Volume 12 Issue 1 (2023)

Volume-Independent Music Matching by Frequency Spectrum Comparison

Anthony Lee[1] and James Choi[#]

[1]Milton Academy, Milton, MA, USA
[#]Advisor

ABSTRACT

Currently, there are applications such as Shazam that provides music matching. However, a limitation is that the same piece performed by the same musician cannot be identified if it is not the same recording. This is because Shazam matches the variation in volume, not the frequencies of the sound. This research attempts to match music the way humans understand it: by the frequency spectrum of music, not the volume variation. We pre-computed the frequency spectrums of the music, then took the unknown piece and tried to match its frequency spectrum against every segment. We did so by sliding the window by 0.1 seconds and calculating the error by subtracting the normalized arrays and taking the sum of absolute differences. The segment that showed the least error was considered the candidate for the match. Matching simple pieces such as single-note pieces was successful, but complex pieces such as symphonies were not successful; that is, the algorithm couldn't produce low error value in any of the music in the database. We suspect that it has to do with having "too many notes," i.e., mismatches in the higher harmonics added up to significant amount of errors, which swamps the calculations.

Introduction

There are moments when one hears a beautiful piece of music but cannot figure out the name of it. Indeed, there is an app called Shazam which somewhat achieves this goal. However, Shazam does not match music. Instead, it identifies a particular recording of a particular music [1]. In fact, it cannot match the same music played by the same musician if they are two different recordings [1]. This is because Shazam isn't comparing frequencies but, instead, it is comparing the loudness variation of the music in time [2]. However, this isn't how humans perceive music. People hear pitches, and harmonies. Our algorithm tries to humanize the music recognition by emphasizing these two factors by using spectrogram. When somebody whistles or hums a melody, even if the rhythm and notes are different, we hope to find the correct match for the piece.

According to Wolfram Documentation, Spectrogram plots the magnitude of the short-time Fourier transform (STFT), computed as a discrete Fourier transform (DFT) of partitions of list [3]. In simple terms, Spectrogram is a visual way of representing the signal, and strength, or "loudness," of a signal over time at various frequencies present in a particular waveform [4]. By visualizing Spectrogram in 3D as shown Figure 1, I can see the fundamental frequency in the front and the harmonics building up behind it. The log scale was used to make the visuals clearer. For spectrograms that depict audios, its fundamental frequency is the clearest marker in the graph, always the bottom most one. In Spectrograms, the darker the line is, the louder its volume is. Figure 2 illustrates the following. The combination of all the harmonics in the spectrogram results in the timber of the sound.

The SpectrogramArray returns the data that went into the construction of the SpectrogramArray, allowing us to use the data from our analysis. Since the Fourier Transform used in the SpectrogramArray returns complex numbers, the absolute value of it was used [5].

Another key part of the algorithm was the Partition command. Partition cuts lists into smaller portions. In Mathematica, by specifying an offset value, it was possible to cut the list with overlaps [6]. This was very useful for this code because there were more parts of the long music to compare the sample music to, which translated to the result being more accurate.

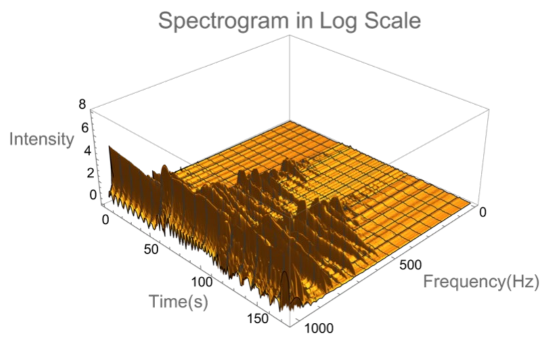


Figure 1. Spectrogram in Log Scale

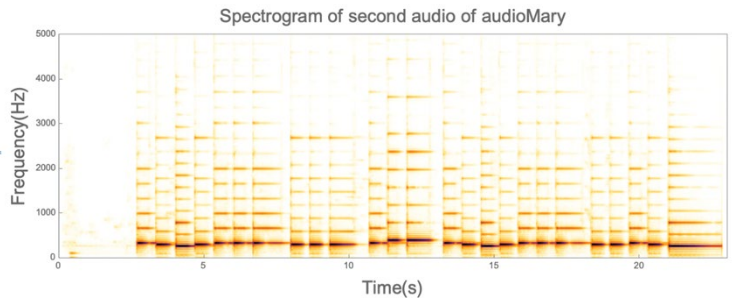


Figure 2. Spectrogram of Mary had a Little Lamb

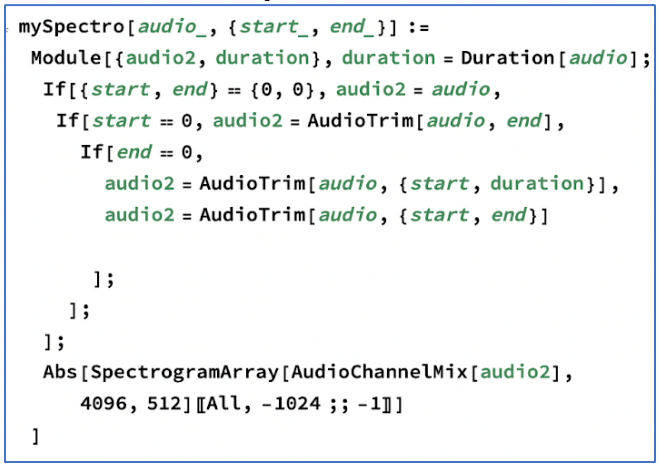
Methods

Preliminary Work

First, we stored the files in the folders and read the names of the audio files. Also, we got all the files to be used as databases, i.e., music to be compared against, and stored them in a folder. The next step was to turn every music into spectrogram arrays. When we are matching the unknown piece, we found the Spectrogram Array of it, then matched it against every piece in the database, one at a time, keeping track of the error. The one with the least error will be declared the match if the error value is reasonable.

Spectrograms & Function

As stated in the introduction, spectrograms are a visual way of representing the signal strength, or "loudness," of a signal over time at various frequencies present in a particular waveform. Mathematica has a great built-in function, and Figure 2 is a spectrogram of one of the audios used. It is clear the darker notes towards the bottom, which represent the fundamental frequencies, show the pattern and progression of the notes of *Mary Had a Little Lamb*. However, because Spectrogram is a plot, it is very difficult to extract any quantitative information from it. Therefore, we used SpectrogramArray instead, which gives the actual data. Figure 3 shows the mySpectro function which gives us the spectrogram array data from an arbitrary segment within a music specified by its parameters. The AudioChannelMix command was used to convert stereo sound into mono so that audio could be match one to one. The two channels were averaged to produce the center-panned stereo audio object. In the function, the length of the partitions (Fourier Transform time window) was set to 4096 samples for all the audio and the offset value was set to 512 samples.



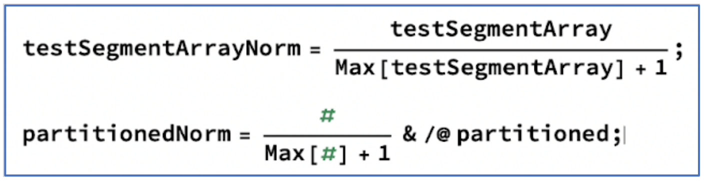
```
mySpectro[audio_, {start_, end_}] :=
 Module[{audio2, duration}, duration = Duration[audio];
  If[{start, end} == {0, 0}, audio2 = audio,
   If[start == 0, audio2 = AudioTrim[audio, end],
     If[end == 0,
       audio2 = AudioTrim[audio, {start, duration}],
       audio2 = AudioTrim[audio, {start, end}]

      ];
    ];
  ];
  Abs[SpectrogramArray[AudioChannelMix[audio2],
     4096, 512][[All, -1024 ;; -1]]]
 ]
```

Figure 3. mySpectro function

Partitioning SpectrogramArrays & Audio Normalization

Now that the mySpectro function is complete, the next step was to partition the music into the length of the small music segment. We partitioned the songs in the database to match the unknown size with the offset value of 10 sample size. The offset value means that partitioning is done with the overlap of 10 samples.
The whole idea of Audio Normalization was to avoid Shazam's way of matching the volume variations. Thus, we wanted the matching to be immune to the loudness of particular performances or recordings by normalizing the music volume. If one performance is recorded at a higher volume than the other, even if the pieces are played perfectly matching, there will be errors when the spectrograms are subtracted because there will be height differences in them. Audio Normalization ensures that the maximum volume of each music segment is the same. As loud music has higher values (manifested as taller heights in 3D, darker colors in 2D) in spectrogram, normalization corrects for this variation reducing false error.



$$\texttt{testSegmentArrayNorm} = \frac{\texttt{testSegmentArray}}{\texttt{Max[testSegmentArray] + 1}};$$

$$\texttt{partitionedNorm} = \frac{\texttt{\#}}{\texttt{Max[\#] + 1}}\ \texttt{\& /@ partitioned;}$$

Figure 4. Audio Normalization

By dividing each element by the Max volume of each element, the values in the spectrum array becomes a volume ratio to the max, rather than absolute volume. This makes the comparison more apple to apple when comparing recordings of different volumes. The +1 added to the max value seen in the code was added in the denominator to prevent dividing by 0 when the max is zero. This could happen if the music was silent. The +1 doesn't affect the value of the spectrum because they are in the thousands and +1 was added equally to both the database and the unknown segment. Therefore, if the music is truly identical, their error should be zero after their point-by-point subtraction.

Subtracting SpectrogramArrays

Now, we subtracted the spectrogram arrays of the unknown from the ones in the database. The goal was to find the interval with the least error, which meant that the segment was found in the large music. To do this, we summed up all the Absolute values of the differences because error is defined to be the sum.

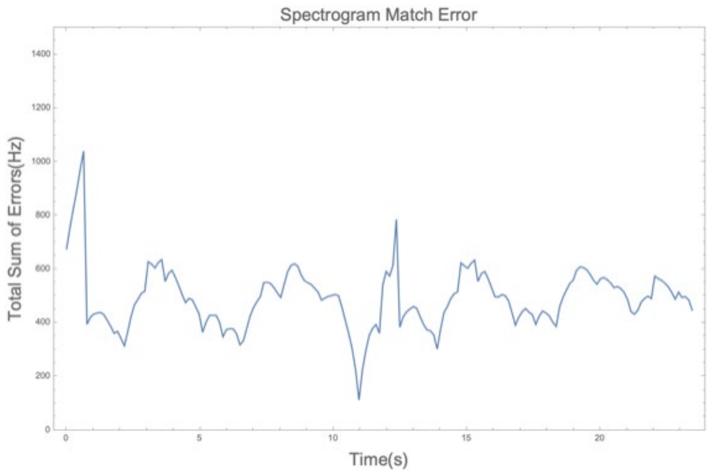
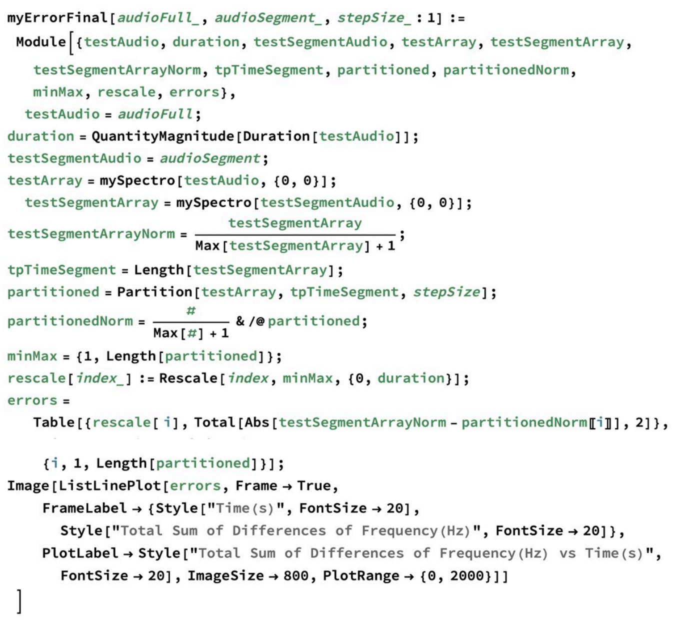


Figure 5. ListLinePlot of Errors

Figure 5 shows the ListLinePlot of errors. Clearly, the least error happened around at 11th seconds. Thus, we can identity the error by looking at the graph because the part with the dip is the part with least error, and thus is the interval we are looking for.

Full Function

We attach the full function for reference.



```
myErrorFinal[audioFull_, audioSegment_, stepSize_ : 1] :=
 Module[{testAudio, duration, testSegmentAudio, testArray, testSegmentArray,
   testSegmentArrayNorm, tpTimeSegment, partitioned, partitionedNorm,
   minMax, rescale, errors},
  testAudio = audioFull;
duration = QuantityMagnitude[Duration[testAudio]];
testSegmentAudio = audioSegment;
testArray = mySpectro[testAudio, {0, 0}];
  testSegmentArray = mySpectro[testSegmentAudio, {0, 0}];
                            testSegmentArray
testSegmentArrayNorm = ------------------------- ;
                       Max[testSegmentArray] + 1
tpTimeSegment = Length[testSegmentArray];
partitioned = Partition[testArray, tpTimeSegment, stepSize];
                      #
partitionedNorm = ----------- & /@ partitioned;
                  Max[#] + 1
minMax = {1, Length[partitioned]};
rescale[index_] := Rescale[index, minMax, {0, duration}];
errors =
   Table[{rescale[ i], Total[Abs[testSegmentArrayNorm - partitionedNorm[[i]]], 2]},

    {i, 1, Length[partitioned]}];
Image[ListLinePlot[errors, Frame → True,
    FrameLabel → {Style["Time(s)", FontSize → 20],
      Style["Total Sum of Differences of Frequency(Hz)", FontSize → 20]},
    PlotLabel → Style["Total Sum of Differences of Frequency(Hz) vs Time(s)",
      FontSize → 20], ImageSize → 800, PlotRange → {0, 2000}]]
 ]
```

Figure 6. Full Function

We also attach a brief step by step summary of the function.

- Function trims the music based on its duration using mySpectro and gets the spectrogram arrays
- The Partition command partitions the large spectrogram array into smaller portions that match the length of the sample, which is two seconds.
- The music volume was normalized when the music was completely silent.
- Partitioned spectrogram arrays were subtracted from the short spectrogram array and the ListPlot was drawn to identify the seconds with the least errors. The least errors were represented with the lowest Total value of the absolute values of the differences.

Results and Discussion

Successful Cases of Matching

The algorithm was tested to see if it could identify the matching segment for identical piano pieces from the same recording. As shown in the Methods & Materials section, the clear low error values confirmed that identification worked. Figure 7 shows a clear low error at 12 seconds, which is indeed the center of the interval between 10 & 14.

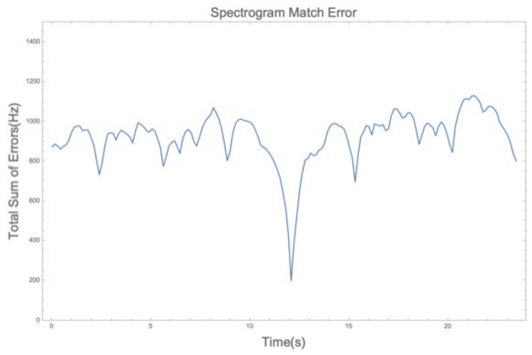


Figure 7. Errors between 10 & 14 seconds

Some Unorthodox Cases of Matching

Because *Mary Had a Little Lamb* has many parts of repetition, not all the results were as expected. Figure 8 shows the two low errors around 6 seconds and 17 seconds. This happens because the notes repeated. Indeed, we can confirm this by listening to the music. However, because our original intent in this research was matching the pieces, and the location of the match, so having multiple low error locations is beneficial for verifying that the match was correct. Also, the symmetry that happens at 6 seconds and at 17 seconds are very interesting too.

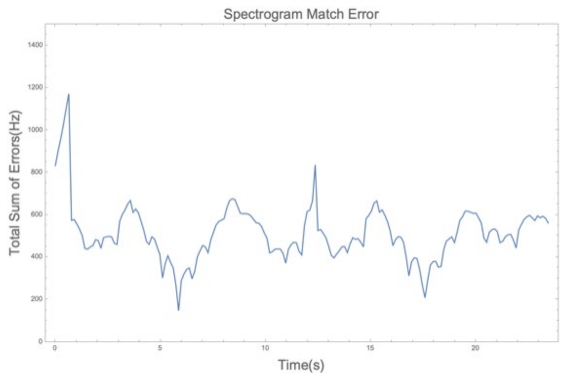


Figure 8. Errors between 16 & 18 seconds

Matching When Mary Had a Little Lamb Was Played By Two Different Recordings

This test showed that even when the same piece is played by two different players, the algorithm still can identify the matching segment with the least error. Figure 9 shows the low error at 12 seconds, which is the center of 10 and 14. Thus, the algorithm can find the matching segment in the case when the same music is played by two different people. However, compared to tests when the same player plays identical pieces, the errors are greater. Nonetheless, the algorithm found the correct match. This type of matching is one that Shazam cannot do. More tests were conducted to verify if it indeed worked, and the test results showed clear low errors unless the segment was repeated at another time in the music.

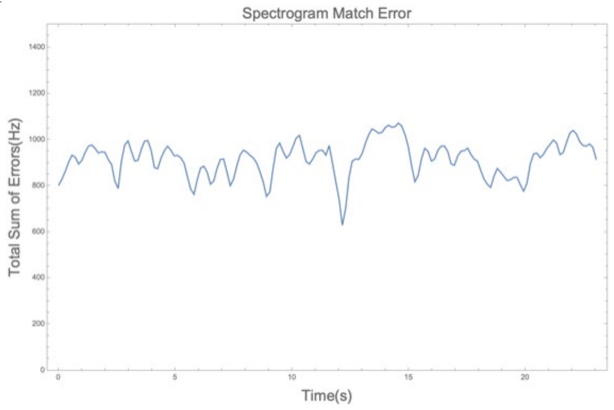


Figure 9. Matching Audios played by different people

Cello Music Matching

We experimented with the cello, a string instrument with vibrations to see how well our algorithm performed. The cello music that we used was a five note scale with breaks in the middle played by the cello. As shown in Figure 10, the algorithm was able to correctly identify the matching interval. We see a clear point with the lowest error around 11 seconds which is where the matching happens.

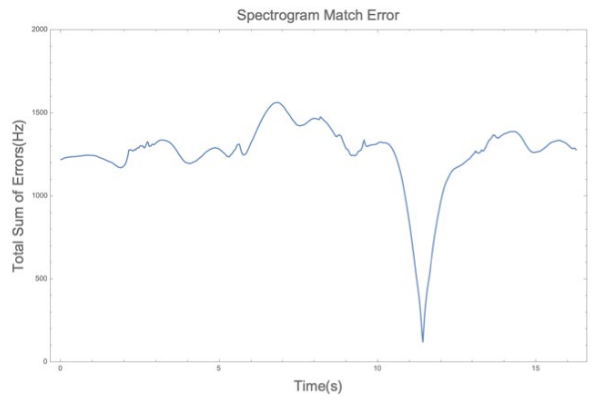


Figure 10. Cello Matching between 10 & 12 seconds

Matching A More Complex Piano Piece with The Same Recording

Now that we've seen that the algorithm performs well with simple pieces, to test the boundaries of our algorithm, we used complex piano pieces to see if the algorithm could find the match. The complex piece that we used was Chopin's Ballade 4, a very rich and complex piano piece. A 30 second segment trimmed and out of the 30 second segment another one second segment was trimmed. Figure 11 shows the low error value at 5.5, which shows that the matching was successful since the example music was chosen between 5&6 seconds. Figure 12 shows the interval of 3 & 4 seconds to confirm that the algorithm works.

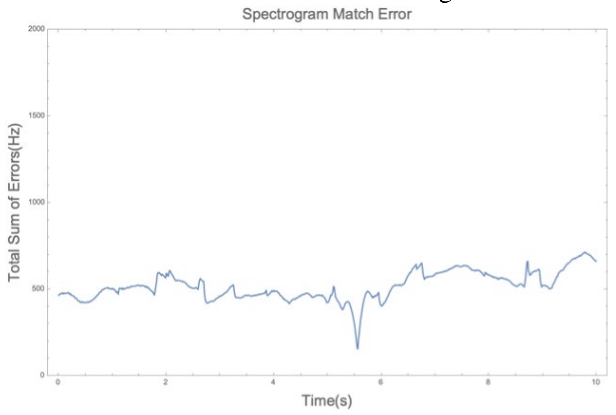


Figure 11. Chopin's Ballade 4 matching between 5 & 6 seconds

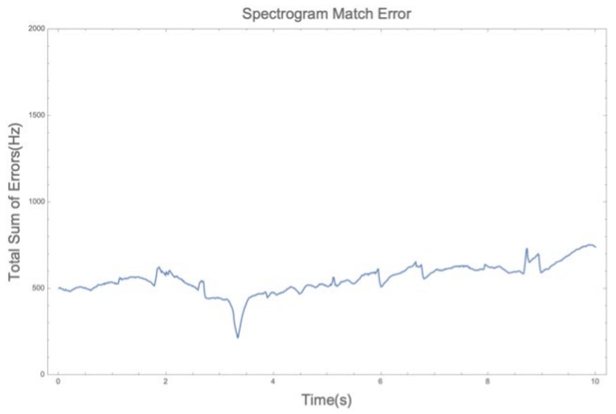


Figure 12. Chopin's Ballade 4 matching between 3 & 4 seconds

Matching Chopin’s Ballade 4 Played by Different Players

This test was done to see if the algorithm could match a complex piece played by different players. The test was conducted with Chopin’s Ballade 4. Unfortunately, the match was unsuccessful as there wasn’t a certain low error value that was evident. Also, at 8.5 seconds, which was where the matching should have been, there isn’t a clear low error value. Figure 13 illustrates this occurrence. The fact that the algorithm breaks down for complex pieces is one limitation that can be addressed with further research.

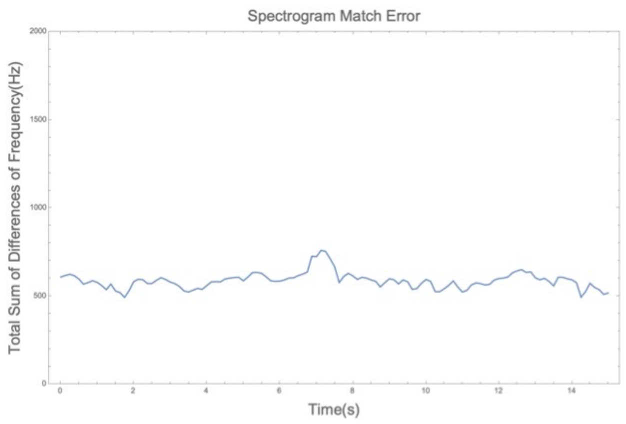


Figure 13. Chopin Ballade 4 matching played by different players

Example of A Complete Mismatch

We also provide an example of a complete mismatch for reference. The goal of this exploration was to see what the errors graph looked like when totally unrelated audios were put in. We compared Mary Had a Little Lamb to ocean sounds. As shown in Error! Reference source not found**.** 14, the graph was almost a straight line without any low error region as expected.

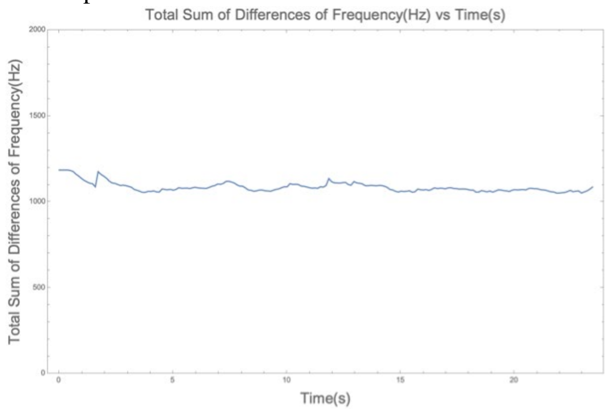


Figure 14. Errors of Ocean & Mary Had a Little Lamb

Conclusion

Humans can recognize an orchestra piece by humming or whistling the melody line. This research attempted to get closer to how humans identify music. Based on experimental results when music with only single notes is played matching was very successful. However, once single note music was played by different recordings, the success rate decreased. When a portion of a complex pieces was compared to the whole, the algorithm found where the portion came from in the timeline. However, when the portion was compared to different recordings of the same music, matching wasn't successful. For future research, the goal is to improve this algorithm by making sure it can match complex pieces played by different recordings.

Acknowledgments

I would like to thank my mentor Dr. James Choi for giving me suggestions and ideas throughout the process. His guidance allowed me to experiment in different paths. I would also like to thank my parents who have been very supportive in this research experience.

References

[1] Jovanovic, J. (2015, February 2). How does Shazam work? music recognition algorithms, fingerprinting, and processing. Toptal Engineering Blog. Retrieved January 20, 2022, from https://www.toptal.com/algorithms/shazam-it-music-processing-fingerprinting-and-recognition
[2] Christophe. (2015, August 6). How does Shazam work. Coding geek. Retrieved January 20, 2022, from http://coding-geek.com/how-shazam-works/
[3] Wolfram Research (2012), Spectrogram, Wolfram Language function, https://reference.wolfram.com/language/ref/Spectrogram.html (updated 2017).
[4] "Spectrogram." *Wikipedia*, Wikimedia Foundation, 10 Feb. 2022, https://en.wikipedia.org/wiki/Spectrogram.
[5] Wolfram Research (2012), SpectrogramArray, Wolfram Language function, https://reference.wolfram.com/language/ref/SpectrogramArray.html (updated 2017).
[6] Wolfram Research (1988), Partition, Wolfram Language function, https://reference.wolfram.com/language/ref/Partition.html (updated 2015).