Anshuman Prasad
Julian Borrey

9th October 2012

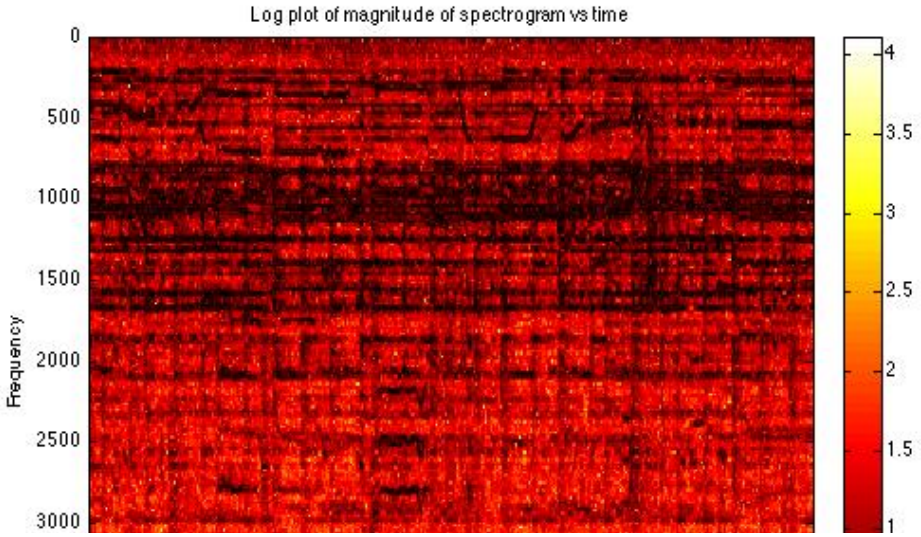
Part 1:

It is common to study the log of the magnitude of the spectrogram. Why might this be a good idea?

The spectrogram of a signal is the squared magnitude of the Short Time Fourier Transform. Therefore it is common to study the log of the spectrogram to see the STFT. Because the numbers in the spectrogram are between 0 and 1, the log of the spectrogram is negative. This necessitates taking the magnitude.

Plot the log of the magnitude of the spectrogram with axes appropriately labeled.

Try



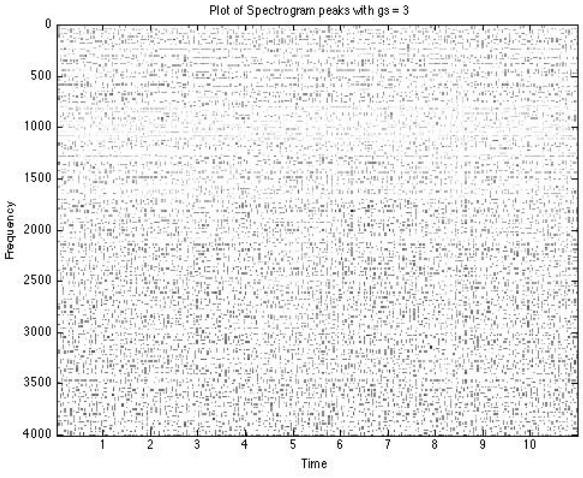


Figure 2: Constellation plot for gs = 3

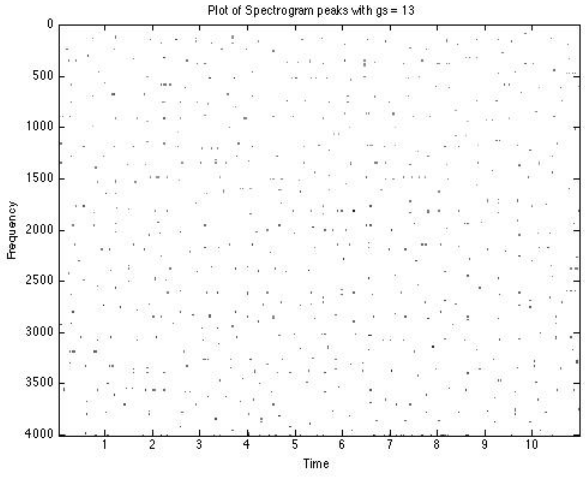


Figure 3: Constellation plot for gs = 13

several values for gs and plot the constellation map. Note the effect of

changing gs.
The larger gs is, the longer time it will take to run the code and find the local maximums, but it will find fewer local maximums. The variable gs stores how large a section should be checked to find the local maximum. A larger gs corresponds to a larger area so more has to be compared which is why it takes longer. There will be fewer local maximums because there can only be one local maximum per section and when the section is larger there are fewer sections.

gs	Number of peaks	Threshold magnitude	Number of peaks above threshold
3	15145	2.4754	329
5	5300	2.4632	329
9	1459	2.4108	329
13	677	2.3220	329

Table 1: Table showing how the number of peaks,

Compute the constellation map for gs=9, i.e. 4 points in each direction. Calculate how many peaks there are and record your answer below. How many peaks are there per second on average?
The number of peaks for the clip 'viva.mp3' was recorded to be 1459 leading to 132 peaks per second for the clip.

We want to use only the larger peaks. Why? (Hint: think about the quality of the clip we would like to identify)?
We only want to use larger peaks because the larger peaks are more unique which means that you are more likely to find a unique match instead of several possibilities.

Record the threshold value below along with the number of peaks kept.

gs	Number of peaks	Threshold magnitude	Number of peaks above threshold
3	15145	2.4754	329
5	5300	2.4632	329
9	1459	2.4108	329
13	677	2.3220	329

Again, display the constellation. Comment on the distribution of peaks. Is it uniform? Are they closely packed? If so, is this a good thing?

Most of the peaks above the threshold value are distributed towards the bottom of the image which corresponds to high frequencies, but they are even across time. They are not nearly so closely packed as they originally were. It is a good thing that they are high values because that should make it easier to identify unique matches.

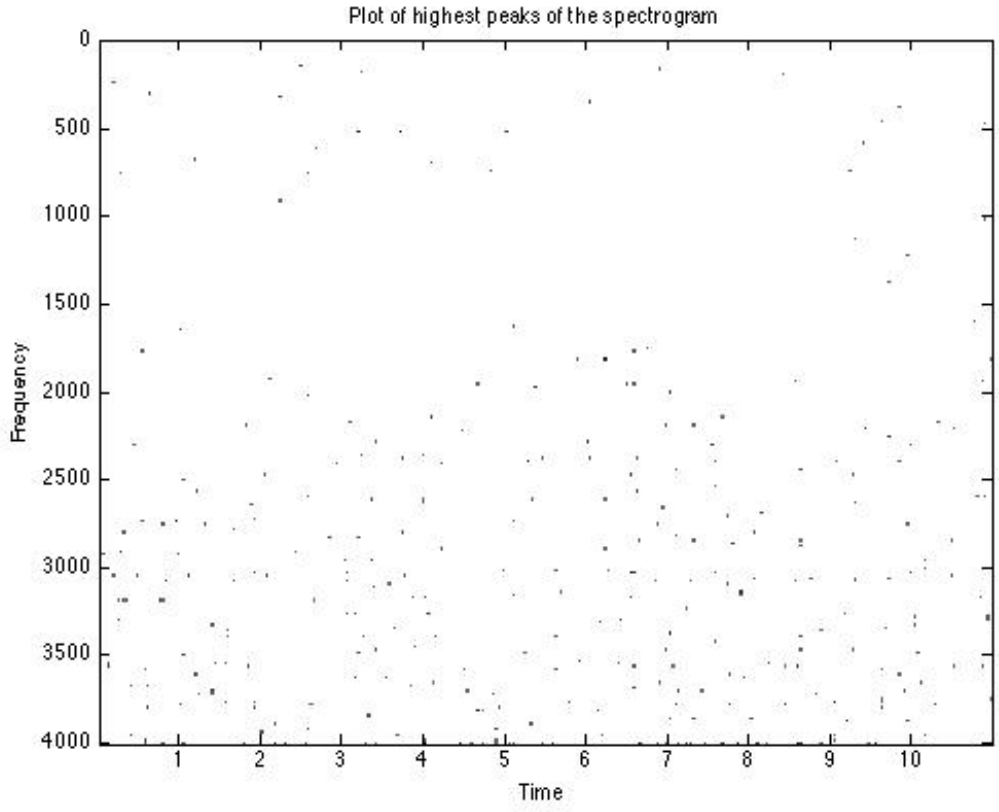
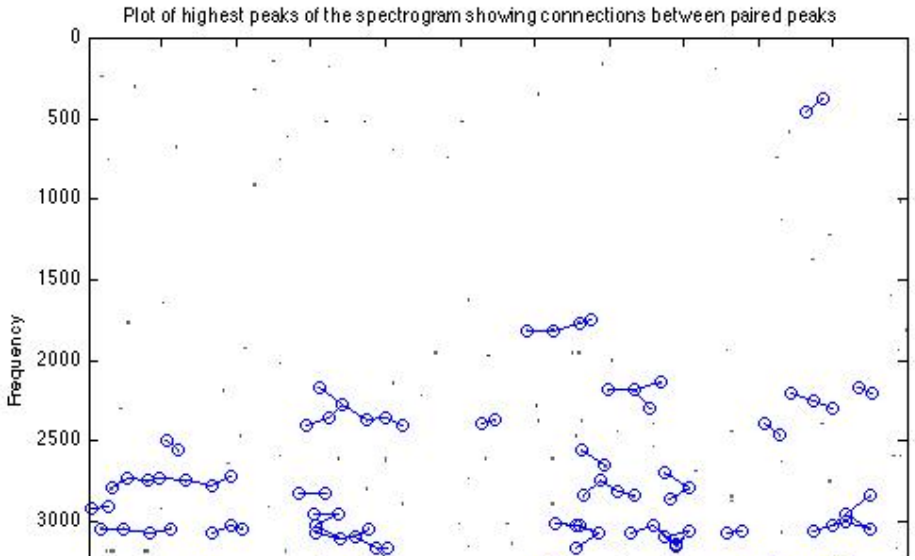


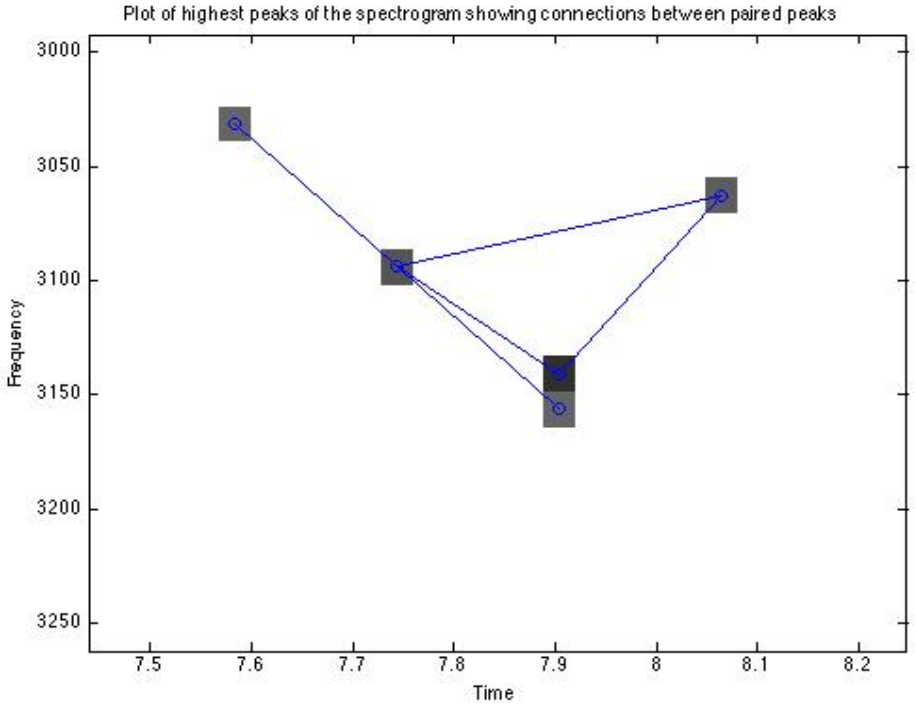
Figure 4: Constellation plot for 'viva.mp3' after thresholding

If we were concerned about having peaks that were too close together, how could we threshold so that we guarantee some separation between peaks? Concretely, outline a basic algorithm which would accomplish this (you don't have to code this).

Before running the basic threshold algorithm which leaves only the largest peaks and results in 30 peaks per second, the peaks should be sorted by computing the difference between the magnitude of adjacent peaks in a sorted vector and having a set difference below which the second peak is eliminated. Then manipulating the number of peaks to result in a 30 peak per second average will have dealt with the problem of having too many peaks.

After you have produced your selection of pairs, display the constellation map and connect the pairs listed in the table with line segments.





Experiment with changing the fan-out for each peak and with changing lt, ut,

Figure 6: Zoomed in Plot showing the multiple connctions that can be made with a fanout value greater than 1

and f. Also try different lengths for window in the spectrogram. Note any significant findings.

As shown in Table 2, as fan-out increases the number of pairs found increases because each
point can be paired with more points. As lt increases the number of pairs decreases, and as ut increases the number of pairs increases. This is because the width of the box decreases and increases respectively. As f increases the number of pairs increases as the height of the box increases allowing more peaks to be considered.

fanout	lt	ut	f	number of pairs
1	5	11	5	125
3	5	11	5	165
5	5	11	5	167
3	3	11	5	181
3	5	11	5	165
3	7	11	5	114
3	5	9	5	125
3	5	11	5	165
3	5	13	5	215

3	5	11	3	125
3	5	11	5	165
3	5	11	7	209

Table 2: Table showing how the number of pairs changes with change in parameters

Part 2

How big does your hash table need to be?

The number of rows in the hash table will be equal to the highest hashvalue generated while creating the database. The hashvalue is generated by the following equation:

$$h(f1, f2, t2c\text{-}t1c)=(t2c\text{-}t1c)*2^{16}+f1*2^{8}+f2$$

This means that the highest hashvalue will most likely occur when the time between the peak frequencies is the greatest. The number of columns in the hash table will be equivalent to the maximum number of collisions that occur at any of the hashvalues. (Note: the hash function had to be floored in the code so that hashvalue would be a positive integer value since the values in the columns of the table for each clip/song

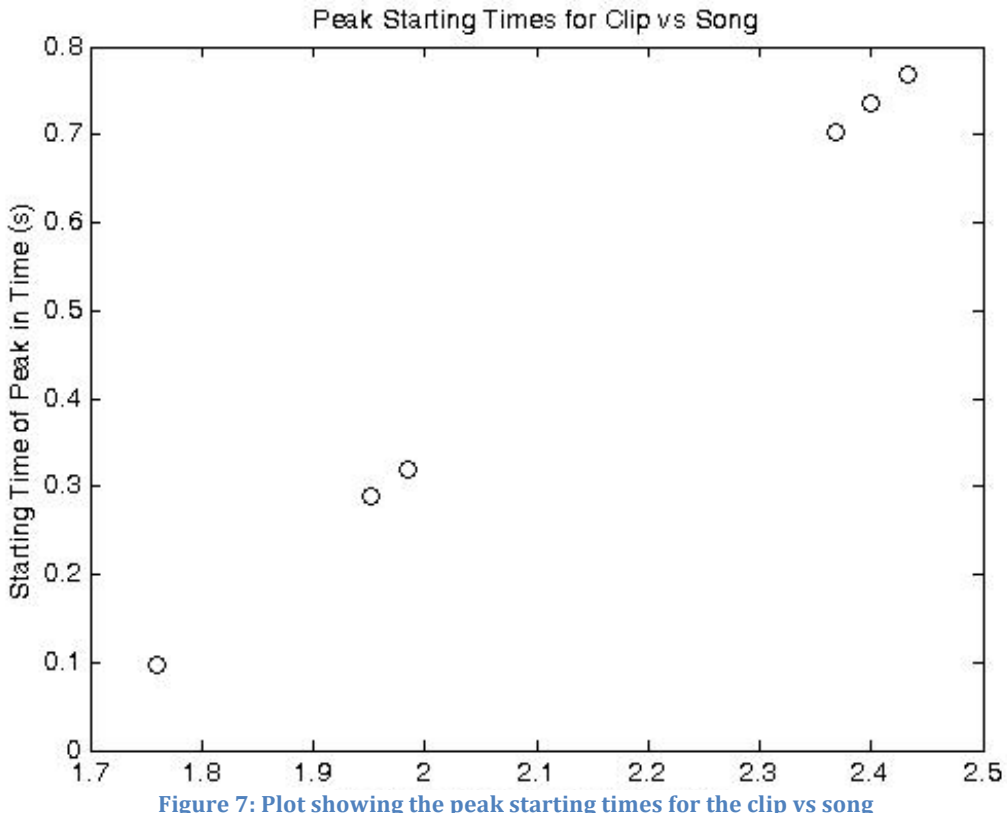


Figure 7: Plot showing the peak starting times for the clip vs song

were not integers. The hashvalue has to be a positive integer because it is used as a row index in the hashtable)

How does the distribution of peaks together with your method of choosing the peak pairs from within the target box affect the number of collisions?
If the peaks are too close together resulting in similar frequency and time difference values, the hashvalues could possibly be the same thus resulting in collisions. This can be addressed while making the table by expanding the range of the target window. The best way to do this would be to expand the deltaf parameter which would allow more peaks with different frequency values to be paired resulting in less collisions.

Have the program draw the scatter plot of the t^c_1 vs. t^s_1 for the matching song.

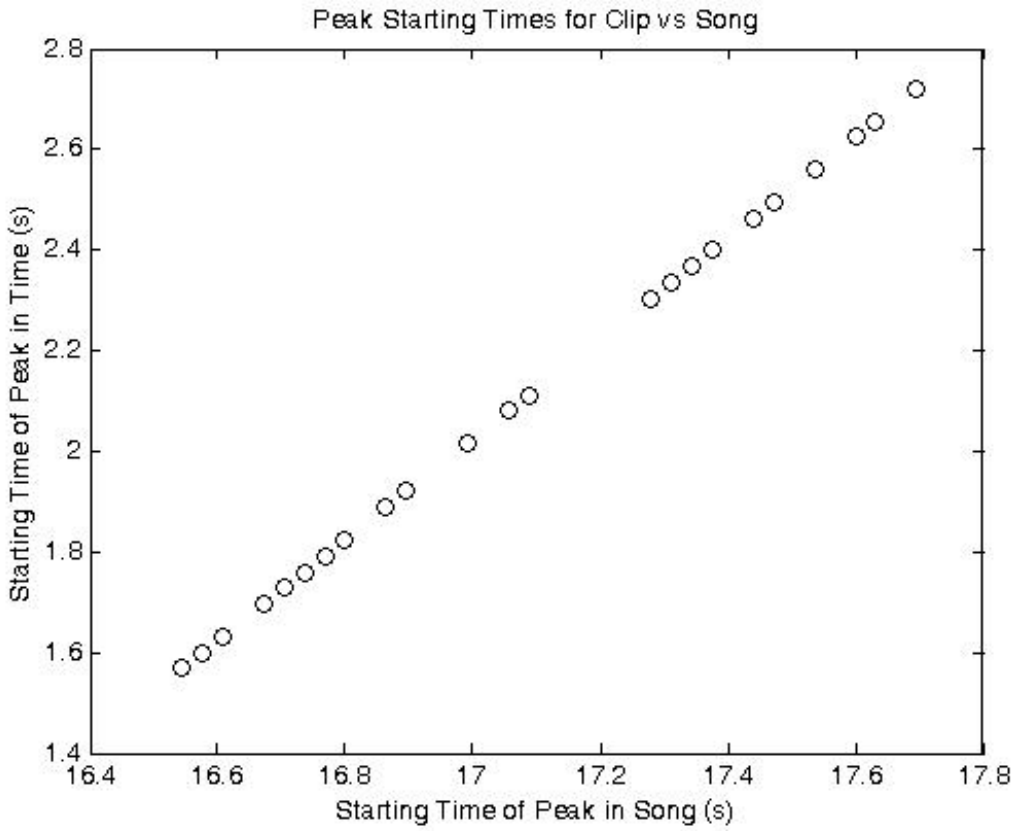


Figure 8: Plot showing the peak starting times for the clip vs song

How could you measure confidence in a classification?
Confidence in a classification could be measured by performing a linear fit through the graph of t^c_1 vs. t^s_1 and calculating the r^2 value. The higher the r^2 value, the higher the confidence in the classification.
Currently our code only plots t^c vs t^s when t^s-t^c is exactly the most frequent t_0 which guarantees a perfect correlation using r^2 values, so another way to measure confidence in a fit would be to count the number of points on the graph. In this case, a higher number of points would signify a higher confidence in classification. If there

are two points or less, there should be little or no confidence in classification.

Run the matching function on multiple clips from each song in the database and record your correct classification rate.

Song	**# correct classifications**	**Runs**	**Correct classification rate**
Johanna from Sweeney Todd	3	3	100
Theme from Zelda	3	3	100
Green Finch and Linnet Bird	3	3	100

Plot the percentage of correct classifications vs. SNR with clips of length 5, 10, and 15 seconds.

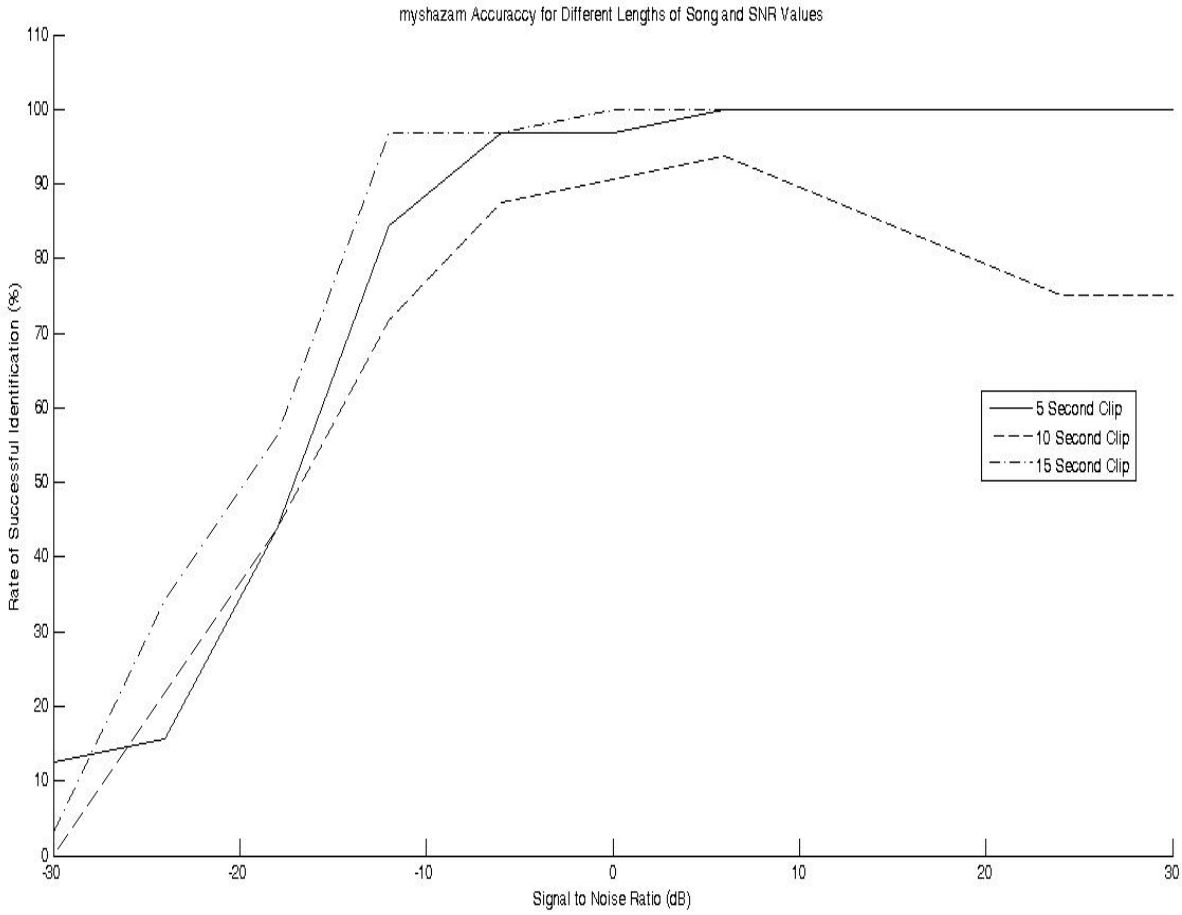


Figure 7: Plot showing the classification rate for 5,10 and 15 second clips for varying levels of noise

The signal to noise ratio of the *myshazam* algorithm was tested with different levels of white Gaussian noise added to the input signal. The noise is generated via a random number generator inbuilt in MATLAB. A 4 song data base was made using 4 tracks from the musical *Sweeny Todd.* Each song was then clipped at a random intervals to produce a 5, 10 and 15 second clip of each song. Each song was passed through 8 times at varying signal-to-noise ratio (SNR) values from 30 dB to -30 dB. The plots show the success rate of trials with different quantities of noise for each clip length. As expected, when the SNR was large the probability of identifying the correct song was lower. The drop in success rate occurred at ~ 5 dB SNR.

How does the length of the clip affect the correct classification rate?
Clips of longer length have more 4-tuplets to exhibit and so have a higher chance of being identified correctly. We therefore expect longer clips to have a higher resilience to noise than shorter clips. This trend is observed between the 15

second and 5 second clip however there is a very strange phenomenon that occurred for the 10 second clip. This length seemed to perform the worst. This is likely to be an anomaly due to our relatively small sample space. With more songs in the data base and a higher number of trials (this test was 32 trials per SNR increment) this error should vanish.

Test your program on a bunch of clips corrupted with speech or some other form of noise, how does it perform?

White Gaussian noise is a good way to test for absolutely random distortion but the real application of such an algorithm is to identify music in the physical world. This would almost always involve using a microphone to obtain the signal of the music. Furthermore it is likely that song identification would be craved for in a social setting where the noise of speaking and even other music is present. To simulate how well the algorithm works under these conditions sound was recorded at au bon pain in the Brian Center, Duke University West Campus. This noise involved background music and talking. When this sound was added to the signal of a 5 second clip recognition failed. After using a scaling factor *myshazam* was able to recognize the song. For this particular piece of moderate level noise a scaling factor of just 0.91 was needed to get the algorithm to succeed. For a longer clip of 15 the algorithm, as expected, was much better at recognizing the correct clip with added natural noise. The two plots below show the signal of a 15 second clip with and with out noise. It was found that a scaling factor of $\sim$2.5 was the limit of recognition.

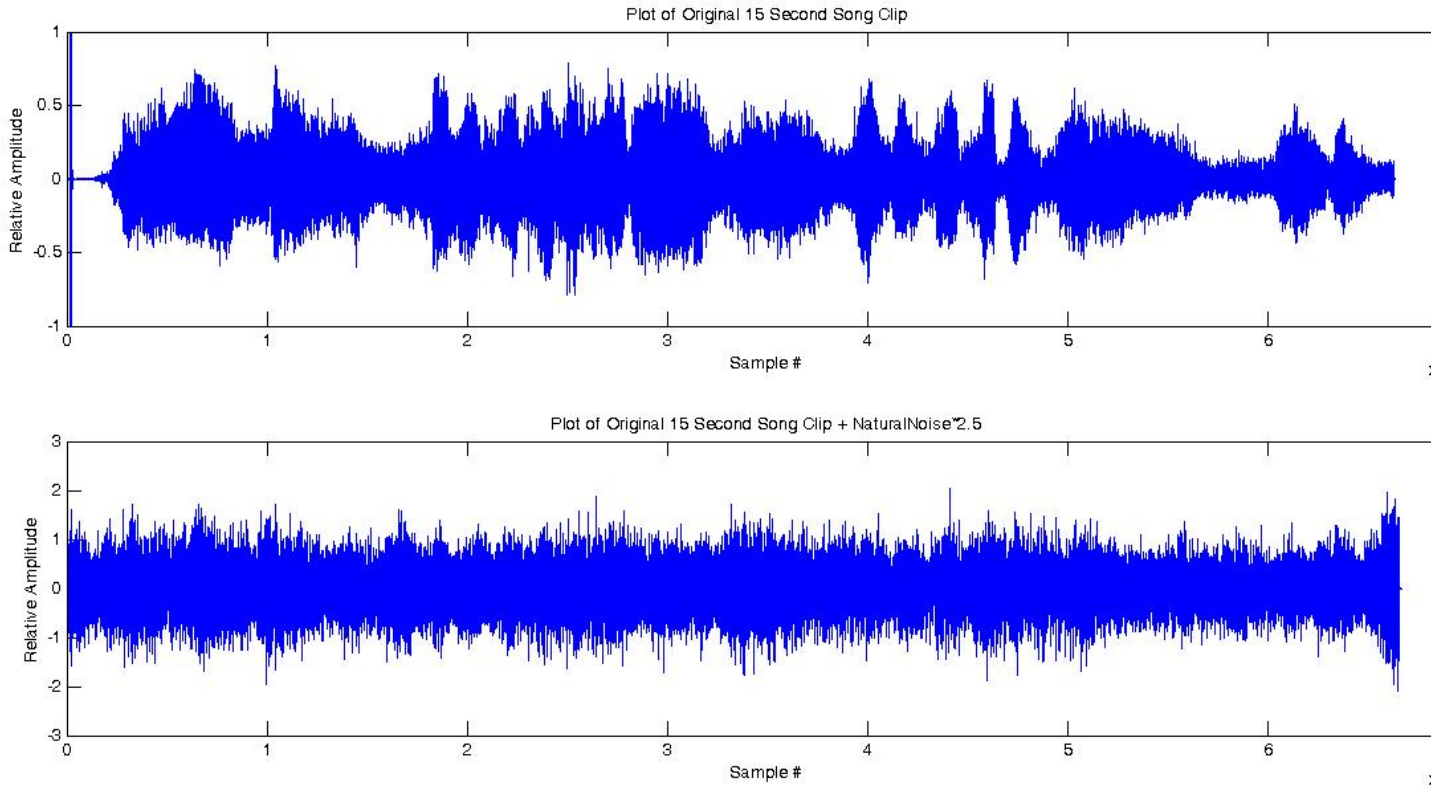


Figure 8: Plot showing how noise changes the original signal

A real scenario test of using the myshazam function was done by recording the playing of one of the songs in the database from an iphone. The recording of 13 seconds, which included just noise for one second on either side of the song playing, was enough to allow for correct identification. When the recording was reduced to just 8 seconds long however the algorithm failed.

Appendix – A (Code for the Adaptive Threshold)

One problem faced when matching clips with their songs was that to reach a 30 peak per second average, the peak distribution could be extremely non-uniform

throughout the clip. This occurs in clips where the highest peaks are concentrated together as shown in Figure 9. The adaptive filter breaks down the clip into one second chunks and finds individual threshold values for each chunk such that there is a 30 peak per second average in each individual chunk. This leads to a more uniform distribution of peaks which will make it easier to correlate to a thresholded table of the entire song.

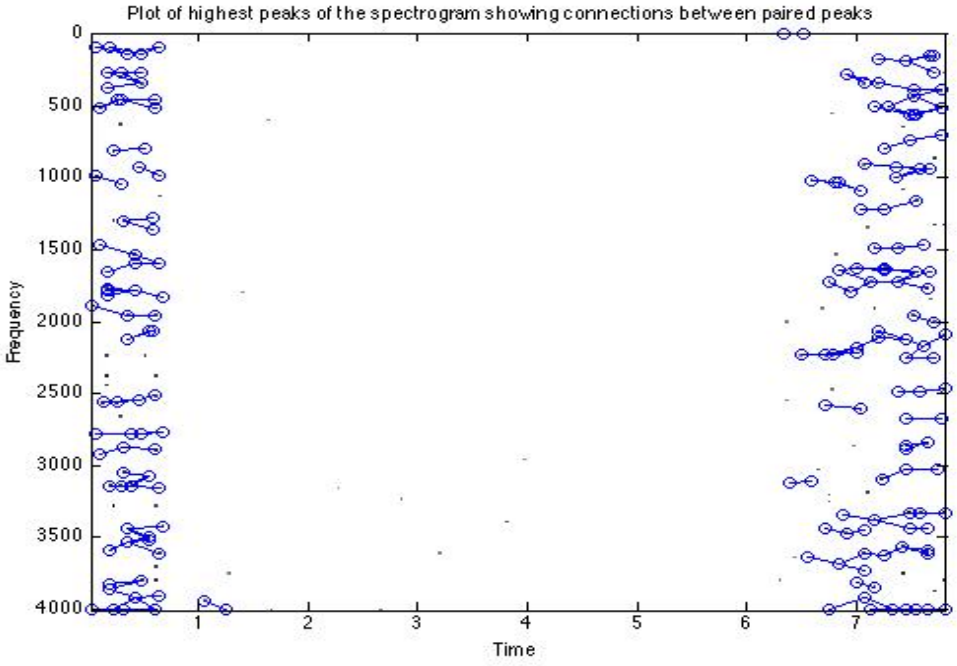


Figure 9: Constellation plot for a sample clip without the adaptive filter

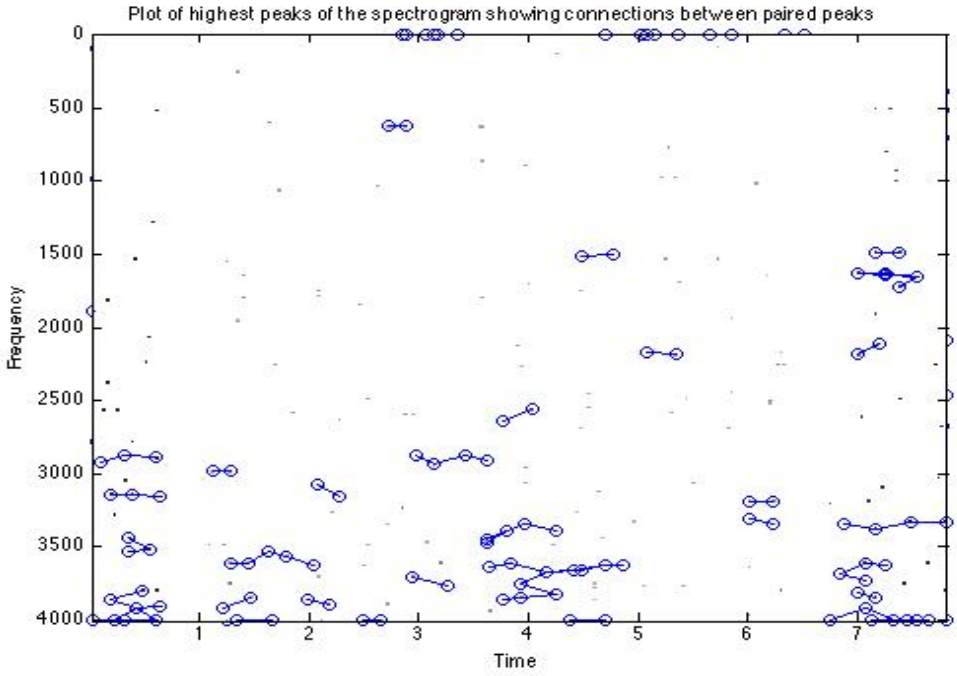


Figure 10: Constellation plot for the same clip with the adaptive filter