Fourier Transform

Angel Martinez

Department of Computer Science

California State University, Chico 400 W First St Chico, CA, United States 95929-0722

ammartinez20@csuchico.edu, martinezangel3698@gmail.com

***Abstract*—**

The Fourier Transform stands as a pivotal algorithm with profound implications for our daily lives. This paper explores diverse aspects of the Fourier Transform, encompassing both the Discrete and Fast Fourier Transform algorithms. It delves into the underlying mathematical foundations, presents algorithmic representations through pseudo code, and compares the advantages and disadvantages of these algorithms. Additionally, it sheds light on the multifaceted applications of the Fourier Transform across various domains. This comprehensive overview is aimed at enhancing the general audience's comprehension of the Fourier Transform algorithm.

***Keywords*—**

Discrete Fourier Transform, Fast Fourier Transform, Time Domain, Frequency Domain, Euler's formula, & Complex numbers

I. Introduction

The purpose of the Fourier Transform is to take a time domain wave-based signal and decompose it into its frequency domain components. For example, these graphs show 2 pure wave signals and their combined waveform [1].

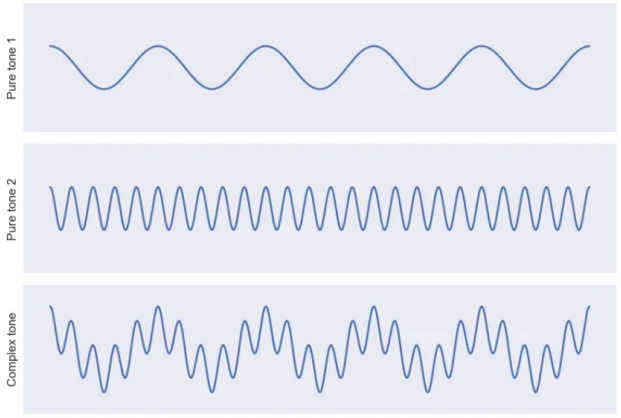

Fig. 1 Three different sinusoidal functions. The third is a combination of the top two

The Fourier Transform is then leveraged to decompose the frequency components of the wave functions. This provides a much more intuitive method of understanding the signals found within the wave function. The following figure illustrates the results of the decompositions of the wave functions

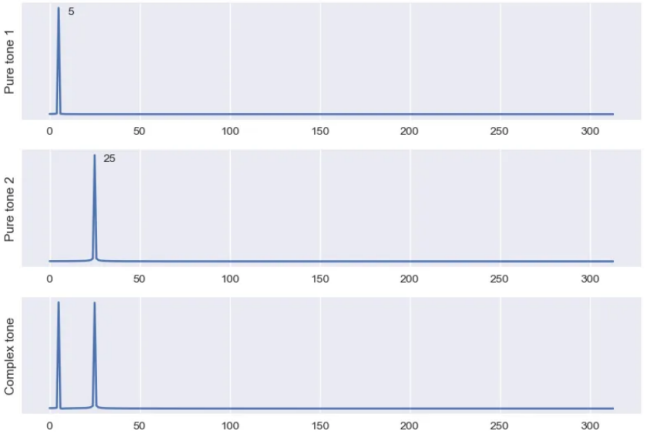

Fig. 2 Frequency Domain representation of wave functions in Figure 1

The Fourier Transform has two main algorithms that are used to convert these signal properties which are the Discrete Fourier Transform and the Fast Fourier Transform. These algorithms are not solvable by hand as it requires a lot of computations. Therefore it needs the aid of computers in order to actually compute a signal.

II. History

The great discovery that founded the fundamentals of the fourier transform was a mathematician by the name Joseph Fourier. Fourier lost his parents at a young age, was reading mathematical literature by middle school, and at the age of 15, he received the first prize for his study of *Bossut's Mécanique en général* (comprehensive treatise on mechanics) [2].

Fourier's study that led him discovering the fundamentals of the fourier transform was during his study on heat waves. Fourier noticed that if the

end of a hot rod touched the other end of a cold rod, its heat wave begins to form a sine wave as the middle becomes neutral and the opposite ends of the rod become the highest peak of the wave.

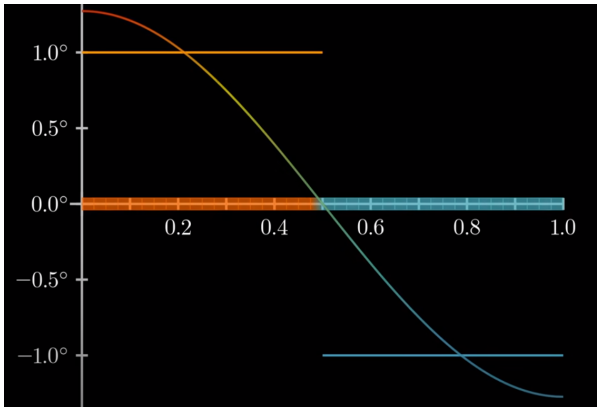

Fig.3 Heat wave where left side is a hot rod and right side is a cold rod touching in the middle

Throughout his research, Fourier began to better understand the relationship between sine waves with functions. In Fourier's analysis, he then proclaimed that, "The equations of heat conduction like those of sound or small oscillations for liquids belong to one of the most recently discovered branches of science which it is important to extend"[3].

III. Input

Before understanding how the algorithm works, it's better to understand what the fourier transform is inputting. As its input, it is receiving wave-based signals in its time domain graph. This means that it will receive a list of numbers that represent where the wave is located in the y-axis of the graph.

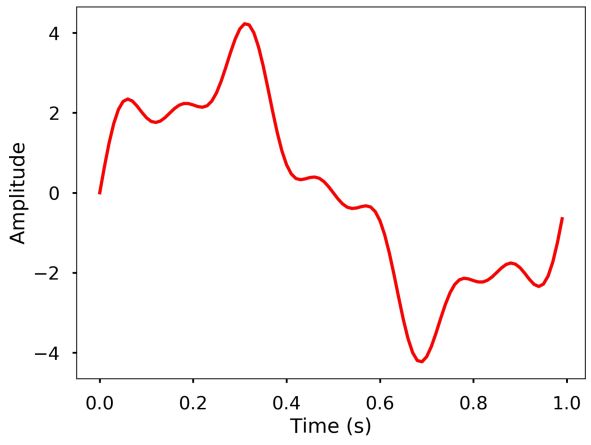

Fig.4 Time domain graph of a complex signal

Using fig.4 as an example, let's say that the fourier transform received 10 numbers representing the location of the graph every 0.1 seconds. At 0.0 seconds, the graph has an aptitude of 0. At 0.1 seconds, the graph has an amplitude of about 2 and so on. An amplitude just means the strength of the signal. Overall, with enough signal inputs, the fourier transform algorithm can interpret how the input graph looks like. The algorithm then uses this information to find the different set of frequencies (original simple signals) that make up the complex graph

IV. Output

The main purpose of the fourier transform is to output the frequency domain signal. However, it doesn't actually print out a graph but instead the information needed to construct the frequency graph. The output is a complex number for each signal input. A complex number is a set of numbers where the first part represents a real number and the second part represents an imaginary number [4].

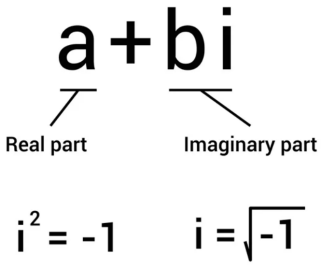

Fig.5 A visual demonstration of what a complex number and imaginary number represent

As fig.5 represents, an imaginary number can represent the square root of a negative one which is not possible with a real number.

A. Imaginary Numbers

The importance of an imaginary number is its pattern. Unlike real numbers, imagery numbers follow a pattern after every 4th exponent. For example:

$$i^0 = 1,\ i^1 = i,\ i^2 = -1,\ i^3 = -i$$

Now if you continue to increase or decrease the exponential value the pattern starts repeating

in this same order. The values of this pattern represent the 4 “corners” of the unit circle in terms of radians.

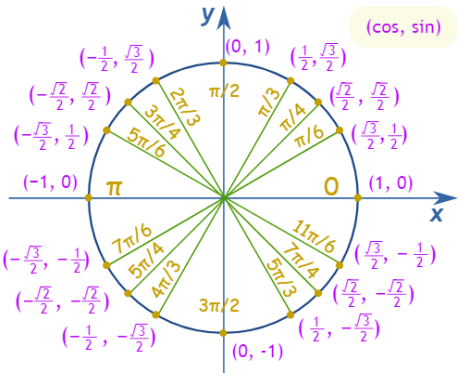

Fig.6 Unit circle

The unit circle represent its values as a complex number (real#, imaginary#). As shown on fig.6, the pattern of imaginary numbers is shown as the right angle is equal to 1, top is equal to i, left is equal to -1, and bottom is equal to -i. With this in mind, imaginary numbers have a way with constructing a circle. This is important because it can be used to help ‘unify’ the number line with the algebraic pattern shown below [5].

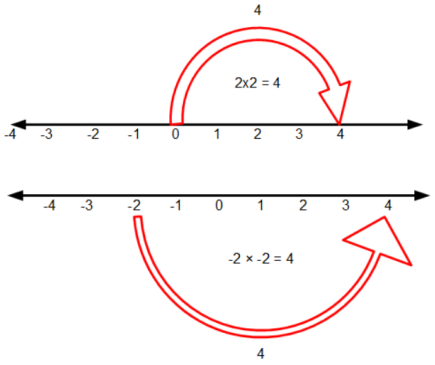

Fig.7 Number lines representing the algebraic pattern on imaginary numbers with number line

B. Euler Formula

With the properties of imaginary numbers and real numbers, complex numbers are used to actually form a circle. The values of a complex number gives information on how big the circle is and how fast it forms it. These circles will then be used to create frequency waves. The Euler formula is depicted as:

$$e^{ix} = cos(x) + i\,sin(x)$$

Combinations of sine and cosine values make a circle. Now with complex numbers, it can be used to be plotted onto a grid where the x-axis represent real numbers and the y-axis represent imaginary numbers.

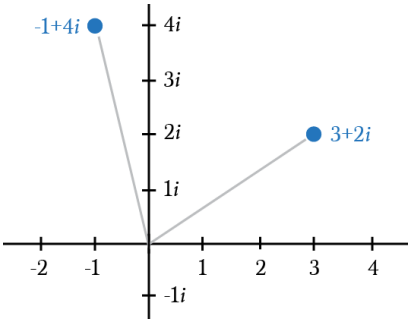

Fig.8 Example of complex number (3, 2i) and (-1, 4i) being plotted in a graph

Once the plot is found where the complex number is positioned on the grid, the cosine value can be applied to the real number and the sin value can be applied to the imaginary number to form the part of the circle that is associated with the given plot [6].

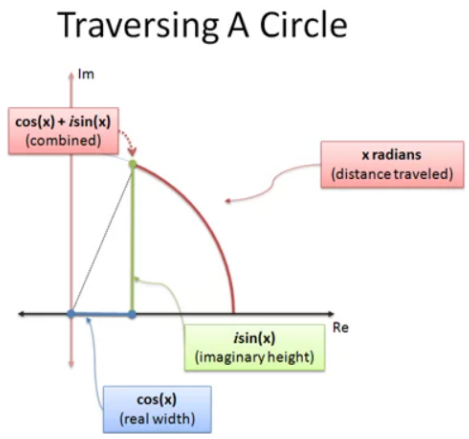

Fig.9 Example of how sin() & cos() can be applied to complex number to shape a circle

As mentioned before, complex numbers give information on how big and fast the circle is made. When complex values are given, the real number represents how big the circle is going to be, which essentially is the radius of the circle. Then the imaginary number represents the strength/growth of how much it pulls the radius at a 90 degree angle. As the imaginary number keeps on pulling on the radius, the radius constantly travels the same direction it's pointing.

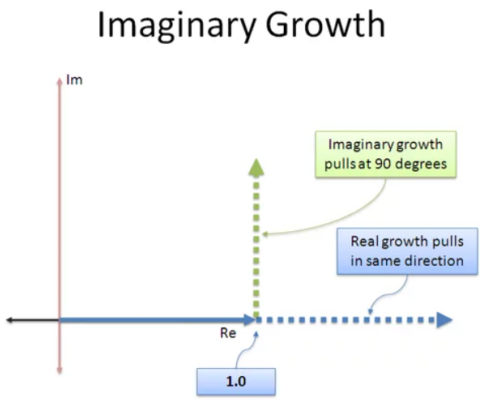

Fig.10 A layout the direction that the real & imaginary values travel

The result of fig.10 eventually begins to form a circle since the two opposing values keep on shifting 90 degrees.

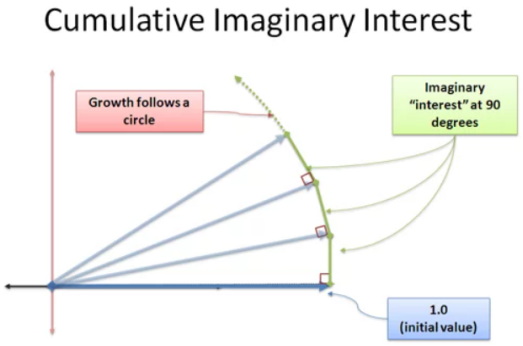

Fig.11 Example of how fig10 forms a circle

C. Circle to Wave

From the result of the Euler formula on a complex number, a sine wave can be made from the size and rotation of a circle. This can be done by representing the circle as a grid. Then as the circle begins to move, follow its point on the grid to construct the sine wave. This sine way is what can give us a representation of what frequency the input signal was.

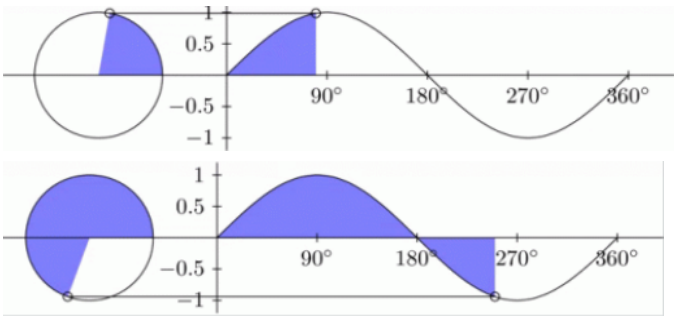

Fig.12 Example of circle can form sine wave

V. Discrete Fourier Transform

The Discrete Fourier Transform (DFT) is the main algorithm that is responsible for finding the frequency components of a graph. Its mathematical representation is seen on the figure below [7].

$$X_k = \sum_{n=0}^{N-1} x_n . e^{-i2\pi kn/N}$$

- **N:** Number of signal samples
- **n:** Current signal sample
- **k:** Current frequency where k ∈ [0, N−1]
- **xn:** The sine value at sample n
- **Xk:** The DFT which includes information on both amplitude and phase (frequency representation)

This equation is applied to each signal it processes. The Sigma notation (big fancy "E") signifies that, in order to find the frequency components of a single signal, it sums up the function being applied to all the input signals by iterating through them. This summation is crucial because it quantifies how each input signal influences the signal currently under evaluation. The notation $e^{-i2\pi kn/N}$ is the part of the algorithm that uses some of the principles of the Euler formula. To better represent the Euler formula in the DFT, the DFT algorithm can also be rewritten as:

$$\sum_{n=0}^{N-1} x_n \cdot \left[\cos\left(\frac{2\pi}{N}kn\right) - i \cdot \sin\left(\frac{2\pi}{N}kn\right)\right]$$

These are the cosine and sine functions that gets applied to the algorithm which returns the complex numbers. As mentioned before, each complex number can create a sine wave. Now in this example, the result of the different sine waves will be put together and those waves that overlap (are the same) begin to have a greater amplitude (strength).

A. Programmable Implementation

When creating a programmable script to execute the DFT algorithm, there will need to be an iterator to go through all the signals to apply the DFT algorithm. Then with each iteration, there will need to be another iteration that represents the Sigma portion of the algorithm. These two iteration will be represented as for-loops, and inside the second for loop is where the equation $e^{-i2\pi kn/N}$ gets applied.

```
function DFT(signals):
    freq_domain = []
    N = length(signals)

    for n from 0 to N-1:
        freq = (0,0)
        for k from 0 to N-1:
            w = e^(-i*2*pi*k*n/N)
            freq = freq + (signal[k] * w)
        freq_domain.append(freq)

```

Fig.13 Pseudo code example of DFT

B. Inverse

An important aspect of the DFT is that it is reversible. That means just like how you are able to break down a signal from its time domain to frequency domain, you can take in a frequency domain and output the time domain graph. Inorder to do so, the DFT algorithm must be tweak a bit. The equation for the IDFT formula is shown below [8].

$$x[n] = \frac{1}{N}\sum_{m=0}^{N-1} X[m] \cdot e^{+2\pi \cdot \mathrm{j} \cdot \frac{m}{N} \cdot n}$$

Fig.14 IDFT equation

- **j:** = i (from DFT formula)
- **m:** = current signal sample
- **X[m]:** frequency domain signal
- **x[n]:** time domain signal

Beside the different annotation used to present similar values from the DFT formula, the main difference between the two equations is that the e^ portion is a positive now instead of a negative. Then the signal is multiplied by 1/N which just divides the output by the number of signals it processes. Its Programmable Implementation can be represented below.

```
function IDFT(freq_domain):
    signals = []
    N = length(freq_domain)

    for n from 0 to N-1:
        signal = (0,0)
        for k from 0 to N-1:
            w = e^(i*2*pi*k*n/N)
            signal = signal + (freq_domain[k]*w)
        signal = signal * (1/N)
        signals.append(signal)

```

Fig.14 Pseudo code example of IDFT

C. Runtime

One of the drawbacks of the DFT or IDFT algorithms has to do with how long it takes to compute a signal. Referring to the Pseudo Code, there are 2 iterations happening with each iteration going through the list of the input signals. That means that if you have an input of 10 signal plots, then the computer has to do (10^2) computations which is 100. Now with real world complex signals, you may need to input thousands of signal plots or even millions to make sure the algorithm understands how your signal looks like. This is where you can experience a long wait time for your computer to be able to process that many signals. Therefore, the asymptotic runtime of the DFT or IDFT algorithm is $O(n^2)$.

VI. Fast Fourier Transform

With a big drawback of the DFT being its runtime, The Fast Fourier Transform is able to drastically reduce the runtime that the DFT algorithm takes. In 1965, James Cooley co-invented with John Tukey the FFT algorithm [9]. The FFT equation can be depicted as:

$$\begin{aligned} F_n &= \sum_{k=0}^{N-1} f_k \, e^{-2\pi i n k/N} \\ &= \sum_{k=0}^{N/2-1} e^{-2\pi i k n/(N/2)} f_{2k} + W^n \sum_{k=0}^{N/2-1} e^{-2\pi i k n/(N/2)} f_{2k+1} \\ &= F_n^e + W^n F_n^o, \end{aligned}$$

Fig.15 FFT equation

The top equation represents the DFT equation that we have covered already. Now the equation below is the Danielson-Lanczos Lemma approach which is how the FFT splits up the equation [10]. Basically, the algorithm splits up the input of signals by its odd and even indexes. It repeats this process until each signal input is in its own array. From here, it begins to reconstruct the signals together by applying the $e^{-i2\pi kn/N}$ equation.

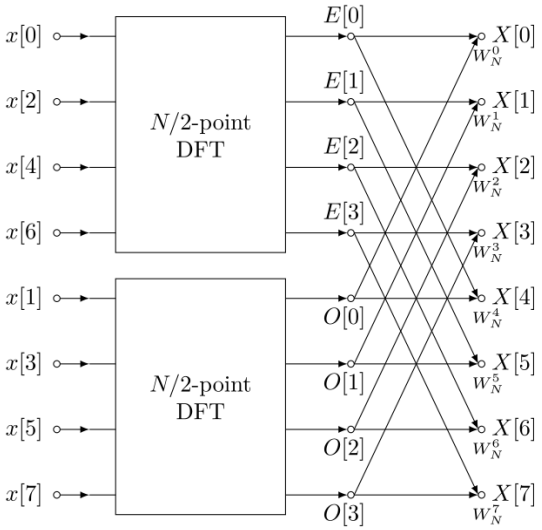

Fig.16 FFT Diagram

This diagram represents how the FFT works. The N/2-point DFT boxes represent the dividing portion of the algorithm which splits up each signal. Then the conquering process begins while still following the same odd and even parameters. By applying the equation to each of the signals in each step in its reconstructing phase, then swapping it back to its even and odd indices, is how the FFT is able to quantifies how each input signal influences the signal currently under evaluation without having to have multiple iterations through the whole list of inputs.

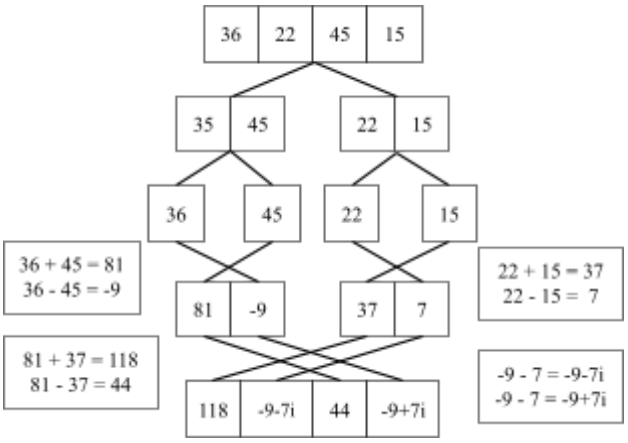

Fig.17 FFT Example

Fig.17 gives an example of how the FFT algorithm works with an input of [36, 22, 45, 15]. It starts dividing the input until you have 4 lists of 1 signal inside. During its conquering phase the signal under review performs its equation operation with its neighboring array corresponding to the same index. That is why in the second to last row, 81 gets paired with 37. Then although the $e^{-i2\pi kn/N}$ equation isn't being shown, it is being applied to the odd index of the array as that is what turns the value into a complex number.

A. Programmable Implementation

When making the programmable implementation of the FFT algorithm, a method of recursion can be used inorder to mimic the divide and conquer portion of the algorithm. Recursion is when the function calls itself over and over until a condition is met which returns its values to continue the rest of the script. Then as values are returning, the conquering phase begins as now you can perform an operation on it which then returns to me merged with the overall answer.

```
1  function FFT(signals):
2      N = length(signals)
3      if N == 1:
4          return signals
5
6      even = []
7      length(even) = N/2
8      odd = []
9      length(odd) = N/2
10
11     for i from 0 to N/2
12         even[i] = signals[2*i]
13         odd[i] = signals[2*i+1]
14
15     even_fft = FFT(even)
16     odd_fft = FFT(odd)
17
18     freq_domain = []
19     length(freq_domain) = N
20     // Conquering occurs here
21     for k from 0 to N/2:
22         w = e^(-2i * pi * k/N)
23         freq_domain[k] = even_fft[i] + w * odd_fft[i]
24         freq_domain[k + N / 2] = even_fft[i] - w * odd_fft[i]
25
26     return freq_domain
27
```

Fig.18 Pseudo code example of FFT

As shown on fig.18, the input is being divided by its even and odd indices. Then in lines 15 and 16, the function is calling itself on the smaller sub arrays. when the array size is = to 1, then it returns it back and it continues down into the for loop in line 22 which applies the equation as seen in fig.15. Then as it returns the freq_domain array, it starts merging the results.

B. Inverse

Just like how the DFT algorithm had its inverse function, the same also applies to the FFT. In the IFFT, the equation changes just like the IDFT where instead of $e^{-i2\pi kn/N}$, it's $e^{+i2\pi kn/N}$. Then once you have your answer, you divide each signal by the size of the whole initial input array of signals. Therefore, when applying the IFFT function in a programmable implementation, the only part of the function that you will change is as shown below.

```
for k from 0 to N/2:
    w = e^(2i * pi * k/N)
    freq_domain[k] = even_fft[i] + w * odd_fft[i]
    freq_domain[k + N / 2] = even_fft[i] - w * odd_fft[i]
    freq_domain[k] = freq_domain[k] * 1/N
    freq_domain[k + N / 2] * freq_domain[k + N / 2] * 1/N
```

Fig.19 Pseudo code example of FFT

C. Runtime

As mentioned before, the main purpose of the FFT is to use the principles of the DFT function but make it faster. By using a divide and conquer approach instead of iteration inside an iteration, the FFT is much more asymptotically efficient. As the FFT does the divide process, it splits the input array recursively in half which gives it a runtime of $O(log(n))$. Then as the function continues to rebuild itself in the conquering phase and applying the FFT equation to the signals, the runtime adds a $O(n)$ runtime to it making the total asymptotic runtime being $O(nlog(n))$. The graph below can help visualize how much more efficient this approach is as the DFT begins to drastically increase the higher the input is while the FFT barely begins to increase in time.

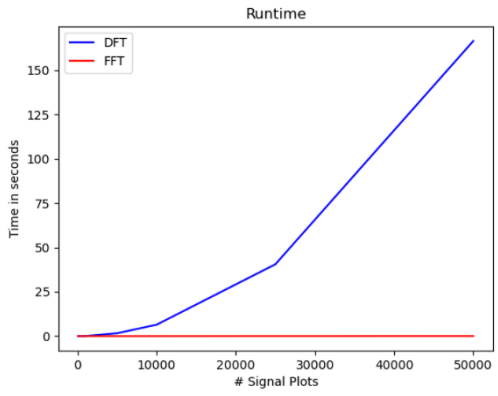

Fig.20 visual graph of DFT vs FFT Runtime

VII. Applications

Overall the Fourier Transform can be thought of as a pattern recognizer. In terms of signal processing, the pattern is finding what frequencies make up the complex signal. Now A use for the fourier transform and its inverse can be filtering out unwanted noises from an audio recording. If you get the frequency domain of a audio recording and realizes that a fan was making the audio hard to listen to, you can remove the frequency that the fan was omitting and use the inverse fourier transform to reconstruct the original audio without any background noise.

Another use for its ability to detect patterns can be with voice recognition. Each person has a unique voice which means that each person's voice is made up of a set of frequencies. Therefore the fourier transform can differentiate the difference between peoples voices. Also a Seismograph uses the fourier transform to determine spikes in the earth to measure when a earthquake is about to hit [11]. The spikes size also measure the magnitude of the earthquake which is very useful inorder to determine how to handle the situation

Lastly, the fourier transform can also be used with images. This is very beneficial especially when it comes to medical scanning. It can receive signal inputs from a scan to evaluate its patterns. This can be used to enhance the clarity of an image or amplify part of an image that otherwise couldn't be detected by a human. The figure below gives a representation of amplifying a voxel in the brain inorder to discover disease biomarkers [12].

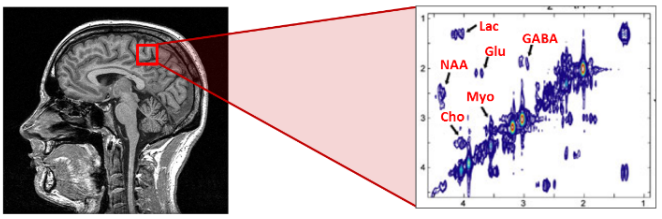

Fig.21 FFT example in Magnetic Resonance Imaging

VIII. Conclusion

Overall the fourier transform is an algorithm that takes a time domain wave-based signal and decomposes it into its frequency domain components. This provides information on what

frequencies a complex signal is made up of. The main algorithm for the fourier transform is the Discrete Fourier Transform. It uses the Euler formula to convert a signal's plot value into a complex value that can be used to form a sine wave. As the DFT algorithm constructs a sine wave for each signal plot input, it is able to connect the matching signals to amplify its amplitude resulting in a spike on the frequency that is used to make up the complex signal. The other algorithm is the Fast Fourier Transform which uses the same principles as the DFT but instead of having to iterate through the list of inputs for each input, it is able to do a divide and conquer technique to drastically reduce the runtime to compute a signal. Lastly, the Fourier Transform has had its many uses in our daily lives as it can be used for audio filtering/compresion, voice recognition, earthquake detection, image processing, and much more.

References

[1] P. B. Bryan, "Fourier transform, applied (1): Introduction to the frequency domain," Medium, https://towardsdatascience.com/the-fourier-transform-1-ca31adbfb9ef (accessed Jul. 6, 2023).

[2] J. J. O'Connor and E. F. Robertson, "Joseph Fourier - biography," MathsHistory, https://mathshistory.st-andrews.ac.uk/Biographies/Fourier/ (accessed Oct. 28, 2023).

[3] [1] Steven Roose, Daniel, Michaelis Dominic, Geremia and Alexandre Eremenko "How was the Fourier transform created?," MathematicsStackExchange, https://math.stackexchange.com/questions/310301/how-was-the-fourier-transform-created (accessed Oct. 28, 2023).

[4] P. Pinak Dusane, "Is there anything more complex than complex numbers?," ScienceABC, https://www.scienceabc.com/pure-sciences/is-there-anything-more-complex-than-complex-numbers.html (accessed Oct. 29, 2023).

[5] C. G. Butler, "What are imaginary numbers used for in the real world?," Quora, https://www.quora.com/What-are-imaginary-numbers-used-for-in-the-real-world (accessed Oct. 29, 2023).

[6] "Intuitive understanding of Euler's formula," BetterExplained, https://betterexplained.com/articles/intuitive-understanding-of-eulers-formula/ (accessed Oct. 29, 2023).

[7] O. Alkousa, "Learn Discrete Fourier transform (DFT)," Medium, https://towardsdatascience.com/learn-discrete-fourier-transform-dft-9f7a2df4bfe9 (accessed Oct. 29, 2023).

[8] "Discrete fourier transforms," Roymech, https://www.roymech.co.uk/Useful_Tables/Maths/fourier/Maths_Fourier_DFT_FFT.html (accessed Oct. 31, 2023).

[9] Cooley-Tukey and other algorithms for fast fourier transform , https://researcher.watson.ibm.com/researcher/view_page.php?id=6921#:~:text=What%20we%20accomplished%3A%20James%20Cooley,was%20invented%20by%20Shmuel%20Winograd. (accessed Oct. 31, 2023).

[10] "Danielson-Lanczos Lemma," from Wolfram MathWorld, https://mathworld.wolfram.com/Danielson-LanczosLemma.html (accessed Nov. 1, 2023).

[11] "Geology," Seismic Applications for the FFT, https://w.astro.berkeley.edu/~jrg/ngst/fft/seismic.html#:~:text=Seismic%20research%20has%20always%20been,Transform%20(and%20the%20FFT). (accessed Nov. 1, 2023).

[12] SFFT: Sparse fast fourier transform, https://groups.csail.mit.edu/netmit/sFFT/applications.html (accessed Nov. 1, 2023).