APPM 3310 Project

Discrete Fourier Transform for

Complex Wave Analysis

By:

Damien Beecroft

Spas Angelov

Eli Johnson

David Kopala

**Abstract**

Fourier analysis is an immensely important tool in the field of mathematics. It is used in a wide spectrum of applications, from noise filtering to data compression. Specifically, this paper will explore the theory, efficiency, and application of the Discrete Fourier Transform (DFT) and Fast Fourier Transform (FFT) in music decomposition and noise filtering. Along with this report, the group produced a Matlab program that can take in a composite sound wave and output the frequencies that compose the wave. The performance from this group’s implementation was compared to the Matlab function, and it was found that the custom solution, while accurate, was worse.

**Attribution**

Eli developed the original Fourier Matrix Code, created the appendix, formatted the writeup, and did additional research.

Spas researched the Radix-2 FFT method along with developing the Matlab code and testing certain generated inputs.

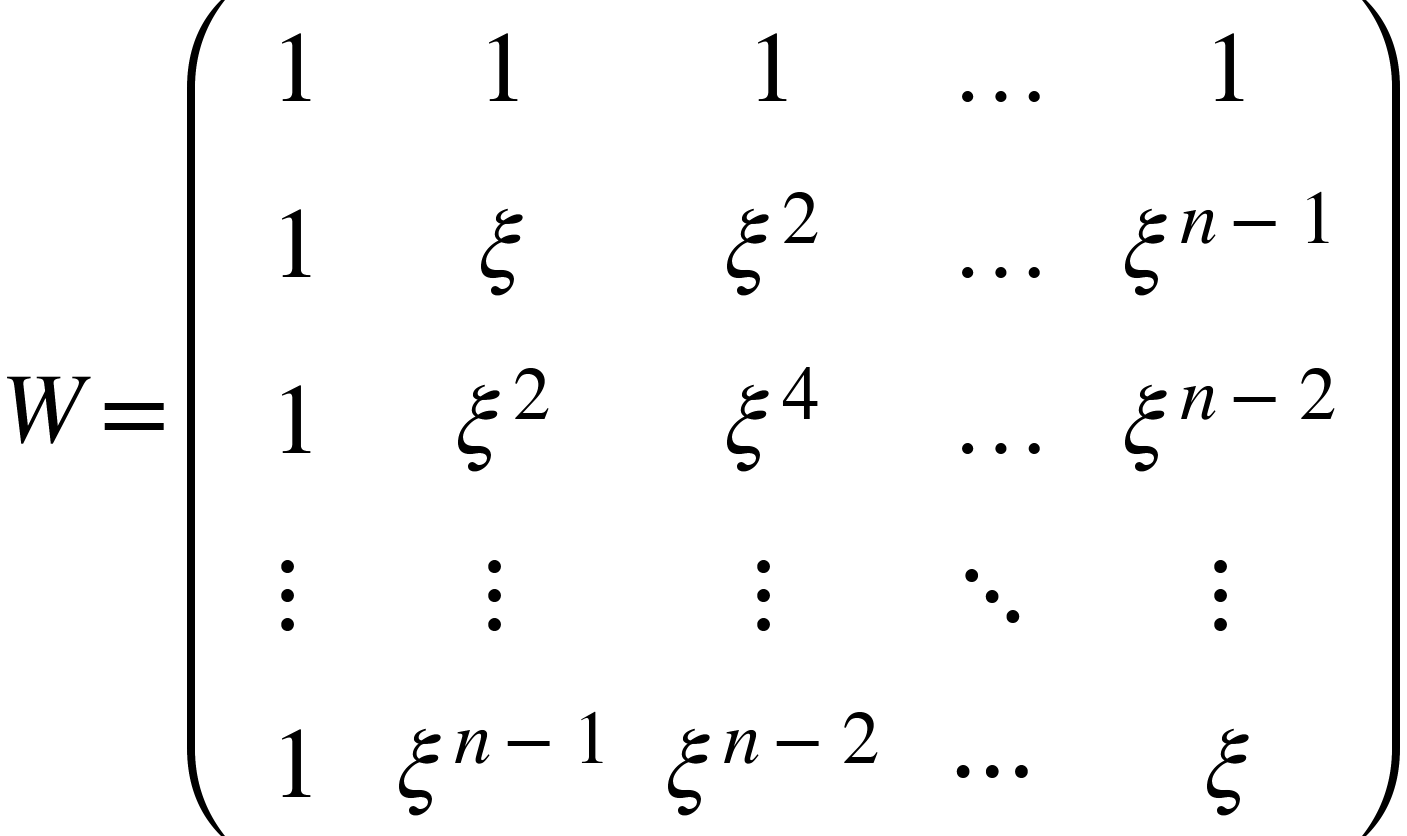
Damien did the abstract, most of the introduction and conclusion, and aided in developing algorithms for the Matlab code.

David did the performance analysis and compared the Radix-2 FFT performance to the Matlab implementation.

**Introduction**

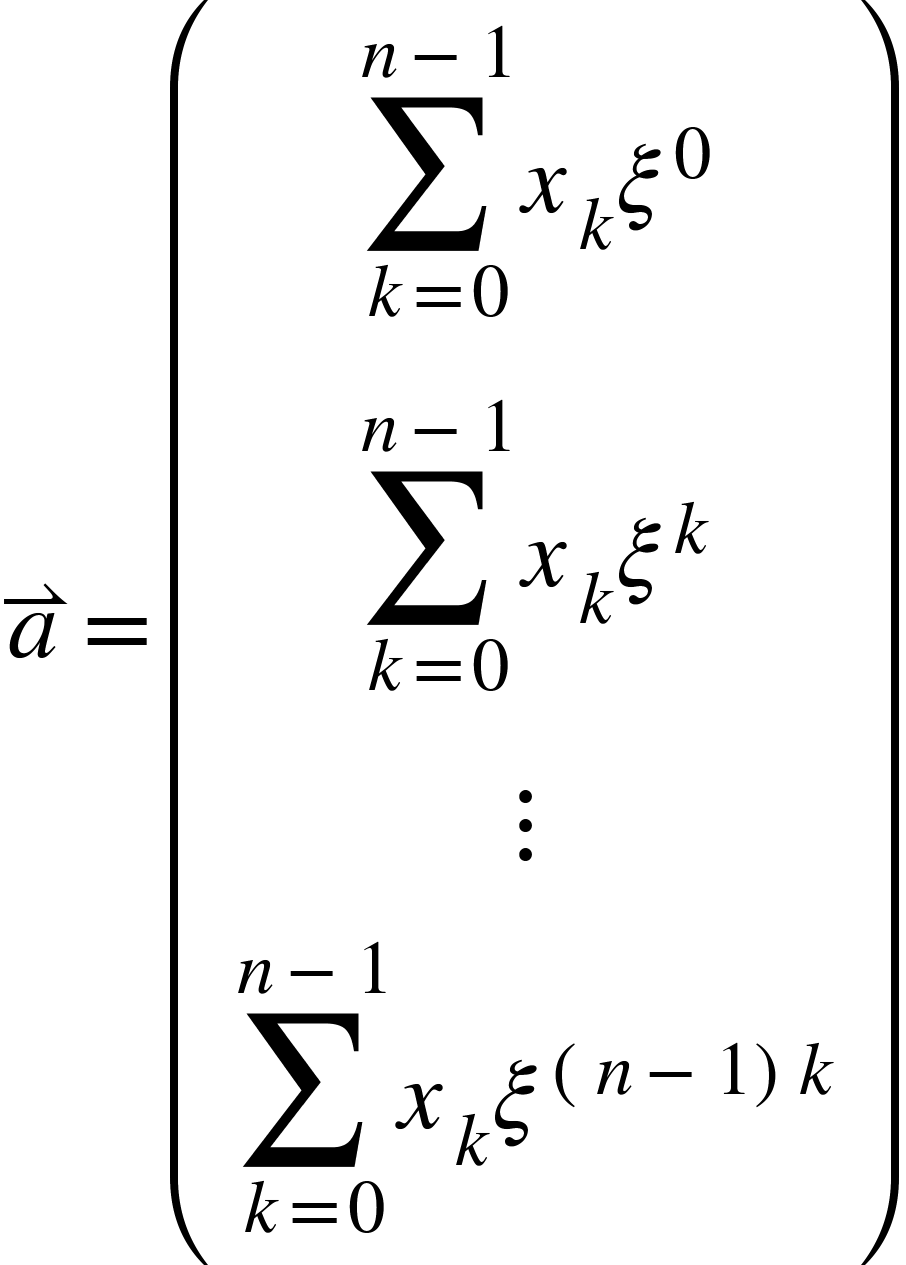
The DFT and FFT take in a vector of measurements, *x*, corresponding to the amplitude of the input wave at certain time intervals as opposed to a continuous wave. Then, this vector is multiplied by the Fourier Matrix, *W*, which maps the vector *x* to a space of frequencies corresponding to the basic waves that compose the input wave.[[1]](#footnote-0)

The rows of the Fourier Matrix correspond to sample amplitudes of simple waves of varying frequencies taken at the same time intervals as the input waves.[[2]](#footnote-1)



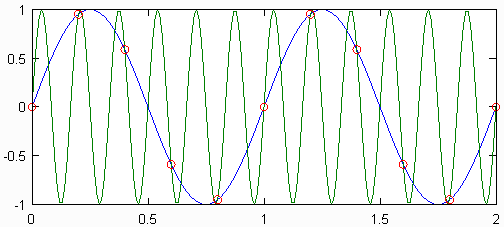
*Figure 1*

Multiplying *W* with *x* produces a vector, *a*, that has, as its entries, the dot product between the input vectors and the basic sine waves of distinct frequencies. This dot product is essentially a similarity comparison between the two waves. If a simple wave of frequency is a component of the input wave corresponding to *x*, then will contain a complex number. Otherwise, it will be zero.



*Figure 2*

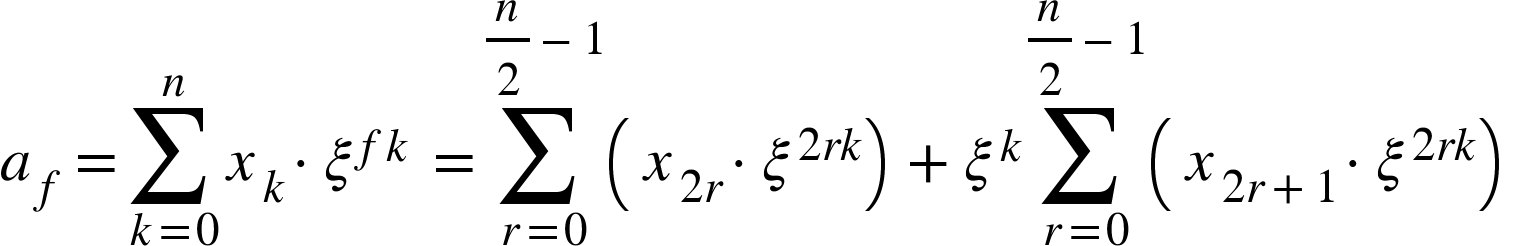
The time complexity of matrix multiplication is , (2) (Ref. 1). It is ideal to use the minimal sampling frequency necessary in order to not waste time with unnecessary data. This is where the Nyquist frequency becomes prevalent. If one desires to detect waves up to a certain frequency, then the sampling frequency must be greater than twice the maximum frequency they care to measure. This is because there are an infinite set of waves with higher frequencies that can satisfy any set of data points (Ref. 2).



*Figure 3 (Ref. 4)*

Therefore, the DFT defaults to the wave of the lowest frequency that satisfies the discrete set of points. If one were attempting to detect for the green wave with the given frequency in *Figure 3*, it would instead register the blue wave since the sampling rate is not sufficient. However, one may notice that if a Fourier matrix were to measure up to a frequency of 3Hz, assuming these bins correspond to non-negative integers, the sampling rate would need to be greater than 6Hz. The bins will cover frequencies from 0 Hz to 5 Hz. This implies that there is extraneous data in the matrix. Anything above 3 Hz is clearly greater than the Nyquist limit. However, upon further investigation, one can see that these frequencies are simply an echo of lower frequencies. Specifically, every wave with frequency *k* will also share the same data points as a wave with frequency *(k +* SR*)* where SR is the sampling rate used (Ref. 3). To reconcile this fact, eliminate all data points above the Nyquist limit and double all the amplitudes of the points below the limit. Then, finally, divide every measurement by the number of samples taken to get the amplitude of the simple wave corresponding to that frequency.

The Radix-2 FFT reduces the time complexity of the computation from to . We know from *Figure 2* how to calculate . To complete the FFT, one will divide the calculations into smaller subsets by separating the even and odd terms.



*Figure 4*

These small subsets can be split into an even number of smaller subsets as long as the number of data points in each subset is divisible by two. The calculation then becomes extremely efficient if the amount of data is a power of two. This method can then be reiterated until the only Fourier matrix needed is one of dimension two. At each stage, the number of multiplications is *n*,and there are stages, leaving the total number of calculations at vs. the original.

This group attempted to make their own implementation of an FFT function in Matlab with the goal of having better speed than the Matlab version. The Fourier Matrix discussed earlier was attempted first, followed by the Radix-2 method. The Radix-2 method showed great improvement over the Fourier Matrix, with a time complexity close to. However, this was still nowhere near the speed of the Matlab implementation. The following pages showcase the functionality of the code written, as well as time performance in comparison with the already optimized FFT that Matlab provides by default.

**Mathematical Formulation**

DFT:

amplitude of wave, number of samples taken

= phase shift, *fth* element of *a*

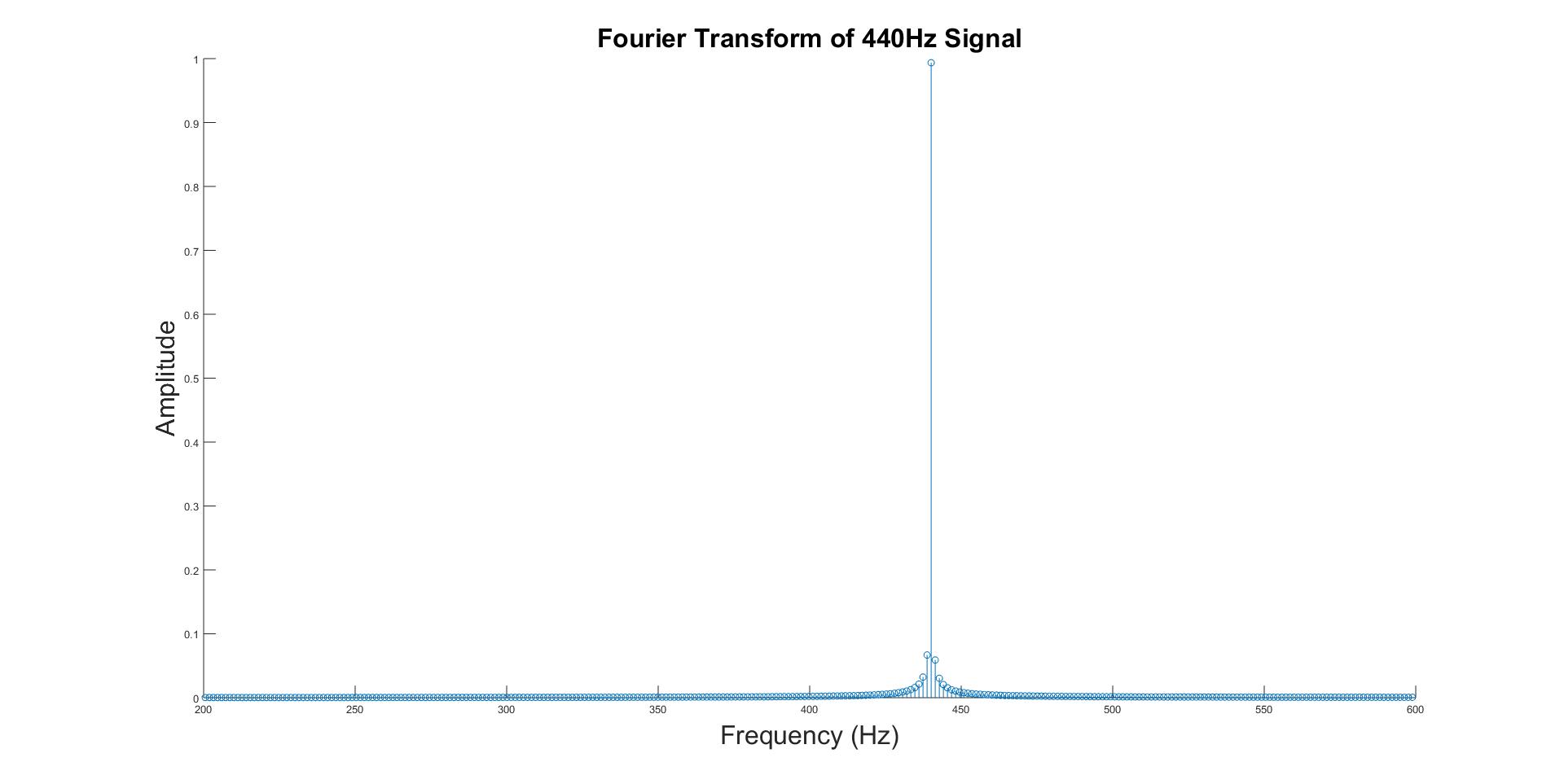
, , (*Figure 1*)

FFT:

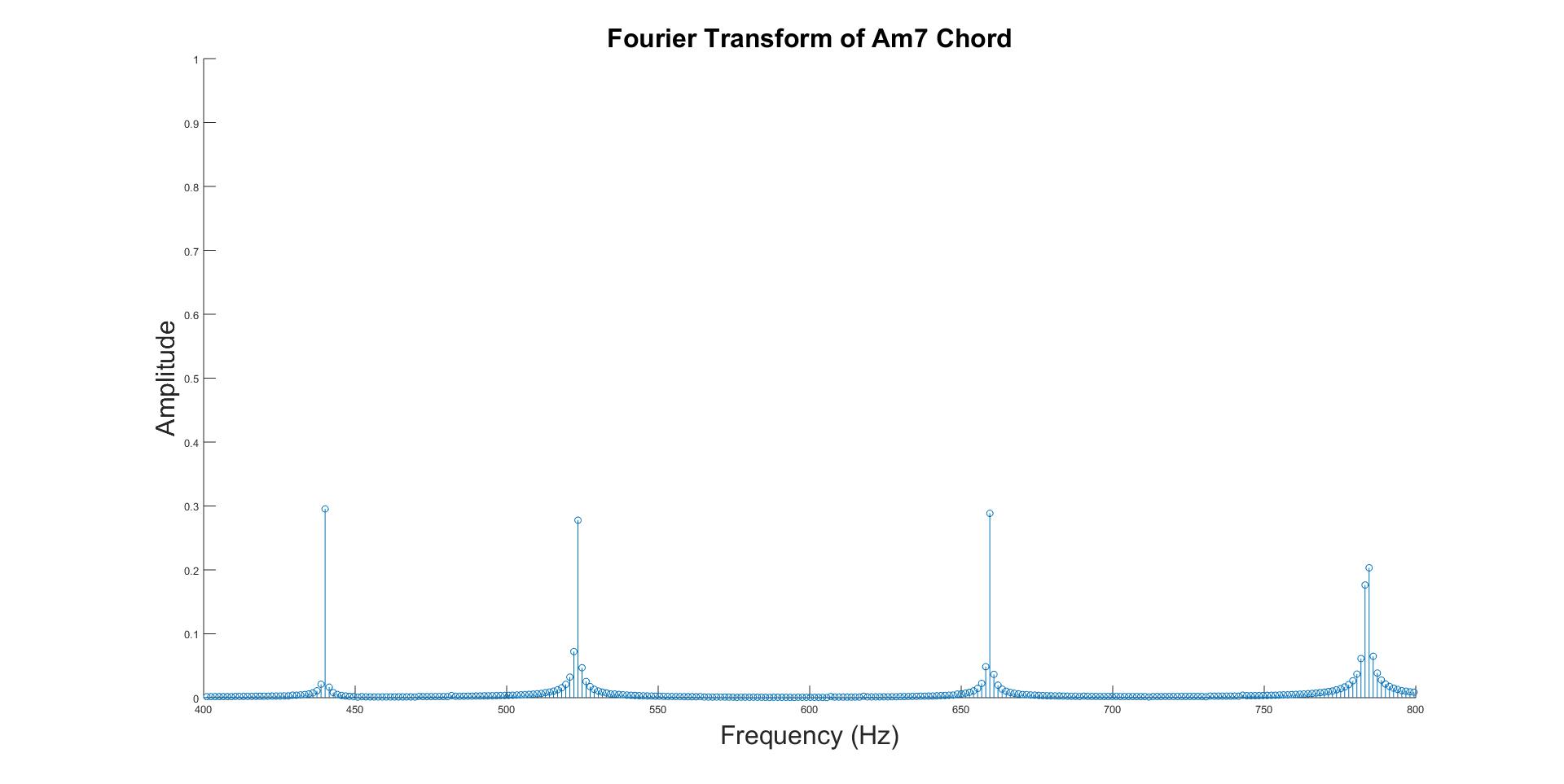
,

**Numerical Results**

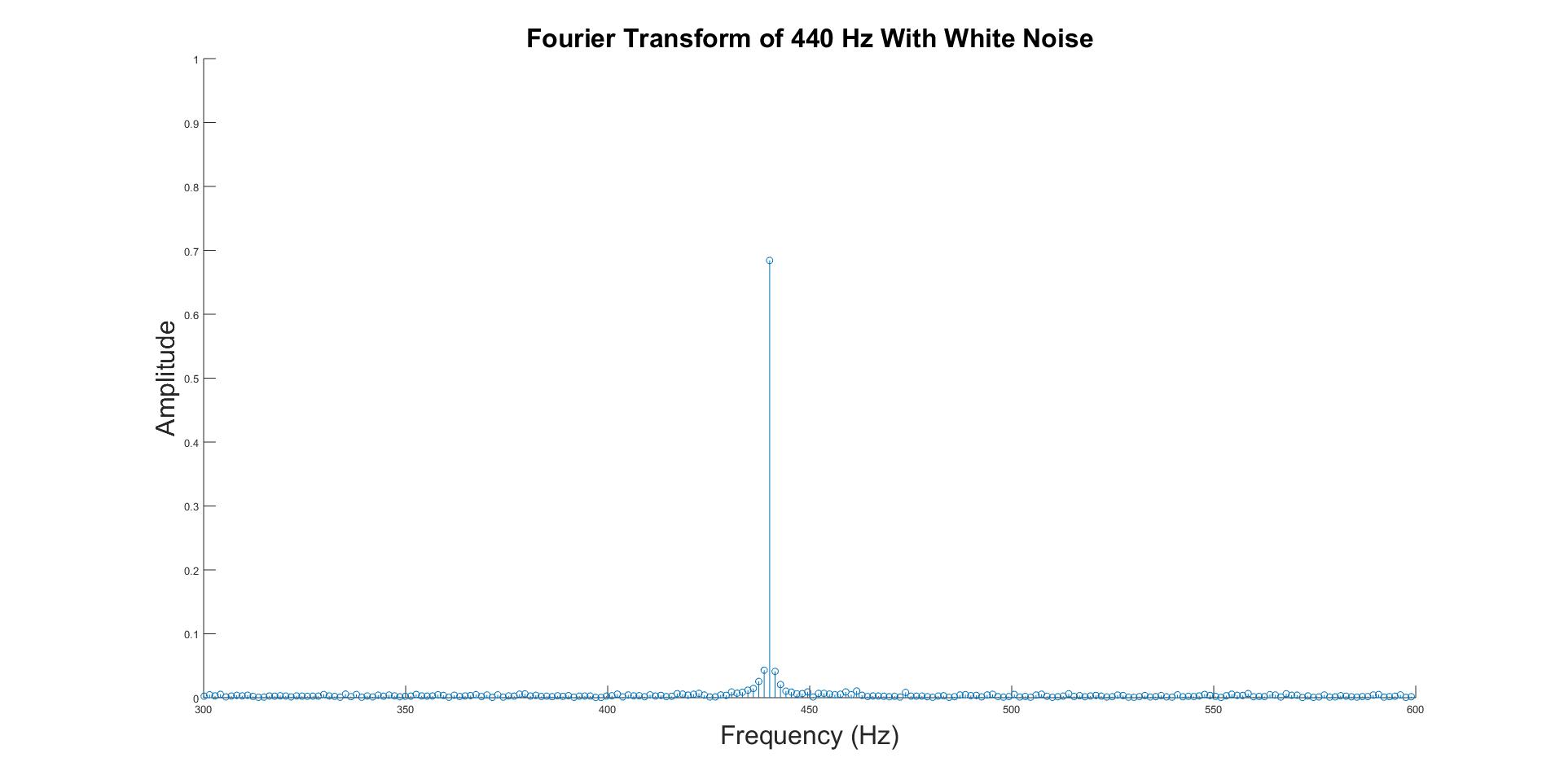
Using Audacity, a sound editing software, we were able to generate a pure sine wave with frequency of 440 Hz and amplitude of 1 in order to check whether our method was accurate. The resulting graph is achieved after the applying our own fast fourier transform to the data. Both the amplitude and the the frequency was displayed correctly.

****

Additional testing of different inputs also resulted in success, giving the exact frequencies and amplitudes of the generated sounds. The following is a plot of the frequency decomposition of an A minor 7 chord.

****

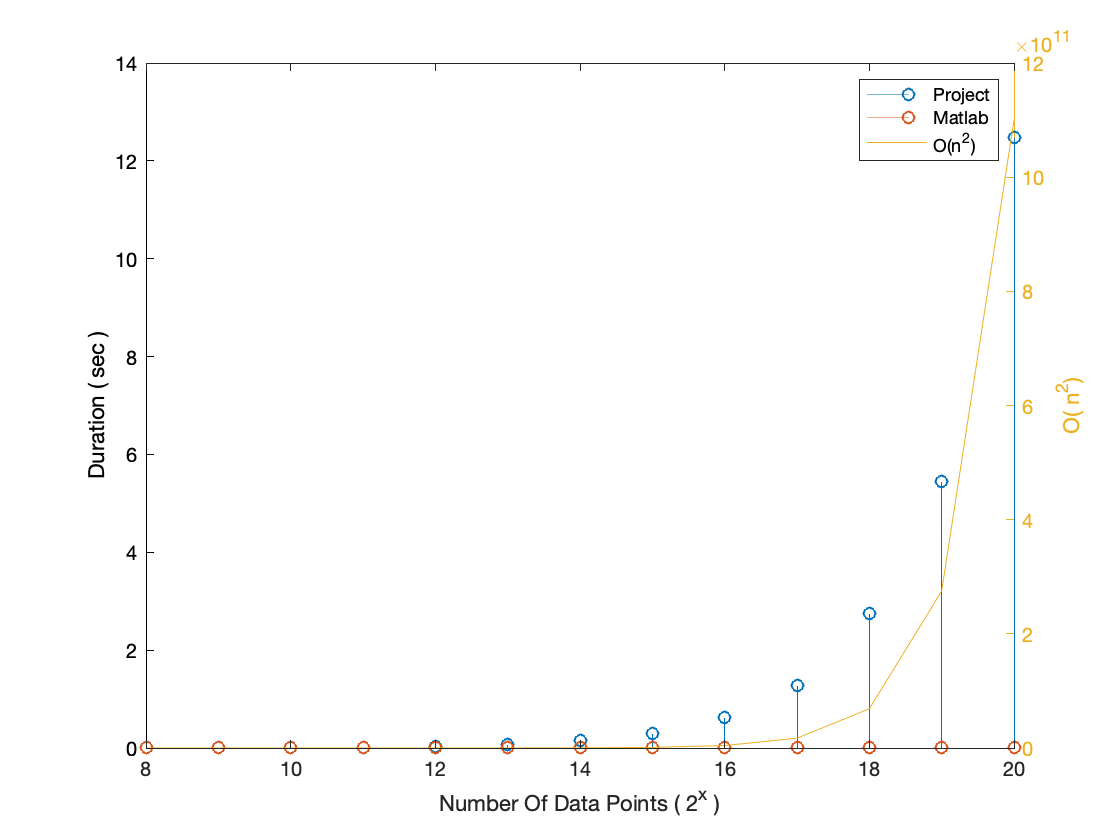
One possible application of this transformation is to reduce the amount of noise relative to the desired sounds. For instance, taking the transformation of a 440 Hz wave with noise produces the following plot:



The noise manifests itself as little perturbations in all frequencies, this however is not too challenging to remove. For instance, if a certain noise threshold was chosen, all other points could be zeroed out, producing a fairly accurate representation of the original sound, just without the noise. All that would be left to do is perform an Inverse Fourier Transform, however this project will not deal with this particular transformation.

**Time Complexity for Fast Fourier Transform**

Initially the group intended to use the standard Discrete Fourier Transform. However, the function written to create the Fourier Matrix required 14.5 GB of data to simply store the matrix. In need of a simpler computation, this plan was replaced by the Radix-2 Fast Fourier Transform. After testing, it was concluded that this form of the Discrete Fourier Transform took far less time. Below is a graph of the time the project’s Radix Fast Fourier Transform took for varying numbers of data points, compared to the time it took for the included MATLAB Fast Fourier Transform function as well as for comparison.

****

As shown in the graph above, the duration of our implementation of the Radix-2 Fast Fourier Transform takes significantly longer than Matlab’s implementation of the Fast Fourier Transform. It is also worth noting that the relationship between Duration and Number of Elements is similar to the time complexity , that was predicted.

**Conclusions and Discussion**

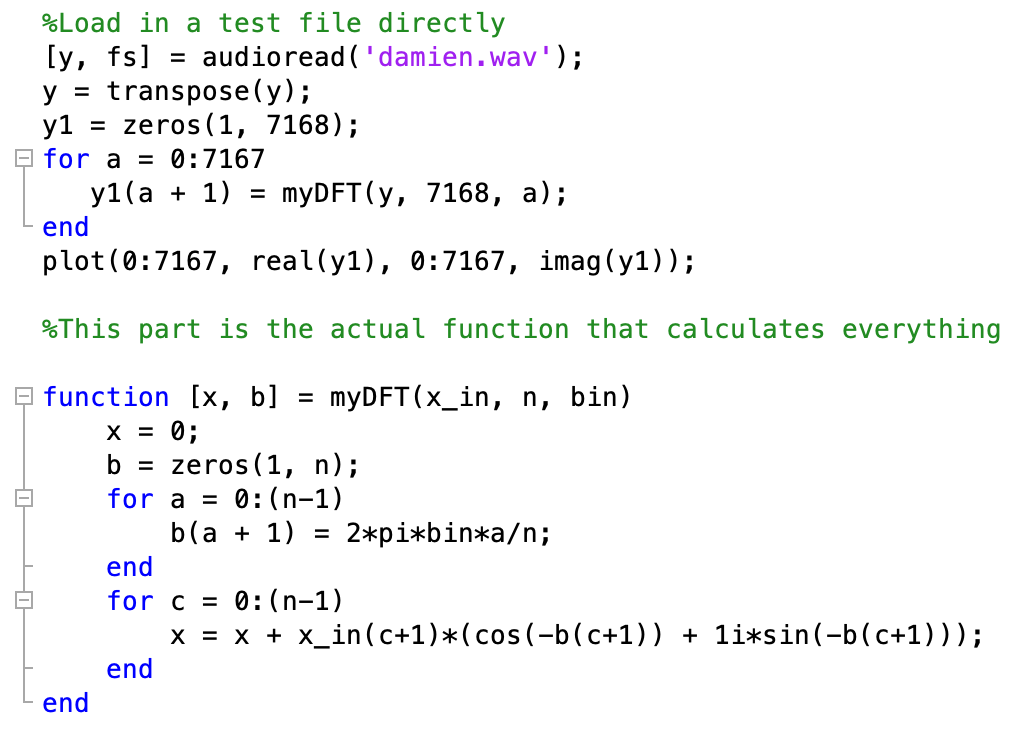
Computers cannot read in analogue waves, they can only sample the amplitude of a wave at high frequencies to achieve an approximation (Ref. 3). The Discrete Fourier Transform of the approximation of the wave performs a change of basis from the time domain to the frequency domain. The DFT accomplishes this by testing similarity between the input wave and simple waves of varying frequency.

During the course of this project, we discovered how to analyze waves numerically with accuracy. Through researching various ways of computing a faster DFT, we were able to speed up the amount of time the transformation took. With the code developed through this project, we are able to decompose composite waves from any source. This algorithm is the basis for many different applications, such as noise filtering and music analysis.

To optimize the DFT algorithm further, one could implement different FFT methods. For instance, the Cooley Tukey variation does not rely on 2n data points Radix-2 does, but instead factorizes and performs the transformation on the factors. This is most likely what the Matlab implementation of the FFT does, since it achieved performance that was significantly faster than our Radix-2 implementation. Additionally, Matlab’s implementation works really quickly for any sized input vectors, not just 2n. Matlab’s implementation also most likely features optimizations that are more Computer Science related, such as maximizing the throughput of Logic Units in the CPU, which could improve the performance several-fold.

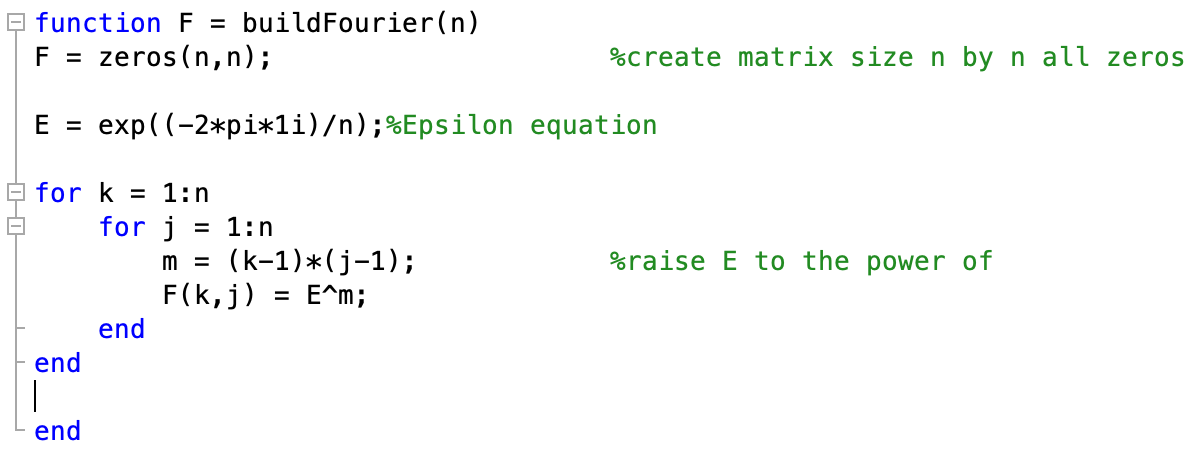
**Appendix**

**Original DFT**

****

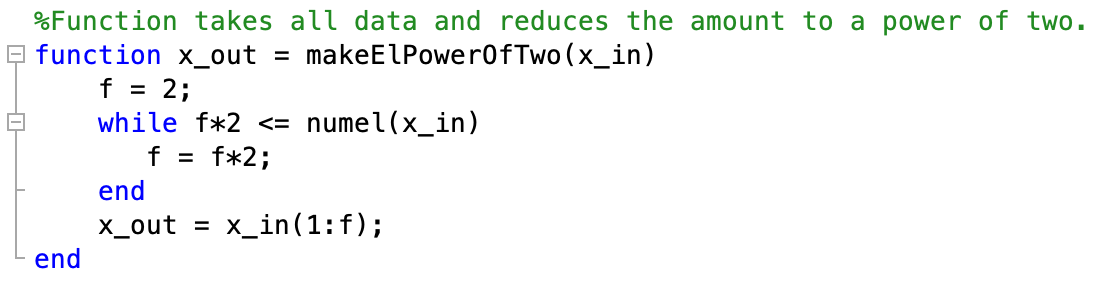
Takes in data points from wav file and computes the fourier transform for each element one at a time. Loops through all data points.

**Fourier Matrix**

****

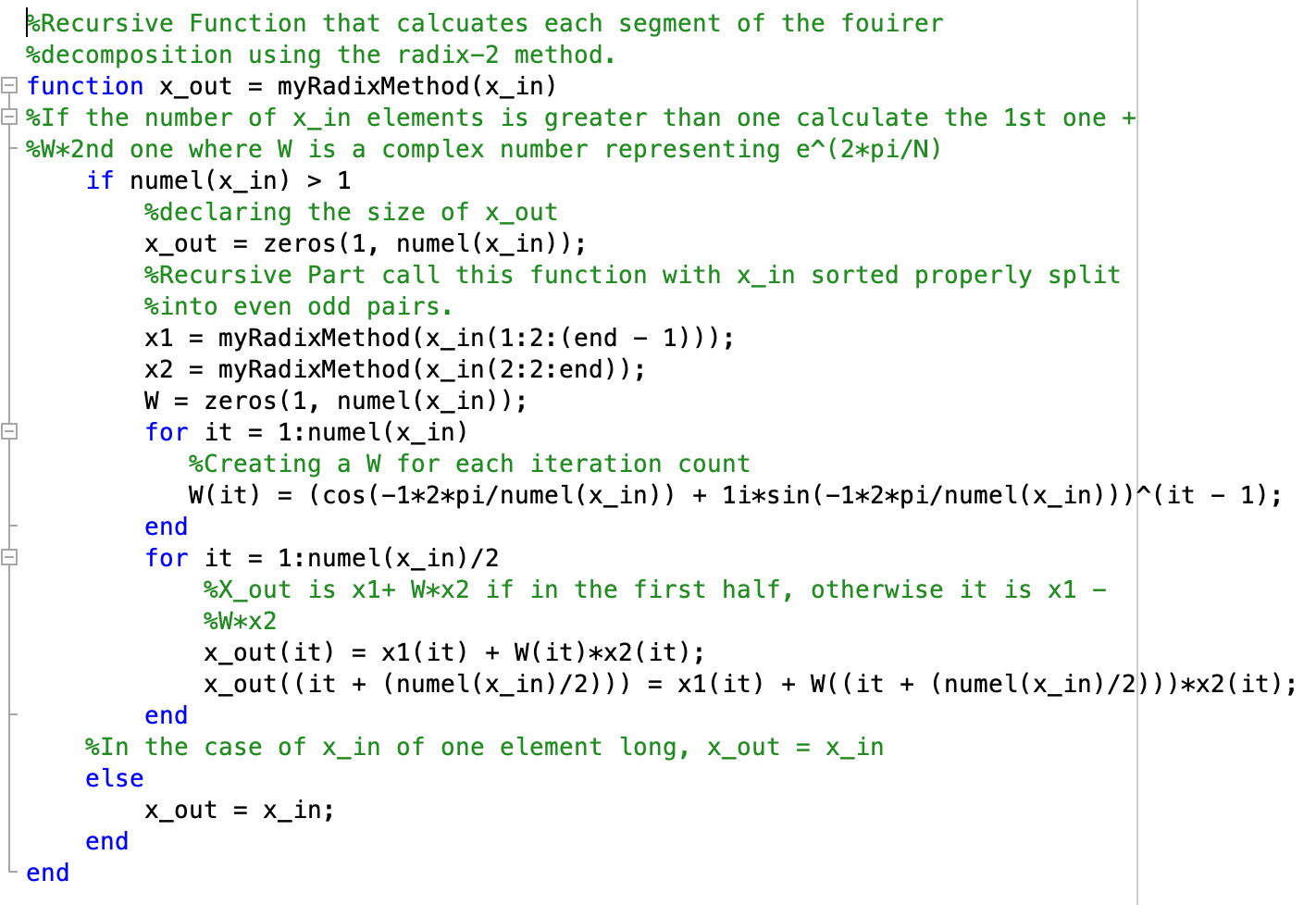
Creates an Fourier Matrix

**Data Reduction**

****

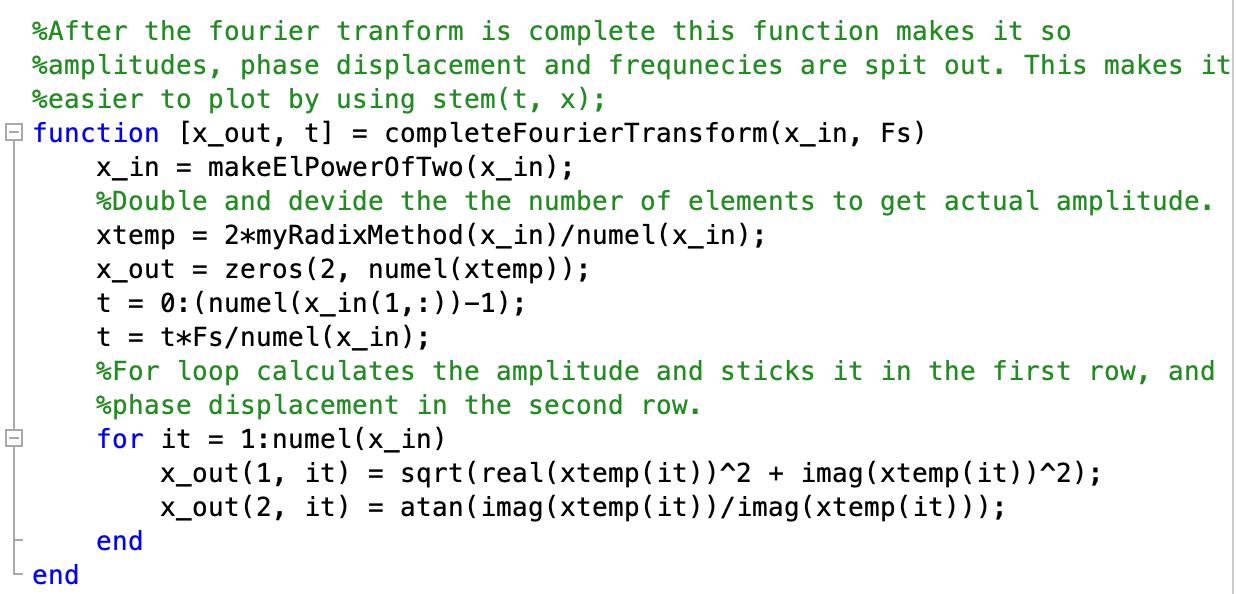
Takes the data matrix which is and reduces it such that is a power of 2.

**Radix Method**

****

Recursive function for FFT

**Final Steps**

****

Takes the completed Fast Fourier Transform and pulls out data on amplitudes, phase displacement, and frequency.

**References**

1. Meyer, Carl D. *Matrix Analysis and Applied Linear Algebra*. SIAM, 2005.
2. James, Andrew. "On the Complexity of Matrix Multiplication." *Edinburgh Research Archive*. The University of Edinburgh, 01 Jan. 1970. Web. 10 Dec. 2018.
3. Lyons, Richard G. *Understanding Digital Signal Processing*. Pearson Education International, 2013.
4. “What Is the Nyquist Limit and What Is Its Significance to Photographers?” *Photography Stack Exchange*
5. *Xu, Simon. “Discrete Fourier Transform - Simple Step by Step.” YouTube, YouTube, 3 Aug. 2015.*

1. All relevant formulations can be found in the section Mathematical Formulations. [↑](#footnote-ref-0)
2. The rows and columns of the Fourier matrix begin incrementing from 0. [↑](#footnote-ref-1)