**Research on the Components of Speech Recognition**

by Howard Zhang

**Objective**

In ninth grade, I began programming with the goal of learning more about artificial intelligence. One of the fields in AI that I was most interested with was speech recognition. Initially I used built-in speech recognition systems in order to implement it into the many other projects that I was working on in parallel, projects such as door alarm systems, calendar notification systems, messenger robots, and so on. But, eventually, I wanted to stop reusing completed speech recognition APIs in an effort to build my own. So, I started this research project to understand the internal engineering, mathematics, and physics that went into speech recognition.

**Methods (Project Duration and Role)**

I began the project in my junior year of high school and it has lasted to the end of my senior year. This project has been done mostly independently. However, starting in my senior year, my former computer science teacher Mr. Mauro has provided me guidance regarding the component of speech recognition known as Dynamic Time Warping.

I initially conducted my research by finding the most common forms of speech recognition currently used and how they were created. After finding the main factors that went into the creation of speech recognition (Dynamic Time Warping, Mel-Frequency Cepstrum Coefficients, Fourier Transforms), I researched the scientific and mathematical theory that they were based on. With that, I further researched and formed a programmable understanding of the theories in order to code them myself to complete a usable speech recognition program.

**Timeline**

**Step 1: Working with Speech Recognition APIs (CMUSphinx)**

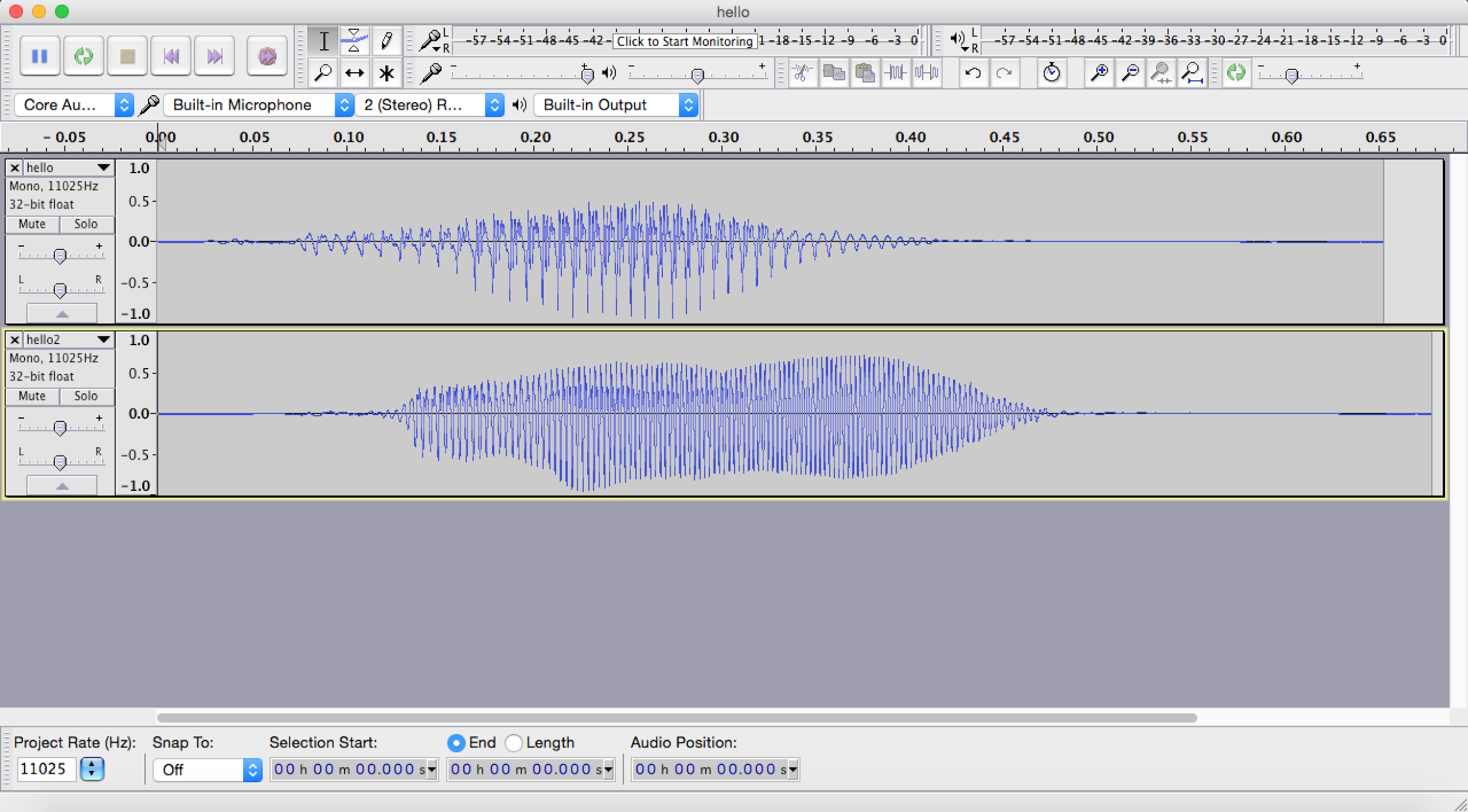
My first foray into speech recognition was through the downloadable speech recognition API known as CMU Sphinx. I used it as a main component of my home automation program. This introduction to speech recognition taught me the basics of the Java Sound API (recording sounds, working with sound signals in Java). I also learned the basic components to a full-fledged speech recognition system, such as dictionaries that map a speech signal to a word, language models to make sense of the grammar of conjoined words, and acoustic models that worked alongside the dictionary to determine the word being spoken. I learned that speech recognition systems compared speech signals to the signals stored in dictionaries, which then mapped to the word that was spoken. To increase the accuracy rate, I limited the dictionary to only a few key phrases that would be used for user interaction with the program. However, as I continued using it, I became more and more fascinated with the internal workings of a speech recognition program, and set out to understand the more intricate details of its creation, focusing on the acoustic model section of speech recognition, which included finding a proper method of comparison between two speech signals.

**Step 2: Understanding the Physics of Sound**

Excited by the prospects of creating my own Speech Recognition API paralleling what I learned from Step 1, I began researching the physics of sound. Borrowing from what I knew from my physics classes and after extensive research online, I found that sound, being a longitudinal wave, had and could be described by the frequencies that made it up, as well as the amplitudes of the wave itself. However, I also learned that sound waves in the real world were messy, being affected by noise, that also factored into the creation of the wave itself. Nevertheless, I knew that my next step would be using a programmable method to find these measures.

**Step 3: Visualizing the Comparison (Audacity)**

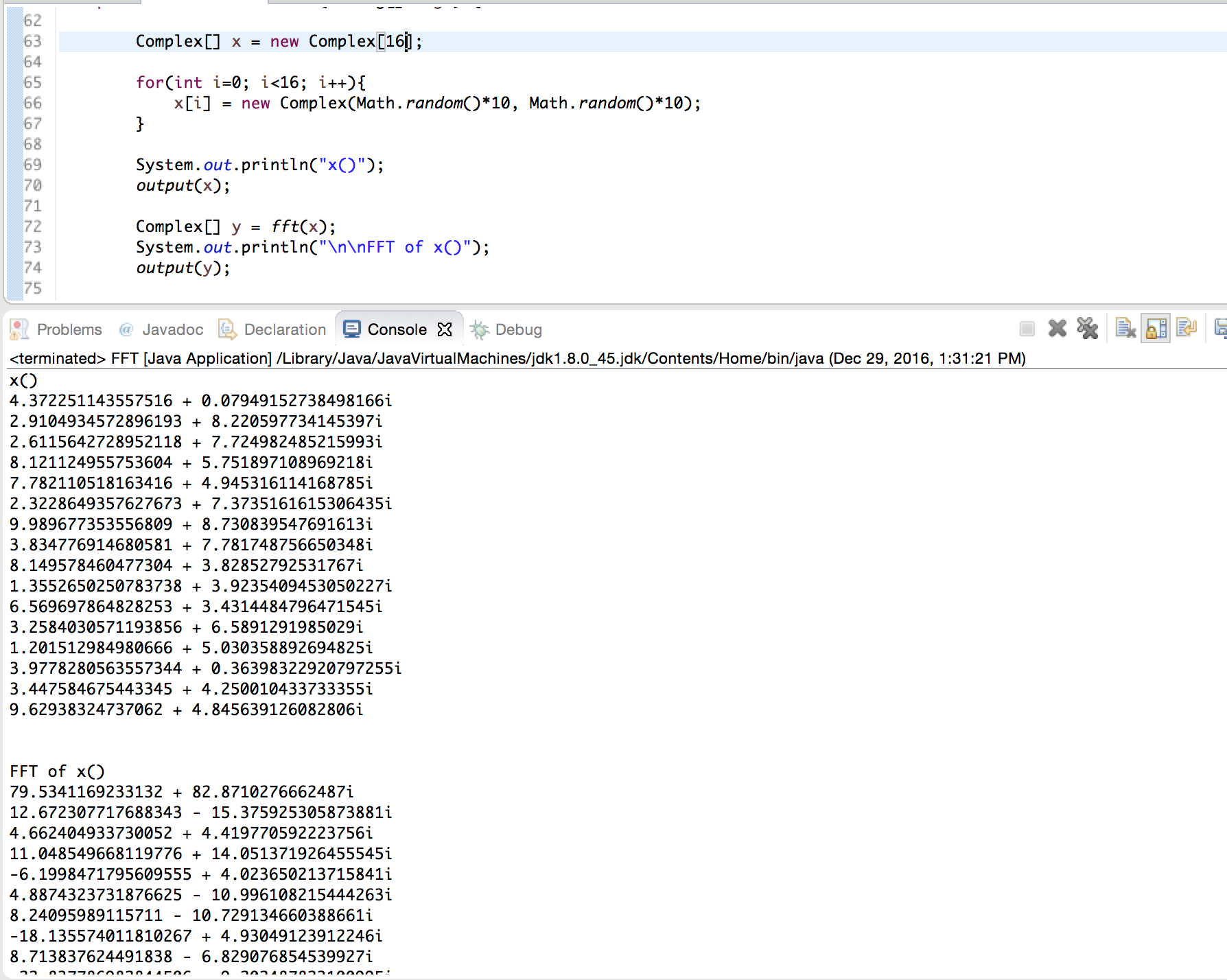
After learning about the physics of sound from Step 2, I wanted a visual depiction of the sound signals that I researched to see what I was working with, so I downloaded Audacity, which allowed me to “see” the graphs of the speech signals that were being compared, visually taking in the amplitudes, frequencies, and noise that made up the waves, but as I moved forward, I knew that I needed a programmable way to analyze the speech signals.



Using Audacity to visually compare two audio samples before analysis (note that this is a perfect example of two series requiring Dynamic Time Warping for comparison, which would better align the two speech signals for a more accurate comparison, which is explained later).

**Step 4: Delving into Signal Analysis**

I began the process of programmable speech recognition after visually depicting the signals from Step 3. I knew that at heart, speech recognition came down to signal analysis. Researching heavily into the mathematical theories behind signal analysis, I came upon the Fourier Transform, which I learned would provide me a way to break down a sound wave into its component frequencies. However, I knew that true speech recognition did not take into consideration every part of the sound wave. Remembering the noise portion of the sound wave from before, I searched up a method of feature extraction that would find the important parts of the sound frequencies for analysis. This method of feature extraction would be finding the Mel-Frequency Cepstrum Coefficients (MFCCs) of the sound wave.



Coding the Fourier Transform Theorem equation in Java and testing it with a random array of Complex numbers.

**Concept: Fourier Transform Theorem**

The Fourier Transform Theorem is the basis of a majority of signal analysis, and is therefore a major component to the creation of speech recognition systems. Essentially, the Fourier Transform Theorem takes a signal (in this case, a sound signal), and returns the frequencies that make up the signal. For sound signal analysis, it transforms the a time-amplitude graph to be on the frequency domain, which is extremely useful in the next process described.

Programming the Fourier Transform is simply implementing its equation in code:

(F(y)) = summation from x=0 to n-1 of(f(x) \*e^(-2\*pi\*i\*x\*y/n))

Where:

F = output vector

f = input vector

x = input index

y = output index

n = size of input vector

**Concept: Mel-Frequency Cepstrum Coefficients**

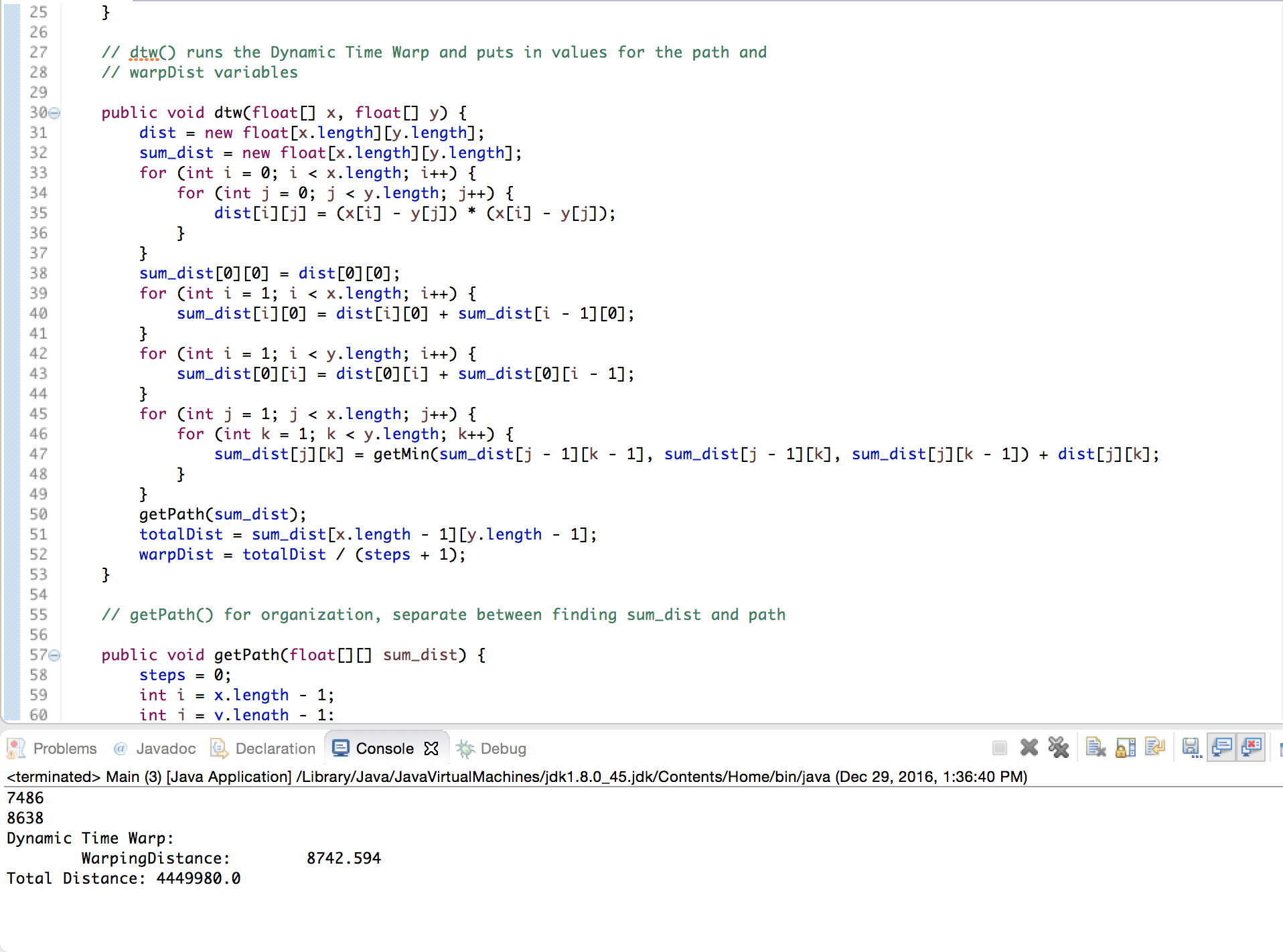
Finding the Mel-Frequency Cepstrum Coefficients or MFCCs of a sound is a method of feature extraction for speech signals. Essentially, it takes a speech signal and picks the most important parts for further evaluation.

The major programming steps to finding the MFCCs of a speech signal are listed below:

1. Find the Fourier Transform of the signal being analyzed.
2. Take the resulting powers of the spectrum and map them on the mel scale. This is a nonlinear scale that mimics the perception of sounds in the human ear, which takes in separate frequencies in a nonlinear fashion.
3. Use a logarithm to find the powers of the frequencies. This again mimics the perception of sounds in the human ear, as humans hear differences in sounds on a logarithmic scale.
4. Use the inverse of the Fourier Transform to transform it back into a signal. The resulting amplitudes are the MFCCs that will be further analyzed.

**Step 5: A Method of Comparison**

Now that I had a set of data ready for analysis from Step 4(the MFCCs of two speech signals), I had to decide on how they would be compared. My former computer science teacher would guide me to the right direction, informing me about a method of speech comparison known as Dynamic Time Warping. Researching and learning the computer science algorithm behind that method finally allowed me to compare the MFCCs that I extracted from the two sound waves (one that was spoken, and one in the dictionary) for a value that I could compare between different sound waves in the database to find the correct word spoken.



Coding the Dynamic Time Warping Algorithm to compare the 2 audio signals.

**Concept: Dynamic Time Warping**

It is usually hard to compare two time domain series effectively, as the time of certain features in the series usually misalign. Dynamic Time Warping essentially warps the time domain of the two series until it finds an optimum comparison. Though this process has been replaced in recent years by Hidden Markov Models, it is still an extremely efficient and useful algorithm that can accomplish speech recognition.

The major programming steps to using Dynamic Time Warping for two speech signals are listed below:

1. For two time series, x and y, find the euclidean distance between all pairs of x and y, and store it in a 2D matrix.
2. Find the accumulated distances in every single “point” of the matrix.
3. Use backtracking to find the path while only moving forwards (upwards or to the right) that leads to the minimum accumulated distance.
4. The resulting minimum accumulated distance is the measure that will be compared to match speech signals.

**Results and Final Thoughts**

Although the project is still currently being worked on, I now have a completed speech recognition program on my computer. In the process, I have researched extensively on the main components that make up speech recognition systems and found that many of these algorithms seek to mimic the processes used to analyze sounds in the human ear. For example, such processes as the Fourier Transform take an amplitude graph of sound and transform it to display frequencies instead. In the human ear, a speech signal is judged based on its component frequencies, not amplitudes. Furthermore, such examples as the logarithmic mel scale closely parallels the human ear, which does not hear differences in sound linearly, but rather logarithmically.

**Future Research**

Although I do have a working speech recognition system, there are still several different factors that need to be improved in order to allow the program to work at industry standards. For one, while MFCC and Fourier Transforming are essential to the comparison of speech signals, Dynamic Time Warping is relatively older and has since been replaced with the use of Hidden Markov Models.

Other factors can also be applied to improve the system, such as the use of cepstral mean normalization, where the cepstral means are subtracted from the MFCCs to account for variation in sound input.

Additionally, this project focused more on the process of analysis between two speech signals, but the entire process of speech recognition requires more than just that. This project did not create a big enough dictionary of speech signals to focus on such things as efficient mapping between the spoken sound signal and the database or other factors in that field of speech recognition. For example, the program could compare two users saying the word “Hello” and understand that the users said the same word through speech signal comparison, but would not be able to understand exactly every word the user said (because it does not have a database that stores a dictionary of speech signals mapped with the words those signals represented). So, the next big step to perfecting a speech recognition system would be the construction of a dictionary that would map speech signals to words, essentially, “teaching” the computers new words to broaden the range of words that the program can interpret. However, as the computer’s vocabulary becomes larger and more varied, a different field of artificial intelligence is necessary, language comprehension through machine learning is the logical next major step for this project.

**Works Cited**

Batra, Nipun. “Programatically Understanding Dynamic Time Warping | Nipun Batra |.” *Nipun Batra Atom*, 31 July 2014, nipunbatra.github.io/2014/07/dtw/.

Bevelacqua, Pete. “Fourier Transforms.” *Fourier Transform*, www.thefouriertransform.com/.

“The Discrete Fourier Transform.” *The Discrete Fourier Transform*, 2001, pp. 31–60. doi:10.1142/9789812810298\_0003.

“DTW Algorithm.” *GenTXWarper - Mining Gene Expression Time Series*, Computational Biology, www.psb.ugent.be/cbd/papers/gentxwarper/DTWalgorithm.htm.

Eddy, Sean R. “What Is a Hidden Markov Model.” *Nature.com*, Macmillan Publishers, 2004, www.nature.com/nbt/journal/v22/n10/full/nbt1004-1315.html.

“How to Implement the Discrete Fourier Transform.” Project Nayuki, www.nayuki.io/page/how-to-implement-the-discrete-fourier-transform.

“An Interactive Guide To The Fourier Transform.” *BetterExplained*, betterexplained.com/articles/an-interactive-guide-to-the-fourier-transform/.

Lama, Palden, and Mounika Namburu. “Speech Recognition with Dynamic Time Warping Using Matlab.” .

Lutter, Michael. “Mel-Frequency Cepstral Coefficients.” *The Speech Recognition Wiki*, 25 Nov. 2014, recognize-speech.com/feature-extraction/mfcc.

“Mel Frequency Cepstral Coefficient (MFCC) Tutorial.” *Practical Cryptography*, practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/.

Ratanamahatana, Chotirat Ann, and Eamonn Keogh. “Everything You Know About Dynamic Time Warping Is Wrong.” doi:10.5040/9781472920430.0010.