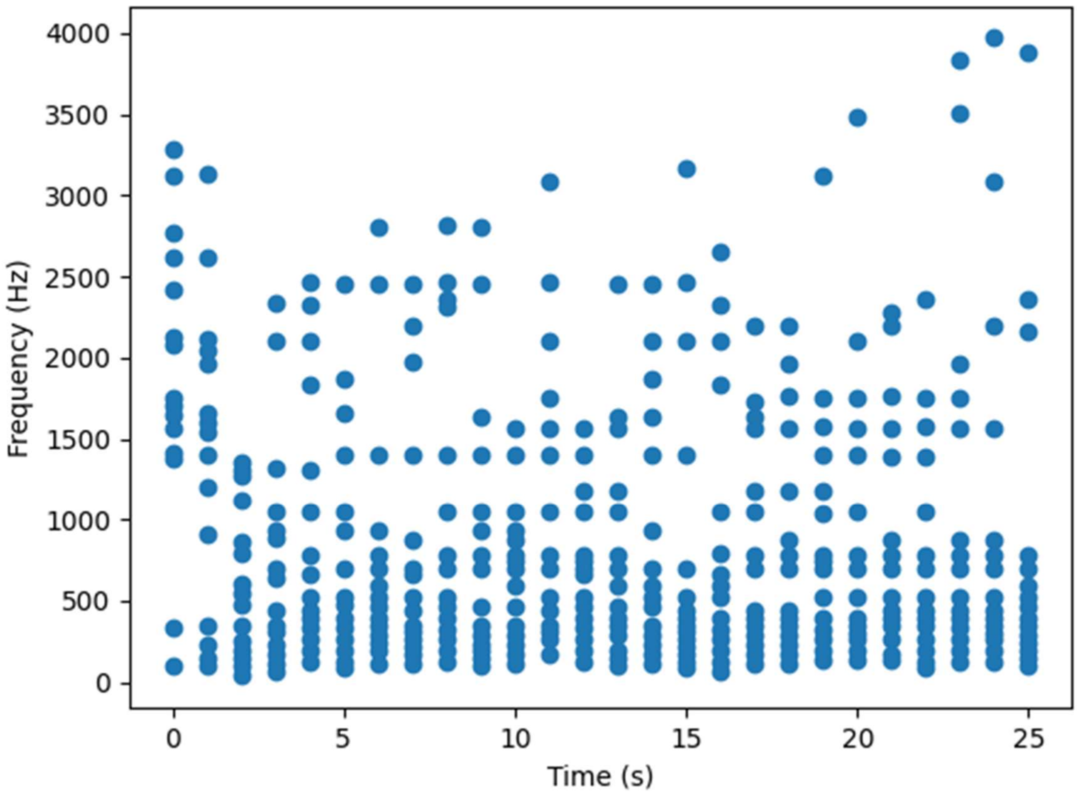
```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fft, signal
from scipy.io.wavfile import read

# The fourier transform can determine frequencies from 0 to the
Nyquist frequency(half of the sampling frequency)
#fs is the sampling frequency
#window = Desired window to use. If window is a string or tuple,
it is passed to get_window to generate the window values, which
are DFT-even by default.
def create_constellation(audio, Fs):
    # Parameters
    window_length_seconds = 5
```

```
    window_length_samples = int(window_length_seconds * Fs)
    window_length_samples += window_length_samples % 2
    number_of_peaks = 15

    # Pad the song to divide evenly into windows
    padding_amount = window_length_samples - input.size %
window_length_samples

    song_input = np.pad(audio, (0, padding_amount))
    #np-pad uses arguments including array and pad-width which
are the two used here

    # Perform a short time fourier transform
    frequencies, times,stft = signal.stft(
        song_input, Fs, nperseg=window_length_samples,
nfft=window_length_samples, return_onesided=True
    )

    print(stft.shape)
   #The size of f is either equal to the hop size H = nperseg -
noverlap or half the value of nfft
   #stft uses the follwing parameters - x(time series of
measurement values which over here is just the song input)
   #fs which is the sampling frequency of the x time series
   #nperseg is the length of each segment defaulted to 256
   #noverlap(not used here) which is the number to points to
overlap between segments which is defaulted to none
   #return_onesided - If True, return a one-sided spectrum for
real data. If False return a two-sided spectrum. Defaults to
True, but for complex data, a two-sided spectrum is always
returned. Since we are dealing with only real data we set the
parameter to True

    constellation_map = []
```

```
    for time_idx, window in enumerate(stft.T):
        # The Spectrum is complex by default and thus we will
turn into into real values
        spectrum = abs(window)
        peaks, props = signal.find_peaks(spectrum, prominence=0,
distance=200)

        #signal.find_peaks finds peaks inside a signal based on
peak properties
        #It takes the following parameters
        # x - A signal with peaks
        # height - Required height of peaks. Either a number,
None, an array matching x or a 2-element sequence of the former.
The first element is always interpreted as the minimal and the
second, if supplied, as the maximal required height.
        # threshold - Required threshold of peaks, the vertical
distance to its neighboring samples. Either a number, None, an
array matching x or a 2-element sequence of the former. The
first element is always interpreted as the minimal and the
second, if supplied, as the maximal required threshold.
        # distance - Required minimal horizontal distance (>= 1)
in samples between neighbouring peaks. Smaller peaks are removed
first until the condition is fulfilled for all remaining
peaks.(Here we want an even spread across the spectrum)
        # prominence - Required prominence of peaks. Either a
number, None, an array matching x or a 2-element sequence of the
former. The first element is always interpreted as the minimal
and the second, if supplied, as the maximal required prominence.
        # width - Used for calculation of the peaks prominences,
thus it is only used if one of the arguments prominence or width
is given.
        # Only want the most prominent peaks
        # With a maximum of 15 per time slice
```

```
        n_peaks = min(number_of_peaks, len(peaks))
        # By using this method we get the number of peaks and
largest peaks from the prominences
        # This is an argpartition
        # A greater understanding of how this works can be
looked over at: https://kanoki.org/2020/01/14/find-k-smallest-
and-largest-values-and-its-indices-in-a-numpy-array/
        largest_peaks = np.argpartition(props["prominences"], -
n_peaks)[-n_peaks:]
        for peak in peaks[largest_peaks]:
            frequency = frequencies[peak]
            constellation_map.append([time_idx, frequency])

    return constellation_map

# This forms a constellation of points which characterize the
song

Fs, input = read("C:/Users/mjvya/OneDrive/Desktop/Mechanical
Semester-5/Fourier Analysis/Fourier Project/untouchable.wav")
constellation_map = create_constellation(input, Fs)

plt.scatter(*zip(*constellation_map))
plt.ylabel("Frequency (Hz)")
plt.xlabel("Time (s)")
plt.show()
```

Output



Step -2
Creating a Hashmap using Constellation Map

After that, we use these constellation maps with the concept of hashmaps by creating hashmaps of different frequencies paired with each other which are stored with the time between them. If the two frequencies are then converted to 10-bit integers (from 0 to 1024 by placing the exact frequency into a bin) and the time difference between them stored as a 12-bit integer, the pair of points produces a single 32-bit integer hash. This produces many times more candidate fingerprints for a song than simply with the constellations, and it is also extremely efficient for the computer to match hashes in the cell phone recording with hashes stored in the song database.

Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fft, signal
from scipy.io.wavfile import read
from constellation_map import create_constellation

#The points in the constellation map are combinatorially
associated - each point is paired with several other points to
form pairs of frequencies, stored with the difference in time
between them.


Fs, audio_input =
read("C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-
5/Fourier Analysis/Fourier Project/untouchable.wav")

constellation_map = create_constellation(audio_input, Fs)
upper_frequency = 20_000
frequency_bits = 10


def create_hashes(constellation_map, song_id=None):
    hashes = {}
    # assume pre-sorted
    # Iterate the constellation
    for idx, (time, freq) in enumerate(constellation_map):
        # Iterate the next 100 pairs to produce the
combinatorial hashes
```

```
        for other_time, other_freq in constellation_map[idx :
idx + 100]:
            diff = other_time - time
            # If the time difference between the pairs is too
small or large
            # ignore this set of pairs
            if diff <= 1 or diff > 10:
                continue

            # Place the frequencies (in Hz) into a 65536 bins
            freq_binned = freq / upper_frequency * (2 **
frequency_bits)
            other_freq_binned = other_freq / upper_frequency *
(2 ** frequency_bits)

            # Produce a 32 bit hash
            hash = int(freq_binned) | (int(other_freq_binned) <<
10) | (int(diff) << 20)
            hashes[hash] = (time, song_id)
    return hashes



#Check 100 hashes produced
hashes = create_hashes(constellation_map, 0)
for i, (hash, (time, _)) in enumerate(hashes.items()):
    if i > 100:
        break
    print(f"Hash {hash} occurred at {time}")
```

Output

Hash 2180218 occurred at 2254
Hash 3167354 occurred at 16
Hash 4236410 occurred at 16
Hash 5254266 occurred at 16
Hash 6407290 occurred at 16
Hash 7423098 occurred at 2365
Hash 8399994 occurred at 16
Hash 9583738 occurred at 1952
Hash 10548346 occurred at 3012
Hash 2117969 occurred at 17
Hash 3187025 occurred at 17
Hash 4204881 occurred at 17
Hash 5357905 occurred at 17
Hash 6374737 occurred at 17
Hash 7350609 occurred at 17
Hash 8535377 occurred at 17
Hash 9499985 occurred at 17
Hash 10590545 occurred at 17
Hash 2138193 occurred at 24034
Hash 3156049 occurred at 24182
Hash 4309073 occurred at 1839
Hash 5325905 occurred at 24037
Hash 6301777 occurred at 24182
Hash 7486545 occurred at 2125
Hash 8451153 occurred at 17533
Hash 9541713 occurred at 6886
Hash 10642513 occurred at 2077
Hash 2107412 occurred at 24186
Hash 3260436 occurred at 23946
Hash 4277268 occurred at 23437
Hash 5253140 occurred at 24186
Hash 6437908 occurred at 11637
Hash 7402516 occurred at 21067
Hash 8493076 occurred at 22585
Hash 9593876 occurred at 22843

Hash 10506260 occurred at 24156
Hash 2211880 occurred at 2178
Hash 3228712 occurred at 24040
Hash 4204584 occurred at 24059
Hash 5389352 occurred at 2803
Hash 6353960 occurred at 12394
Hash 7444520 occurred at 22589
Hash 8545320 occurred at 2418
Hash 9457704 occurred at 24060
Hash 10526760 occurred at 24050
Hash 2180106 occurred at 24180
Hash 3155978 occurred at 24261
Hash 4340746 occurred at 8135
Hash 5305354 occurred at 24261
Hash 6395914 occurred at 19370
Hash 7496714 occurred at 24261
Hash 8409098 occurred at 24178
Hash 9478154 occurred at 24046
Hash 10568714 occurred at 24257
Hash 2107504 occurred at 24239
Hash 3292272 occurred at 1744
Hash 4256880 occurred at 1661
Hash 5347440 occurred at 3446
Hash 6448240 occurred at 23505
Hash 7360624 occurred at 23871
Hash 8429680 occurred at 23963
Hash 9520240 occurred at 23968
Hash 10610800 occurred at 2210
Hash 2243665 occurred at 1522
Hash 3208273 occurred at 22034
Hash 4298833 occurred at 1683
Hash 5399633 occurred at 2011
Hash 6312017 occurred at 24042
Hash 7381073 occurred at 24043
Hash 8471633 occurred at 24035

Hash 9562193 occurred at 2125
Hash 10558545 occurred at 19703
Hash 2159626 occurred at 24264
Hash 3250186 occurred at 23579
Hash 4350986 occurred at 24264
Hash 5263370 occurred at 24165
Hash 6332426 occurred at 24049
Hash 7422986 occurred at 24260
Hash 8513546 occurred at 24173
Hash 9509898 occurred at 22971
Hash 10548234 occurred at 24256
Hash 2201743 occurred at 2420
Hash 3302543 occurred at 1462
Hash 4214927 occurred at 7715
Hash 5283983 occurred at 358
Hash 6374543 occurred at 1616
Hash 7465103 occurred at 2170
Hash 8461455 occurred at 2040
Hash 9499791 occurred at 1162
Hash 10506383 occurred at 11643
Hash 2253885 occurred at 24266
Hash 3166269 occurred at 22165
Hash 4235325 occurred at 17563
Hash 5325885 occurred at 22037
Hash 6416445 occurred at 3047
Hash 7412797 occurred at 21074
Hash 8451133 occurred at 19707
Hash 9457725 occurred at 21075
Hash 10548285 occurred at 17553
Hash 2117734 occurred at 22595
Hash 3186790 occurred at 20719

Step-3
Creating a database and storing it using Pickle

Here we use the wav files in the folder and create a database for the different songs and audios to be compared and scored against using the pickle library which is used to serialize python objects. Here the library tqdm is also used which is a neat way of understanding the progress of the output of the code.

Code

```
import numpy as np
import matplotlib.pyplot as plt
import os
import glob
from typing import List, Dict, Tuple
from tqdm import tqdm
from scipy import fft, signal
import pickle
from scipy.io.wavfile import read
from constellation_map import create_constellation
from hashes import create_hashes

import glob
from typing import List, Dict, Tuple
from tqdm import tqdm
import pickle
songs = glob.glob('C:/Users/mjvya/OneDrive/Desktop/Mechanical
Semester-5/Fourier Analysis/Fourier Project/*.wav')
song_name_index = {}
database: Dict[int, List[Tuple[int, int]]] = {}
# Go through each song, using where they are alphabetically as
an id
for index, filename in enumerate(tqdm(sorted(songs))):
    song_name_index[index] = filename
    # Read the song, create a constellation and hashes
```

```python
    Fs, audio_input = read(filename)
    constellation = create_constellation(audio_input, Fs)
    hashes = create_hashes(constellation, index)
    # For each hash, append it to the list for this hash
    for hash, time_index_pair in hashes.items():
        if hash not in database:
            database[hash] = []
        database[hash].append(time_index_pair)
# Dump the database and list of songs as pickles
with open("database.pickle", 'wb') as db:
    pickle.dump(database, db, pickle.HIGHEST_PROTOCOL)
with open("song_index.pickle", 'wb') as songs:
    pickle.dump(song_name_index, songs, pickle.HIGHEST_PROTOCOL)
```

Output

```
100%|██████████████████████████████████████████████████| 32/32
[00:06<00:00,  4.64it/s]
```

Step -4

Finding matches between songs and creating a scoring system

Now by matching the time of the hashes we can determine the similarity between different audio clips. We use time as a factor because using the hashes produced would give similar results as all the hashes are 32-bit integers. Due to this, we can get incorrect results while matching the hashes for different audio clips. Because the hashes will only have a particular peak at a particular time, the songs can be compared in the domain of time as the peaks will be almost at the same time regardless of the loss of quality and signal noise.

Code

Method -1

Comparing peaks in hash-maps in the frequency domain

Method -1

Comparing peaks in hash-maps in the frequency domain

Code

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
from scipy import fft, signal
from scipy.io.wavfile import read
from constellation_map import create_constellation
from hashes import create_hashes

# Load the database
database = pickle.load(open('database.pickle', 'rb'))
song_name_index = pickle.load(open("song_index.pickle", "rb"))

# Loading a short recording of the song sung by me with the same
guitar chords
Fs, audio_input =
read("C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-
5/Fourier Analysis/Fourier Project/untouchable_meet.wav")
# Create the constellation and hashes
constellation = create_constellation(audio_input, Fs)
hashes = create_hashes(constellation, None)
# For each hash in the song, check if there's a match in the
database
# There could be multiple matching tracks, so for each match we
increase the id counter of the song by 1
```

```
matches_per_song = {}
for hash, (sample_time, _) in hashes.items():
    if hash in database:
        matching_occurences = database[hash]
        for source_time, song_id in matching_occurences:
            if song_id not in matches_per_song:
                matches_per_song[song_id] = 0
            matches_per_song[song_id] += 1
for song_id, num_matches in
list(sorted(matches_per_song.items(), key=lambda x: x[1],
reverse=True))[:10]:
    print(f"Song: {song_name_index[song_id]} - Matches:
{num_matches}")
```

Output

Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\untouchable_meet.wav - Matches: 8153Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\untouchable.wav - Matches: 3095
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\neutron_merger.wav - Matches: 2815Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\blackhole_2.5sm.wav - Matches: 2432
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\blackhole_100.wav - Matches: 785Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW150914_H1_whitenbp.wav - Matches: 36

Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW150914_L1_whitenbp.wav - Matches: 36
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW151226_H1_whitenbp.wav - Matches: 36
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW151226_L1_whitenbp.wav - Matches: 36
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW170104_H1_whitenbp.wav - Matches: 36

Method-2
Comparing peaks in hash-maps in the time domain

Code

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
from scipy import fft, signal
from scipy.io.wavfile import read
from constellation_map import create_constellation
from hashes import create_hashes

Fs, audio_input =
read("C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-
5/Fourier Analysis/Fourier Project/untouchable_meet.wav")


constellation = create_constellation(audio_input, Fs)
```

```
hashes = create_hashes(constellation, None)

database = pickle.load(open('database.pickle', 'rb'))
song_index_lookup = pickle.load(open("song_index.pickle", "rb"))


def score_songs(hashes):
    inidvidual_song_matches = {}
    for hash, (sample_time, _) in hashes.items():
        if hash in database:
            places_of_match = database[hash]
            for source_time, song_index in places_of_match:
                if song_index not in inidvidual_song_matches:
                    inidvidual_song_matches[song_index] = []

inidvidual_song_matches[song_index].append((hash, sample_time,
source_time))


    scores = {}
    for song_index, matches in inidvidual_song_matches.items():
        song_scores_by_offset = {}
        for hash, sample_time, source_time in matches:
            time_difference = source_time - sample_time
            if time_difference not in song_scores_by_offset:
                song_scores_by_offset[time_difference] = 0
            song_scores_by_offset[time_difference] += 1

        max = (0, 0)
        for offset, score in song_scores_by_offset.items():
            if score > max[1]:
                max = (offset, score)

        scores[song_index] = max
```

```
    # Sort the scores for different songs in accordance to the input audio
    scores = list(sorted(scores.items(), key=lambda x: x[1][1], reverse=True))

    return scores

scores = score_songs(hashes)
for song_index, score in scores:
    print(f"{song_index_lookup[song_index]=}: Score of {score[1]} at {score[0]}")
```

Output

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\untouchable_meet.wav': Score of 8153 at 0
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\neutron_merger.wav': Score of 12 at -6246
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\blackhole_2.5sm.wav': Score of 10 at -16694
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\untouchable.wav': Score of 9 at 2738
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\blackhole_100.wav': Score of 8 at -21440

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170814-56.wav': Score of 4 at -20792
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_H1_whitenbp.wav': Score of 3 at -20000
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_L1_whitenbp.wav': Score of 3 at -20011
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_H1_whitenbp.wav': Score of 3 at -19989
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_L1_whitenbp.wav': Score of 3 at -19979
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_H1_whitenbp.wav': Score of 3 at -19969
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_L1_whitenbp.wav': Score of 3 at -19951
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170729-84.9.wav': Score of 3 at -20775
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170818-62.3.wav': Score of 3 at -20765
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_H1_whitenbp.wav': Score of 3 at -19945

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_L1_whitenbp.wav': Score of 3 at -19979
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170817-2.73.wav': Score of 3 at -18794
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_template_whiten.wav': Score of 3 at -20749
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914-66.2.wav': Score of 3 at -20778
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_template_whiten.wav': Score of 3 at -20789
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226-21.4.wav': Score of 3 at -20796
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_template_whiten.wav': Score of 3 at -20799
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_template_whiten.wav': Score of 3 at -20791
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170823-69.wav': Score of 3 at -20793
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104-51.1.wav': Score of 2 at -20780

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170809-59.wav': Score of 2 at -20748
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_template_shifted.wav': Score of 2 at -20780
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_template_shifted.wav': Score of 2 at -20788
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_template_shifted.wav': Score of 2 at -20780
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_template_shifted.wav': Score of 2 at -20776
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151012-36.9.wav': Score of 2 at -20798
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170608-18.5.wav': Score of 2 at -20795

Step-5
Extending the analysis to gravitational waves

LIGO has officially published the data for 12 gravitational waves, some of which are publicly available and have been used in this analysis. These audio samples are plotted with the strain which is the distance between the two arms of the detector divided by the length of the detector which is 4 kilometers. The audio signals are created using python code from the data gathered in json format by the scientists working at LIGO. These audio signals contain a lot of signal-to-noise ratio as well as need to be calibrated according to the equipment at the LIGO observatories. Thus the audios available are whitened and band passed. Even though there are these constraints, the above analysis can be used to find similarities between different mergers and also have an in-depth look at how the waveforms of the gravitational waves appear. Gravitational waves based on the strain data at 4000 Hz and 16000Hz have been prepared by scientists which are beautifully visualized using the above method and shown below. Thus we can create a very insightful method to peer and understand the nature of gravitational waves.

Step-1
Redefining the window length to fit the current data

Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fft, signal
from scipy.io.wavfile import read

# The fourier transform can determine frequencies from 0 to the
Nyquist frequency(half of the sampling frequency)
#fs is the sampling frequency
#window = Desired window to use. If window is a string or tuple,
it is passed to get_window to generate the window values, which
are DFT-even by default.
def create_constellation(audio, Fs):
    # Parameters
    window_length_seconds = 2
```

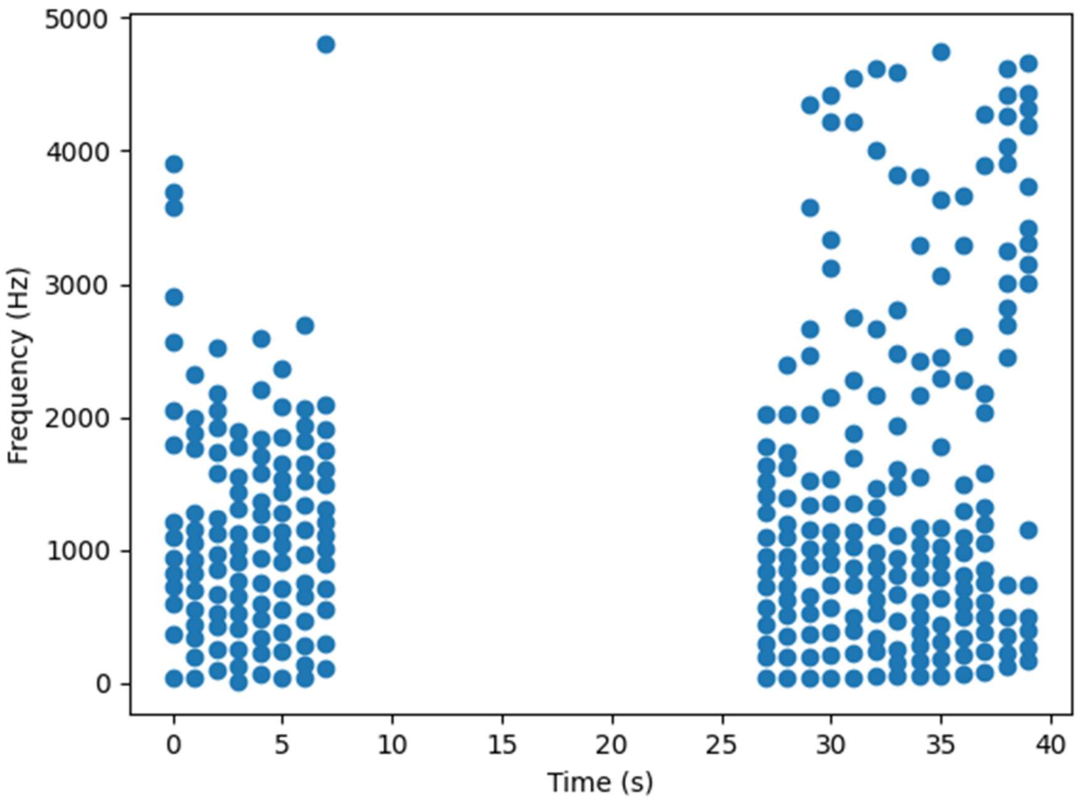
```
    window_length_samples = int(window_length_seconds * Fs)
    window_length_samples += window_length_samples % 2
    number_of_peaks = 15

    # Pad the song to divide evenly into windows
    padding_amount = window_length_samples - input.size %
window_length_samples

    song_input = np.pad(audio, (0, padding_amount))
    #np-pad uses arguments including array and pad-width which
are the two used here

    # Perform a short time fourier transform
    frequencies, times,stft = signal.stft(
        song_input, Fs, nperseg=window_length_samples,
nfft=window_length_samples, return_onesided=True
    )

    print(stft.shape)
   #The size of f is either equal to the hop size H = nperseg -
noverlap or half the value of nfft
   #stft uses the follwing parameters - x(time series of
measurement values which over here is just the song input)
   #fs which is the sampling frequency of the x time series
   #nperseg is the length of each segment defaulted to 256
   #noverlap(not used here) which is the number to points to
overlap between segments which is defaulted to none
   #return_onesided - If True, return a one-sided spectrum for
real data. If False return a two-sided spectrum. Defaults to
True, but for complex data, a two-sided spectrum is always
returned. Since we are dealing with only real data we set the
parameter to True

    constellation_map = []
```

```
    for time_idx, window in enumerate(stft.T):
        # The Spectrum is complex by default and thus we will
turn into into real values
        spectrum = abs(window)
        peaks, props = signal.find_peaks(spectrum, prominence=0,
distance=200)

        #signal.find_peaks finds peaks inside a signal based on
peak properties
        #It takes the following parameters
        # x - A signal with peaks
        # height - Required height of peaks. Either a number,
None, an array matching x or a 2-element sequence of the former.
The first element is always interpreted as the minimal and the
second, if supplied, as the maximal required height.
        # threshold - Required threshold of peaks, the vertical
distance to its neighboring samples. Either a number, None, an
array matching x or a 2-element sequence of the former. The
first element is always interpreted as the minimal and the
second, if supplied, as the maximal required threshold.
        # distance - Required minimal horizontal distance (>= 1)
in samples between neighbouring peaks. Smaller peaks are removed
first until the condition is fulfilled for all remaining
peaks.(Here we want an even spread across the spectrum)
        # prominence - Required prominence of peaks. Either a
number, None, an array matching x or a 2-element sequence of the
former. The first element is always interpreted as the minimal
and the second, if supplied, as the maximal required prominence.
        # width - Used for calculation of the peaks prominences,
thus it is only used if one of the arguments prominence or width
is given.
        # Only want the most prominent peaks
        # With a maximum of 15 per time slice
```

```
        n_peaks = min(number_of_peaks, len(peaks))
        # By using this method we get the number of peaks and
largest peaks from the prominences
        # This is an argpartition
        # A greater understanding of how this works can be
looked over at: https://kanoki.org/2020/01/14/find-k-smallest-
and-largest-values-and-its-indices-in-a-numpy-array/
        largest_peaks = np.argpartition(props["prominences"], -
n_peaks)[-n_peaks:]
        for peak in peaks[largest_peaks]:
            frequency = frequencies[peak]
            constellation_map.append([time_idx, frequency])

    return constellation_map

# This forms a constellation of points which characterise the
song

Fs, input = read("C:/Users/mjvya/OneDrive/Desktop/Mechanical
Semester-5/Fourier Analysis/Fourier Project/neutron_merger.wav")
constellation_map = create_constellation(input, Fs)

plt.scatter(*zip(*constellation_map))
plt.ylabel("Frequency (Hz)")
plt.xlabel("Time (s)")
plt.show()
```

Here the window time has been fitted to set the time of the audio which is around 40 seconds and thus the constellation map has been formatted for the same. The output also prints the size of the short-time fourier transform which in this case is (9601, 41).

Output



Step-2
Create and check the hashes for the same

Code

```
import numpy as np
import matplotlib.pyplot as plt
from scipy import fft, signal
from scipy.io.wavfile import read
```

```python
from constellation_map import create_constellation

#The points in the constellation map are combinatorially
associated - each point is paired with several other points to
form pairs of frequencies, stored with the difference in time
between them.


Fs, audio_input =
read("C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-
5/Fourier Analysis/Fourier Project/neutron_merger.wav")

constellation_map = create_constellation(audio_input, Fs)
upper_frequency = 20_000
frequency_bits = 10


def create_hashes(constellation_map, song_id=None):
    hashes = {}
    # assume pre-sorted
    # Iterate the constellation
    for idx, (time, freq) in enumerate(constellation_map):
        # Iterate the next 100 pairs to produce the
combinatorial hashes
        for other_time, other_freq in constellation_map[idx :
idx + 100]:
            diff = other_time - time
            # If the time difference between the pairs is too
small or large
            # ignore this set of pairs
            if diff <= 1 or diff > 10:
                continue

            # Place the frequencies (in Hz) into a 65536 bins
```

```
            freq_binned = freq / upper_frequency * (2 **
frequency_bits)
            other_freq_binned = other_freq / upper_frequency *
(2 ** frequency_bits)

            # Produce a 32 bit hash
            hash = int(freq_binned) | (int(other_freq_binned) <<
10) | (int(diff) << 20)
            hashes[hash] = (time, song_id)
    return hashes



#Check 100 hashes produced
hashes = create_hashes(constellation_map, 0)
for i, (hash, (time, _)) in enumerate(hashes.items()):
    if i > 100:
        break
    print(f"Hash {hash} occurred at {time}")
```

Output

Hash 2159687 occurred at 12097
Hash 3208263 occurred at 12252
Hash 4204615 occurred at 15272
Hash 5295175 occurred at 12016
Hash 6343751 occurred at 12004
Hash 7402567 occurred at 12097
Hash 8440903 occurred at 12013
Hash 9509959 occurred at 12135
Hash 10538055 occurred at 12000

Hash 2159667 occurred at 15349
Hash 3156019 occurred at 11925
Hash 4246579 occurred at 11925
Hash 5295155 occurred at 11870
Hash 6353971 occurred at 11897
Hash 7392307 occurred at 11929
Hash 8461363 occurred at 11874
Hash 9489459 occurred at 11837
Hash 10548275 occurred at 11874
Hash 2107453 occurred at 15286
Hash 3198013 occurred at 11894
Hash 4246589 occurred at 12006
Hash 5305405 occurred at 12099
Hash 6343741 occurred at 11972
Hash 7412797 occurred at 12090
Hash 8440893 occurred at 11867
Hash 9499709 occurred at 12255
Hash 10558525 occurred at 12099
Hash 2149437 occurred at 12046
Hash 4256829 occurred at 12095
Hash 5295165 occurred at 11903
Hash 6364221 occurred at 12082
Hash 7392317 occurred at 11868
Hash 8451133 occurred at 12082
Hash 9509949 occurred at 12079
Hash 10496061 occurred at 15286
Hash 2149386 occurred at 12008
Hash 3208202 occurred at 15281
Hash 4246538 occurred at 11871
Hash 5315594 occurred at 15267

Hash 6343690 occurred at 11947
Hash 7402506 occurred at 15279
Hash 8461322 occurred at 15250
Hash 9447434 occurred at 15320
Hash 10610698 occurred at 15212
Hash 3198003 occurred at 11834
Hash 4267059 occurred at 12010
Hash 7412787 occurred at 11875
Hash 8398899 occurred at 11953
Hash 9562163 occurred at 11711
Hash 10526771 occurred at 11693
Hash 2149427 occurred at 11896
Hash 3218483 occurred at 12010
Hash 5305395 occurred at 12021
Hash 6364211 occurred at 12048
Hash 7350323 occurred at 11864
Hash 8513587 occurred at 11712
Hash 9478195 occurred at 11874
Hash 2169917 occurred at 12095
Hash 5315645 occurred at 12159
Hash 6301757 occurred at 15286
Hash 7465021 occurred at 12104
Hash 8429629 occurred at 11727
Hash 9478205 occurred at 11745
Hash 3208243 occurred at 15349
Hash 5253171 occurred at 11756
Hash 6416435 occurred at 11701
Hash 7381043 occurred at 11692
Hash 8429619 occurred at 11875
Hash 9447475 occurred at 11824

Hash 3218503 occurred at 12141
Hash 5367879 occurred at 15258
Hash 6332487 occurred at 11491
Hash 7381063 occurred at 11628
Hash 8398919 occurred at 15272
Hash 9478215 occurred at 11851
Hash 10526791 occurred at 11586
Hash 2169907 occurred at 11953
Hash 4319283 occurred at 11802
Hash 5283891 occurred at 11654
Hash 6332467 occurred at 11697
Hash 10538035 occurred at 11898
Hash 3270717 occurred at 12264
Hash 4235325 occurred at 11879
Hash 5283901 occurred at 11855
Hash 7381053 occurred at 11728
Hash 9489469 occurred at 11944
Hash 10590269 occurred at 11848
Hash 2222151 occurred at 12109
Hash 3186759 occurred at 11455
Hash 4235335 occurred at 11432
Hash 5253191 occurred at 15272
Hash 9541703 occurred at 12263
Hash 2138122 occurred at 11857
Hash 3186698 occurred at 11857
Hash 4204554 occurred at 15325
Hash 5283850 occurred at 11621
Hash 6332426 occurred at 11679
Hash 7392266 occurred at 11890
Hash 8493066 occurred at 15277

Hash 9489418 occurred at 11747
Hash 10537994 occurred at 11814

Step-3
Find scores with other merger files from the database using the two methods mentioned above

Method -1
Comparing peaks in hash-maps in the frequency domain

Code

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
from scipy import fft, signal
from scipy.io.wavfile import read
from constellation_map import create_constellation
from hashes import create_hashes

# Load the database
database = pickle.load(open('database.pickle', 'rb'))
song_name_index = pickle.load(open("song_index.pickle", "rb"))

# Loading a short recording of the song sung by me with the same
guitar chords
Fs, audio_input =
read("C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-
5/Fourier Analysis/Fourier Project/neutron_merger.wav")
# Create the constellation and hashes
constellation = create_constellation(audio_input, Fs)
hashes = create_hashes(constellation, None)
```

```
# For each hash in the song, check if there's a match in the database
# There could be multiple matching tracks, so for each match we increase the id counter of the song by 1
matches_per_song = {}
for hash, (sample_time, _) in hashes.items():
    if hash in database:
        matching_occurences = database[hash]
        for source_time, song_id in matching_occurences:
            if song_id not in matches_per_song:
                matches_per_song[song_id] = 0
            matches_per_song[song_id] += 1
for song_id, num_matches in list(sorted(matches_per_song.items(), key=lambda x: x[1], reverse=True))[:10]:
    print(f"Song: {song_name_index[song_id]} - Matches: {num_matches}")
```

Output

Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\neutron_merger.wav - Matches: 4004
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\blackhole_2.5sm.wav - Matches: 3162
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\untouchable.wav - Matches: 2889
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\untouchable_meet.wav - Matches: 2815
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\blackhole_100.wav - Matches: 910

Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW150914_H1_whitenbp.wav - Matches: 27
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW150914_L1_whitenbp.wav - Matches: 27
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW151226_H1_whitenbp.wav - Matches: 27
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW151226_L1_whitenbp.wav - Matches: 27
Song: C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\GW170104_H1_whitenbp.wav - Matches: 27

Method-2
Comparing peaks in hash-maps in the time domain

Code

```
import numpy as np
import matplotlib.pyplot as plt
import pickle
from scipy import fft, signal
from scipy.io.wavfile import read
from constellation_map import create_constellation
from hashes import create_hashes

Fs, audio_input =
read("C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-
5/Fourier Analysis/Fourier Project/neutron_merger.wav")
```

```
constellation = create_constellation(audio_input, Fs)
hashes = create_hashes(constellation, None)

database = pickle.load(open('database.pickle', 'rb'))
song_index_lookup = pickle.load(open("song_index.pickle", "rb"))


def score_songs(hashes):
    inidvidual_song_matches = {}
    for hash, (sample_time, _) in hashes.items():
        if hash in database:
            places_of_match = database[hash]
            for source_time, song_index in places_of_match:
                if song_index not in inidvidual_song_matches:
                    inidvidual_song_matches[song_index] = []

inidvidual_song_matches[song_index].append((hash, sample_time,
source_time))


    scores = {}
    for song_index, matches in inidvidual_song_matches.items():
        song_scores_by_offset = {}
        for hash, sample_time, source_time in matches:
            time_difference = source_time - sample_time
            if time_difference not in song_scores_by_offset:
                song_scores_by_offset[time_difference] = 0
            song_scores_by_offset[time_difference] += 1

        max = (0, 0)
        for offset, score in song_scores_by_offset.items():
            if score > max[1]:
```

```
                max = (offset, score)

        scores[song_index] = max

    # Sort the scores for different songs in accordance to the
input audio
    scores = list(sorted(scores.items(), key=lambda x: x[1][1],
reverse=True))

    return scores

scores = score_songs(hashes)
for song_index, score in scores:
    print(f"{song_index_lookup[song_index]=}: Score of
{score[1]} at {score[0]}")
```

Output

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\neutron_merger.wav': Score of 4004 at 0
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\blackhole_2.5sm.wav': Score of 23 at -8727
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170817-2.73.wav': Score of 17 at -12506
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\untouchable.wav': Score of 14 at 8984

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\untouchable_meet.wav': Score of 12 at 6246
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_template_whiten.wav': Score of 12 at -14508
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_template_whiten.wav': Score of 9 at -14501
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_template_whiten.wav': Score of 9 at -14503
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_template_shifted.wav': Score of 9 at -14534
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_template_shifted.wav': Score of 9 at -14542
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_template_shifted.wav': Score of 9 at -14534
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_template_shifted.wav': Score of 9 at -14530
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\blackhole_100.wav': Score of 8 at -15016
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151012-36.9.wav': Score of 7 at -14510

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_template_whiten.wav': Score of 7 at -14511
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914-66.2.wav': Score of 6 at -14500
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_H1_whitenbp.wav': Score of 6 at -13691
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_L1_whitenbp.wav': Score of 6 at -13693
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170729-84.9.wav': Score of 6 at -14498
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226-21.4.wav': Score of 5 at -14511
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_H1_whitenbp.wav': Score of 5 at -13704
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170608-18.5.wav': Score of 5 at -14512
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170809-59.wav': Score of 5 at -14506
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170814-56.wav': Score of 5 at -14504

song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_H1_whitenbp.wav': Score of 5 at -13693
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\LVT151012_L1_whitenbp.wav': Score of 5 at -13691
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170823-69.wav': Score of 5 at -14505
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW151226_H1_whitenbp.wav': Score of 4 at -13701
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104-51.1.wav': Score of 4 at -14510
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170104_L1_whitenbp.wav': Score of 4 at -13705
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW170818-62.3.wav': Score of 4 at -14503
song_index_lookup[song_index]='C:/Users/mjvya/OneDrive/Desktop/Mechanical Semester-5/Fourier Analysis/Fourier Project\\GW150914_L1_whitenbp.wav': Score of 3 at -13702

Conclusions and Results

Thus by using the key concepts of fourier analysis real world problems like song recognition and understanding the behaviour of gravitational waves can be done which is very crucial in increasing our understanding of the universe.