[redacted]

Statement of problem and its history.

As of present, Fourier analysis is necessary for many applications spanning a variety of fields including signal processing, software-defined radio, control theory, ect. The discrete Fourier Transform with N samples is in the form of $X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}}$ and is noted to consist of N complex multiplications and N complex additions resulting in a Fourier Transform requiring N^2 operations. Despite its common usage today, the exponentially increasing complexity and time to calculate the discrete Fourier Transform was not viable in the early 1800's but the topic resurfaced in the mid 1900's leading to one of the important numerical algorithms of all time: the Fast Fourier Transform.

Historically, the Fast Fourier Transform, commonly referred to as the FFT, was discovered by Carl Friedrich Gauss in 1805 as a computationally memory efficient method to calculate the discrete Fourier Transform of a signal with N samples. What sets the FFT apart from previous discrete Fourier Transform algorithms is its remarkable speed. Despite the lowered calculation complexity, it was still impractical to utilize the FFT algorithm and this result went largely unnoticed until the fast-paced developments of the digital computer in the 20th century.

IBM researcher James Cooley and Princeton mathematician John Tukey are credited with rediscovering in 1965 the FFT algorithm most used today [1]. The Cooley-Tukey FFT algorithm utilizes a divide and conquer process to reduce an N sampled discrete Fourier Transform into two smaller N/2 sized discrete Fourier Transforms. Then, each of the N/2 sized discrete Fourier Transforms are reduced into two smaller N/4 sized Fourier Transforms and so on. The process is repeated until each discrete Fourier Transform contains N=2 elements. For

[redacted]

this reason, the Cooley-Tukey FFT is most effective and accurate when the number of samples N is a power of two.

In cases where N is not a power of two, two approaches are generally taken. The first and easiest solution is to include zeros at the end of the original sequence commonly referred to as zero padding to increase the sequence elements, so they are a power of two. This method concatenates the original signal with a sequence composed of zeros to the end of the signal to the next highest power of two allowing for the Cooley-Tukey FFT to be applied. The second alternative involves utilizing a different FFT algorithm altogether such as Rader's FFT algorithm for prime sizes. Unfortunately, due to the difficulty in implementing the alternative FFT algorithms, the Cooley-Tukey FFT remains the most used and practical implementation.

With the historical background explained, the discrete Fourier Transform will be introduced along with important complex analysis properties that will be used to manipulate the original direct Fourier Transform into the Cooley-Tukey FFT. Furthermore, a FFT program will be written to perform the simple signal processing application of removing background noise from a noisy audio signal.

Definitions and Notations

The discrete time Fourier Transform (DTFT) of a continuous sequence x(t) is shown in equation 1 as

$$X_\delta(i\omega) = \sum_{n=0}^{N-1} x(nT_s)e^{-i\omega nT_s} \tag{1}$$

where T_s is the sampling period in between each of the N samples in the original sequence of x(t). In simpler terms, the continuous signal x(t) is sampled at uniform length intervals of T_s to create a discrete sequence which is then made periodic. However, computers can only represent discrete information rather than continuous signals of finite duration since a continuous signal would take an infinite number of points to recreate. For this reason, the discrete time Fourier Transform will be converted into a uniformly spaced discrete Fourier Transform where the samples are each apart from one another by 1. To accomplish this, let $T_s = \frac{T_o}{N}$ where T_s is the original sampled period and N is the number of samples taken of x(t). Substituting this value into equation 1 results in the spectrum shown in equation 2.

$$X_\delta(i\omega) = \sum_{n=0}^{N-1} x(nT_o/N\,)e^{-\frac{i\omega nT_o}{N}} \tag{2}$$

Note that if the angular frequency of the fundamental frequency is $\omega = 2\pi f_o = \frac{2\pi}{T_o}$ and the interval between each discrete element is 1 so $x_n = x(nT_s) = x(nT_o/N)$, then we can utilize the convolution property (*) of the discrete time Fourier Transform between $x_\delta(t)$ and an infinite number of unit impulses spaced T_o to show for some $x_2(t) = x_\delta(t) * \sum_{n=-\infty}^{\infty} \delta(t - nT_o\,)$ results in equation 3.

$$X_2(i\omega) = X_\delta(i\omega)\frac{2\pi}{T_o}\sum_{k=-\infty}^{\infty}\delta\left(\omega - \frac{k2\pi}{T_o}\right) = \frac{2\pi}{T_o}\sum_{k=-\infty}^{\infty} X_\delta\left(\frac{ik2\pi}{T_o}\right)\delta\left(\omega - \frac{k2\pi}{T_o}\right) \tag{3}$$

Substituting this expression further with $\omega = 2\pi f_o = \frac{2\pi}{T_o}$ reveals that algebraic simplifications can be performed to reduce the complex exponential resulting in equation 4.

$$X_2(i\omega) = \frac{2\pi}{T_o}\sum_{k=-\infty}^{\infty}[\sum_{n=0}^{N-1} x\left(\frac{nT_o}{N}\right)e^{-\frac{i2\pi knT_o}{NT_o}}]\,\delta\left(\omega - \frac{k2\pi}{T_o}\right) \tag{4}$$

$$X_2(i\omega) = \frac{2\pi}{T_o} \sum_{k=-\infty}^{\infty} \left(\sum_{n=0}^{N-1} x\left(\frac{nT_o}{N}\right) e^{-\frac{i2\pi kn}{N}} \right) \delta\left(\omega - \frac{k2\pi}{T_o}\right) \tag{4}$$

It can be observed that $X_2(i\omega)$ is discrete in frequency with uniform sampling spaces of $\omega = \frac{2\pi}{T_o}$. From the previous definition that $x_n = x(nT_s) = x(nT_o/N)$, the discrete Fourier Transform can now be defined from equation 4 resulting in equation 5

$$X_k = \sum_{n=0}^{N-1} x\left(\frac{nT_o}{N}\right) e^{-\frac{i2\pi kn}{N}} = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}} = \sum_{n=0}^{N-1} x_n \, {W_N}^{kn} \tag{5}$$

where $W_N = e^{\frac{-2\pi i}{N}}$ is commonly referred to as the twiddle factor to simplify complex exponential values. This notation will be used extensively when showing the derivation of the Cooley-Tukey FFT algorithm.

Given that the DFT contains a complex exponential, the next step is to prove Euler's identity to simplify later calculations.

Theorem: Euler's Identity $\quad e^{i\Theta} = \cos\Theta + \mathrm{i}\sin\Theta$

Proof: Suppose

$$\mathrm{z} = \cos\Theta + \mathrm{i}\sin\Theta$$

where z∈**C** and when $\Theta = 0, \mathrm{z} = 1 + \mathrm{i}0 = 1$. Then, the derivative of z will be taken with respect to Θ resulting in

$$\frac{dz}{d\Theta} = -sin\Theta + \mathrm{i}\cos\Theta$$

but note that $iz = i(\cos\Theta + \mathrm{i}\sin\Theta\,) = -\sin\Theta + \mathrm{i}\cos\Theta$.

Thus,

[redacted]

$$\frac{dz}{d\Theta} = -sin\Theta + \mathrm{i}\cos\Theta = \mathrm{iz}.$$

By multiplying by $d\Theta$ and dividing by z on both sides, the previous expression can be rearranged to obtain

$$\frac{dz}{z} = \mathrm{i}d\Theta.$$

Integrating both sides results in

$$\int \frac{dz}{z} = \ln|z| = \int \mathrm{i}d\Theta = \mathrm{i}\,\Theta + \mathrm{c}$$

where c is some constant. Further manipulation reveals that

$$e^{\ln|z|} = z = e^{i\Theta + \mathrm{c}}$$

and since z=1 when $\Theta = 0$, then $z = 1 = e^{\mathrm{c}}$ resulting in c=ln1=0.

Therefore,

$$z = e^{i\Theta} = \cos\Theta + \mathrm{i}\sin\Theta. \qquad (6)$$

Another property of the DFT is the result X_k is always periodic for every N samples.

Property: Discrete Fourier Transform is periodic.

Proof: Given that

$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}}$, $X_{k+N} = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi i(k+N)n}{N}} = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}} e^{\frac{-2\pi iNn}{N}} = X_k e^{-2\pi in}$. Since n and N are integers such that n∈{0,N-1}, then $e^{-2\pi ni} = \cos(-2\pi n) + \mathrm{i}\sin(-2\pi n)$=1+i0=1 for any value of n. This results in $X_{k+N} = X_k$ and therefore the DFT X_k is periodic for N samples.

Two properties that will be crucial in simplifying the inverse discrete Fourier Transform are the complex conjugate properties conj(z_1 z_2)=conj(z_1)conj(z_2) and conj(z_1+ z_2)=conj(z_1)+conj(z_2).

<u>Property:</u> conj(z_1 z_2)=conj(z_1)conj(z_2).

<u>Proof:</u> Let z_1=a+ib and z_2=c+id where a, b, c, and d are real numbers. Then,

conj(z_1 z_2)=conj((a+ib)(c+id))=conj(ac-bd+ibc+iad)=conj((ac-bd)+i(bc+ad)).

Reducing further results in conj(z_1 z_2)=conj((ac-bd)+i(bc+ad))=(ac-bd)-i(bc+ad).

Moreover, conj(z_1)conj(z_2)=conj(a+ib)conj(c+id)=(a-ib)(c-id)=(ac-bd)-i(bc+ad).

Therefore, conj(z_1 z_2)=conj(z_1)conj(z_2).

<u>Property:</u> conj(z_1+ z_2)=conj(z_1)+conj(z_2).

<u>Proof:</u> Let z_1=a+ib and z_2=c+id where a, b, c, and d are real numbers. Then,

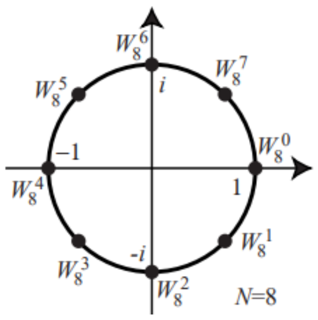
conj(z_1+ z_2)=conj((a+ib)+(c+id))=conj((a+c)+i(b+d))=(a+c)-i(b+d).

Furthermore, conj(z_1)+conj(z_2)=conj(a+ib)+conj(c+id)=(a-ib)+(c-id)=(a+c)-i(b+d).

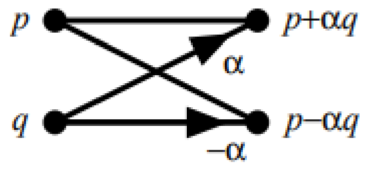
Therefore, conj(z_1+ z_2)=conj(z_1)+conj(z_2).

The twiddle factor $W_N = e^{\frac{-2\pi i}{N}}$ can also be simplified from the original DFT equation utilizing the Nth roots of unity of ${W_N}^{kn} = e^{\frac{-2\pi kni}{N}}$. Thus, for the general cases where N is a power of two, ${W_N}^0 = {W_N}^N = 1$, ${W_N}^{\frac{N}{4}} = -\mathrm{i}$, ${W_N}^{\frac{N}{2}} = {W_N}^{\frac{(odd\,k)N}{2}} = -1$, and ${W_N}^{\frac{3N}{4}} = \mathrm{I}$. Since the DFT was also shown to be periodic every N samples, then ${W_N}^{kn} = {W_N}^{kn+cN}$ where c∈**Z.** ${W_N}^1$ is commonly referred to as the primitive Nth root of unity and can be visually described as rotating about the unit circle when raised to integer powers.

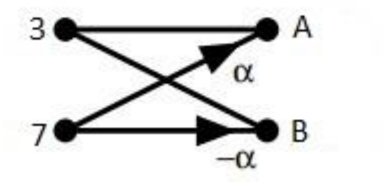
The figure below shows an example where N=8. The primitive root in this case is ${W_8}^1 = e^{\frac{-2\pi i}{8}} = e^{\frac{-1\pi i}{4}}$ which is equivalent to $e^{\frac{-1\pi i}{4}} = \cos\left(-\frac{\pi}{4}\right) + i * \sin\left(-\frac{\pi}{4}\right) = \frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}$ using Euler's identity. For ${W_8}^2$, the primitive root is just to the second power and results in ${W_8}^2 = e^{\frac{-4\pi i}{8}} = e^{\frac{-1\pi i}{2}} = -i$ and so on. When visualizing raising the twiddle factor to an integer power, on the unit circle below a rotation is performed around the unit circle as a factor of the primitive roots angle. ${W_8}^0$ is the starting point located at 1, ${W_8}^1$ is the primitive root located at $\frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}$ and is 45° clockwise of ${W_8}^0$. ${W_8}^2$ is located at i and is 45° clockwise of ${W_8}^1$ and so on. Furthermore, the twiddle factor $W_N = e^{\frac{-2\pi i}{N}}$ was also shown to be periodic allowing integers greater than N to reuse previous values. Continuing from the example below, ${W_8}^0 = {W_8}^8 = 1, {W_8}^1 = {W_8}^9 = \frac{1}{\sqrt{2}} - i\frac{1}{\sqrt{2}}$ and so on. Note that ${W_8}^0 = {W_8}^8 = 1, {W_8}^2 = -\mathrm{i}, {W_8}^4 = -1$, and ${W_8}^6 = \mathrm{i}$. The Nth roots of unity will be used extensively to simplify later calculations of the complex exponential.



Another seemingly unrelated concept that will help to visualize the FFT is a butterfly diagram. Shown below is an example of how a butterfly diagram's operations are performed. Butterfly diagrams are read horizontally in the direction the arrows are pointing towards. In the figure, the arrows are pointing right, so the diagram will be read from left to right. The empty lines with no constants are set to 1 by default. The rightmost points are the results of the butterfly operations which are the respective multiplications and additions where the value at the arrow's origins are multiplied by their respective constants towards their arrow tip. For the top right result, p is multiplied by 1 since the line contains no value and is summed to q multiplied by the present α value. For the bottom right result, q is multiplied by – α and is summed to p multiplied by 1 since no value is present. Butterfly diagrams will be helpful since p, q, and α can each be complex numbers or variables representing large operations. Moreover, butterfly diagrams will help to organize larger computations and equations later on as opposed to writing them all out.



An example of a butterfly diagram calculation is shown below with p=3, q=7, and α=8. From the previously described process, A is calculated as 3 multiplied by the constant 1 since the line is empty and is summed to 7 multiplied by α=8. This results in A=p*1+q*α= 3*1+7*8=3+56=59. B is calculated as 7 multiplied by -α=-8 and is summed to 3 multiplied by the constant 1 since the line is empty. This results in B= p*1+q*(-α)= 3*1+7*(-8)=3-56=-53.



Restatement of the problem and proof.

As previously stated, the Cooley-Tukey FFT algorithm utilizes a divide and conquer process to reduce an N sampled discrete Fourier Transform into two smaller N/2 sized discrete Fourier Transforms and so on [2]. First, the discrete Fourier Transform will be separated into even and odd parts. Next, algebraic manipulations along with some Nth root of unity simplifications will take place reducing both even and odd segments. From this reduction, it will become clear that the N sized discrete Fourier Transform can be expressed as a sum of two smaller N/2 sized discrete Fourier Transforms. Once completed, each of the N/2 DFTs will be able to be separated into even and odd segments again repeating the previous process to obtain four N/4 sized discrete Fourier Transforms. If repeated until only 2 element DFTs remain, the initial N sampled discrete Fourier Transform will be calculated as only additions and multiplications of each element in the original sequence and various twiddle factors with differently raised powers.

From the definition of the DFT shown in equation 1 and below for convenience

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}} = \sum_{n=0}^{N-1} x_n {W_N}^{kn}$$

where k,n∈{0,1,…,N-1}, the expression Cooley and Tukey desired was of the form shown in equation 7.

$$X_k = \sum_{n=0}^{N-1} x_n {W_N}^{kn} = \sum_{n\ even} x_n {W_N}^{kn} + \sum_{n\ odd} x_n {W_N}^{kn} \qquad (7)$$

This is achieved by setting n=2a for the even n values and n=2a+1 for the odd n values where a∈ **Z**∩{0,1,…,N/2-1}. Furthermore, since x_k has N elements, the upper bound of the summations will be replaced by N/2-1. The resulting expression is shown in equation 8.

$$X_k = \sum_{a=0}^{\frac{N}{2}-1} x_{2a} {W_N}^{(2a)k} + \sum_{a=0}^{\frac{N}{2}-1} x_{2a+1} {W_N}^{(2a+1)k} \qquad (8)$$

In this newly rearranged formula, several simplifications can be made. First, ${W_N}^k$ can be factored out in front of the odd element summation and by the associative property (2a)k=2(ak) resulting in equation 9.

$$X_k = \sum_{a=0}^{\frac{N}{2}-1} x_{2a} \left({W_N}^2\right)^{ak} + {W_N}^k \sum_{a=0}^{\frac{N}{2}-1} x_{2a+1} ({W_N}^2)^{ak} \qquad (9)$$

An important observation can be made here regarding ${W_N}^2$. Since W_N is a complex exponential defined as $W_N = e^{\frac{-2\pi i}{N}}$, when this twiddle factor is squared ${W_N}^2 = e^{\frac{-2*2\pi i}{N}}$. This statement is equivalent to $W_{\frac{N}{2}}$ if N was replaced by N/2 in the original equation since ${W_N}^2 = e^{\frac{-2*2\pi i}{N}} = e^{\frac{-2\pi i}{\left(\frac{N}{2}\right)}} = W_{\frac{N}{2}}$. Replacing ${W_N}^2$ with $W_{\frac{N}{2}}$ yields equation 10.

$$X_k = \sum_{a=0}^{\frac{N}{2}-1} x_{2a} {W_{\frac{N}{2}}}^{ak} + {W_N}^k \sum_{a=0}^{\frac{N}{2}-1} x_{2a+1} {W_{\frac{N}{2}}}^{ak} \tag{10}$$

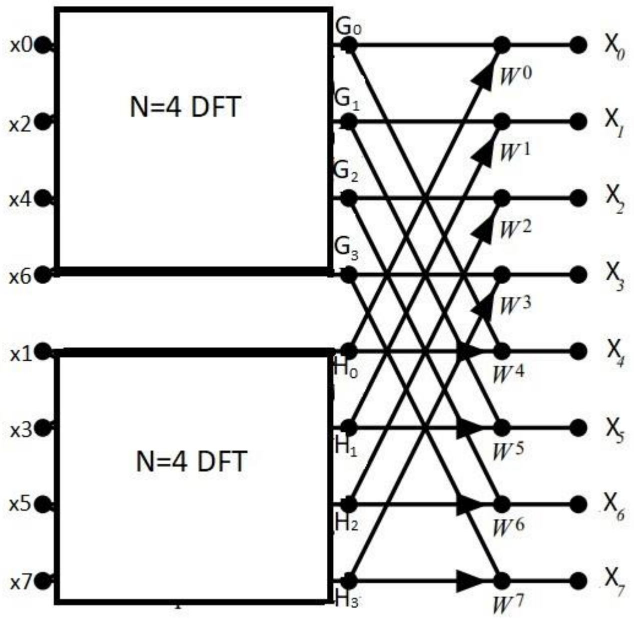
This final expression can draw several conclusions. First, both summations are of the DFT form with N/2 elements instead of N and the original x_k elements are replaced with x_{2a} for the even elements and x_{2a+1} for the odd elements. Moreover, ${W_N}^k$ is a complex constant that can be immediately calculated for the current element of X_k being solved for with k. This allows the final equation to be rewritten as equation 11

$$X_k = G_k + {W_N}^k H_k \tag{11}$$

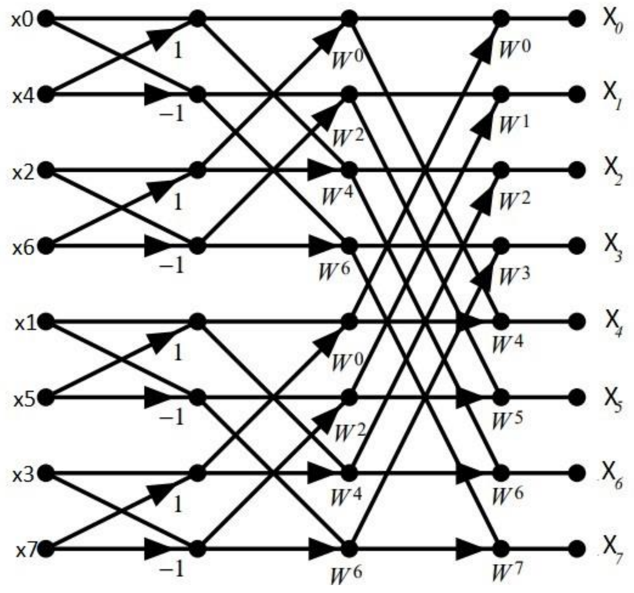
where G_k is the DFT of the even elements of x_k and H_k is the DFT of the odd elements of x_k both with N/2 elements. Cooley and Tukey came to the realization that a DFT, if N is a power of two, can be continuously reduced into smaller and smaller DFTs until only 2 element DFTs remain.

This process can be visualized easiest by utilizing butterfly diagrams as previously described. For example, let N=8 and the 8 element DFT is separated into its even and odd components. The result of this process is shown below with two 4 element DFTs present. Note for the final 8 element DFT, the butterfly diagram states that $X_0 = G_0 + {W_8}^0 H_0$ which is correct according to the equation 11.

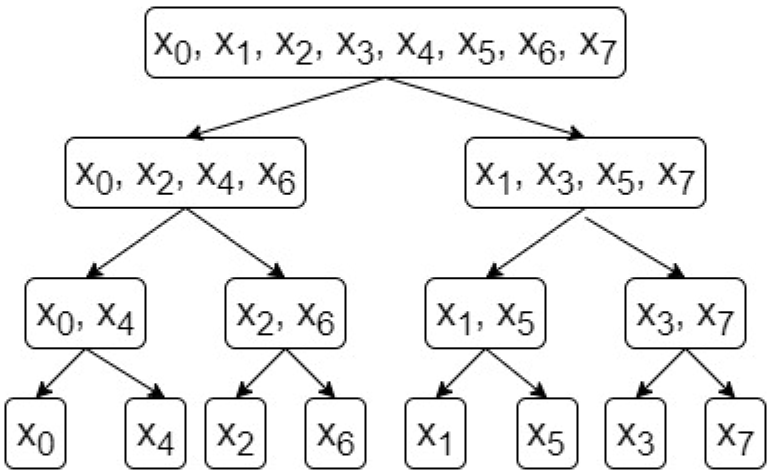
[redacted]



This process can be repeated until there are four 2 element DFTs which can easily be solved using equation 11 as simple sums and products of the initial sequence and ${W_N}^k$ values. Shown below is the final butterfly diagram including all constants when N=8.

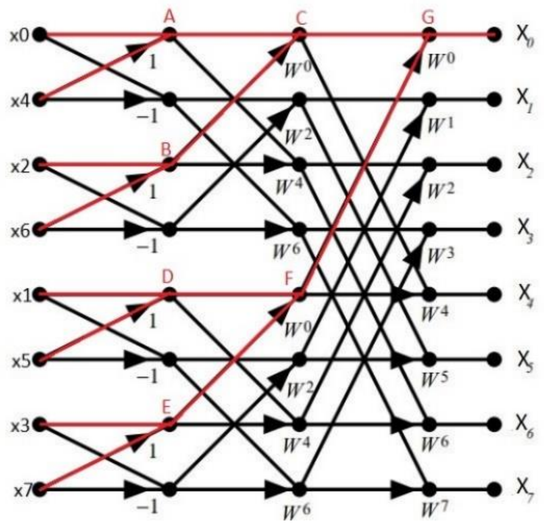


A consequence of the divide and conquer approach is shown in the image above where the original sequence is in the order of x_0, x_4, x_2, x_6, x_1, x_5, x_3, x_7. This occurs when the DFT is separated into its even and odd parts [3]. For the first iteration of creating two N=4 element DFTs, the original sequence moves the even elements to the left and odd elements to the right resulting in x_0, x_2, x_4, x_6 and x_1, x_3, x_5, x_7. The second iteration moves the even elements of the N=4 sized DFTs to the left and odd elements to the right creating four N=2 sized DFTs resulting in x_0, x_4 and x_2, x_6 and x_1, x_5 and x_3, x_7. The process is shown below for N=8.



To illustrate how to use the final butterfly diagram, the zeroth element of the DFT X_0 will be solved for. Shown below is a butterfly diagram for the N=8 example and will be solved along the red lines using the previously described process. To solve for X_0, the previous nodes C and F must be calculated which need nodes A, B, D, and E. To calculate A, x_0 is multiplied by 1 since the line has no constant and is added to x_4 multiplied by 1 resulting in $A=x_0+x_4$. To calculate B, x_2 is multiplied by 1 since the line has no constant and is added to x_6 multiplied by 1 resulting in $B=x_2+x_6$. With nodes A and B now calculated, node C can be computed as A multiplied by 1 since the line is empty added to B multiplied by ${W_8}^0 = 1$. This results in $C=A+{W_8}^0B=A+B=x_0+x_4+x_2+x_6$. The same process can be repeated to solve for nodes D, E, and G resulting in $D=x_1+x_5$, $E=x_3+x_7$, and $F=D+{W_8}^0E=D+E=x_1+x_5+ x_3+x_7$. To solve for X_0, the process is repeated once more using nodes C and node F. Node C is multiplied by 1 since the line has no constant and is summed to node F multiplied by ${W_8}^0$ which results in

$X_0 = C + {W_8}^0 F = C + F = x_0 + x_4 + x_2 + x_6 + x_1 + x_5 + \ x_3 + x_7.$



In the DFT element X_0, it can be observed that each element of the original sequence is present in the final result. X_0 was selected for demonstration purposes to show how a butterfly diagram can be used to calculate a single DFT element. When calculating the other seven DFT elements, the same process can be repeated. The only complications arise when the twiddle factor is not an integer or directly i or -i. The desired DFT element may be simple to calculate, but difficult to write in its entirety. For this reason, butterfly diagrams are useful as a means to describe the process of calculating each individual DTF term in a simple image as opposed to writing out long equations.

For implementation purposes, typically the indexes of the original sequence are bit-reversed to obtain the required order for the Cooley-Tukey FFT. The table shown below considers the N=8 example. First, each index value of k is converted into its binary representation. For example, 2^2=4, 2^1=2, and 2^0=1, so to write 7 from base 10 to base

[redacted]

2(binary), the sum is recreated from its binary digits one by one. Since $7=1*2^2+1*2^1+1*2^0$, then 7 in base 10 converted to binary is 111.

Index (k)	Binary	Bit-reversed Binary	Bit-reversed Index
0	000	000	0
1	001	100	4
2	010	010	2
3	011	110	6
4	100	001	1
5	101	101	5
6	110	011	3
7	111	111	7

Now that the Cooley-Tukey FFT has been defined and explained in a step-by-step process of how it works, the next step is to define and derive the inverse Fast Fourier Transform (IFFT). Unlike the FFT which needed extensive manipulations to simplify the final calculations, the IFFT can be calculated by using the FFT and previously defined complex conjugate properties.

The inverse discrete Fourier Transform with N samples is defined in equation 12 as

$$x_n = \frac{1}{N}\sum_{k=0}^{N-1} X_k e^{\frac{2\pi ikn}{N}}. \tag{12}$$

When compared to the DFT shown below for convenience, it can be observed that the only differences between the DFT and IDFT is the exponential sign is different and the IDFT is divided by a factor of N.

$$X_k = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}}$$

Starting with the DFT, the complex conjugate of $e^{\frac{-2\pi ikn}{N}}$ is $e^{\frac{2\pi ikn}{N}}$. For simplicity, the notation conj() will be used to denote the complex conjugate. This allows the DFT to be rewritten as equation 13.

$$X_k = DFT(x_n) = \sum_{n=0}^{N-1} x_n e^{\frac{-2\pi ikn}{N}} = \sum_{n=0}^{N-1} x_n conj(e^{\frac{2\pi ikn}{N}}) \tag{13}$$

Taking the complex conjugate of x_n and using the previously defined property conj(z_1 z_2)=conj(z_1)conj(z_2), the above equation can be rewritten as equation 14.

$$DFT(conj(x_n)) = \sum_{n=0}^{N-1} conj(x_n e^{\frac{2\pi ikn}{N}}) = \sum_{n=0}^{N-1} conj(x_n)conj(e^{\frac{2\pi ikn}{N}}) \tag{14}$$

Another complex conjugate property conj(z_1 +z_2)=conj(z_1)+conj(z_2) can be used to factor the conjugate operation out of the summation in equation 14 resulting in equation 15.

$$DFT(conj(x_n)) = \sum_{n=0}^{N-1} conj(x_n e^{\frac{2\pi ikn}{N}}) = conj(\sum_{n=0}^{N-1} x_n e^{\frac{2\pi ikn}{N}}) \tag{15}$$

It can be observed that the right side of equation 15 is nearly identical to the definition of the IDFT supplied in equation 12 besides a division of N. Substituting this value yields equation 16.

$$DFT(conj(x_n)) = conj(\sum_{n=0}^{N-1} x_n e^{\frac{2\pi ikn}{N}}) = conj(N * IDFT) \tag{16}$$

Since N is a real number, it can be factored out of the conjugate operation in equation 16 and N can be divided on both sides resulting in equation 17.

$$\frac{1}{N} DFT(conj(x_n)) = conj(IDFT) \tag{17}$$

[redacted]

To cancel the conjugate operation on the right side of equation 17, the conjugate operation will be performed once more on both sides and since $\frac{1}{N}$ is a real number, it can be factored out of the conjugate resulting in equation 18.

$$\frac{1}{N}conj(DFT(conj(X_k))) = IDFT \tag{18}$$

Equation 18 shows that the IDFT can be calculated using the DFT and complex conjugate operation. Since the DFT has been shown to be accurately calculated with the FFT, then the IFFT can be calculated as shown in equation 19 by taking the complex conjugate of the given DFT sequence, performing the FFT on the result, and taking the complex conjugate once more and dividing by N for all elements, yielding the IDFT.

$$IFFT = \frac{1}{N}conj(FFT(conj(X_k))) \tag{19}$$

In order to compare the DFT and FFT computation speed, a concept called big O notation will be used. Big O notation is used to compare and communicate how fast an algorithm is. Typically, the order of a function is a notation for the asymptotic upper bound. For example, in O(1) the order is constant because 1 is not a function of N and will result in the same time, every time, to compute the function. In O(N), the order is linear since the time to compute the function will depend on the size of N and its complexity will increase in a linear trend as N increases. In O(N^2), the order is quadratic since the time to compute the function will depend on the size of N^2. Comparing O(N) and O(N^2), for low values of N the difference is unnoticeable but for sufficiently large N values, the difference between the two becomes quickly realized. If $N=10^9$, then an O(N) function would have an upper bound of 10^9 while the quadradic O(N^2) would have an upper bound of 10^{18} which is considerably greater than 10^9. For this reason, it is preferred to select algorithms with a lower complexity order.

The original DFT equation was found to consist of N complex multiplications and N complex additions resulting in an order of O(N^2). The Cooley-Tukey FFT is of the form

$$X_k = G_k + {W_N}^k H_k$$

Where G_k and H_k are the even and odd indexed DFTs of X_k both with N/2 elements. It is also known that N is a power of two so $N=2^L$ where $L \geq 0$ and $L \in \mathbf{Z}$. This results in the Cooley-Tukey FFT algorithm having a complexity of O($N\log_2 N$).

The main reason the FFT separates itself from the discrete Fourier Transform is its computationally efficient way to calculate the DFT. From big O notation, the complexity of the original DFT equation is O(N^2) due to the N complex multiplications and N complex additions while Cooley-Tukey FFT was proved to have O($N\log_2 N$) for a sufficiently large N. The table shown below illustrates the complexity for different cases of N. This is relevant because of the real-world timing implications to calculate the extensively used DFT. Suppose a complex operation takes 1 nanosecond to complete. For the case where $N=10^9$, the direct DFT equation would take $10^{18} operations * \frac{1\ second}{10^9\ operations} * \frac{1 hour}{3600\ seconds} * \frac{1\ day}{24\ hours} * \frac{1\ year}{365\ days} = 31.7$ years while the Cooley-Tukey FFT would take $30 * 10^9 operations * \frac{1\ second}{10^9\ operations} = 30$ seconds.

N	N^2	$N\log_2 N$
10^3	10^6	$9965.8 \approx 10^4$
10^6	10^{12}	$1.993*10^7 \approx 20*10^6$
10^9	10^{18}	$29.897*10^9 \approx 30*10^9$

Application

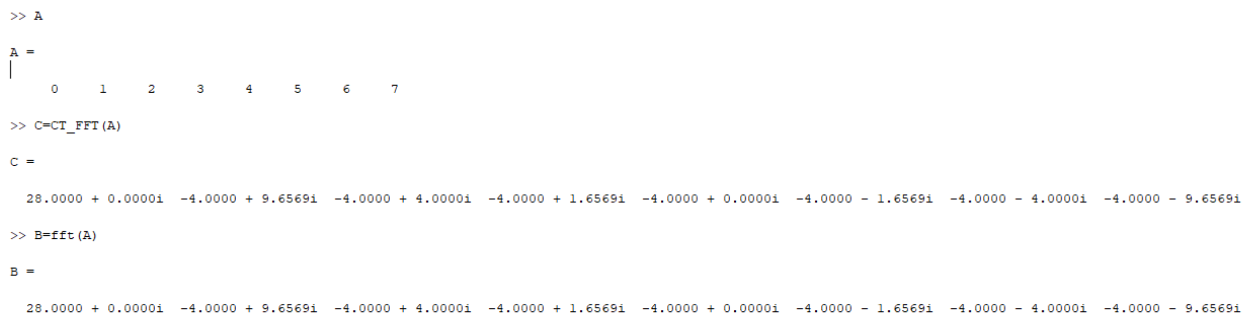
Fast Fourier Transform Program

For the sake of convenience, the MATLAB programing language will be used to create a FFT and IFFT algorithm that will implemented to remove noise from an audio signal. MATLAB has a built in FFT function which will only be used for verification purposes. Moreover, all signals will be made to have a power of two elements within the main function. All code is attached in the appendix.

For the FFT function CT_FFT, the input x will be a vector with a size that is a power of two and the output X will be a vector with the same size. First, N will be calculated as the length of vector x. Next, an if condition will be introduced with the condition that if N=1, then the program will assign the only element in x as the output vector X and will return this vector. This condition will assure the program will run recursively such that the first instance will try to calculate the DFT for N samples, but this requires the program to calculate the DFT of N/2 samples for the second instance and so on. This process will repeat until the DFT is only composed of one element which is itself and will be returned allowing the remaining DFTs to be calculated. Next, the even and odd indexed elements will be stored in the vectors even and odd, respectively. Once completed, the program will execute the CT_FFT function on both even and odd vectors as previously described. Moreover, a for-loop was introduced that will cycle a loop control variable i from 1 to N/2. Within the loop, the twiddle factor will be calculated as ${W_N}^{kn} = e^{\frac{-2\pi kni}{N}}$, and the DFTs of the even and odd elements will be used to calculate the elements of the output X. From the previously described butterfly diagrams, the equation to calculate the even elements of X is the even node summed to the product of the odd node and the twiddle factor. For the odd elements of X, the even node is summed to the product of the odd node and a negative twiddle factor.

For the IFFT function CT_IFFT, the input X will be a vector with a size that is a power of two and the output x will be a vector with the same size. First, N will be calculated as the length of vector X and the output vector x will be created with the same length N. Following the order to calculate the IDFT shown in equation 19, a for-loop was used to take the complex conjugate of every element in X and store it in x with the same indices. Next, the FFT function created called CT_FFT was used on the vector x containing the complex conjugates. Moreover, a for-loop was used again to take the complex conjugate of every element in x once more and each element was divided by N.

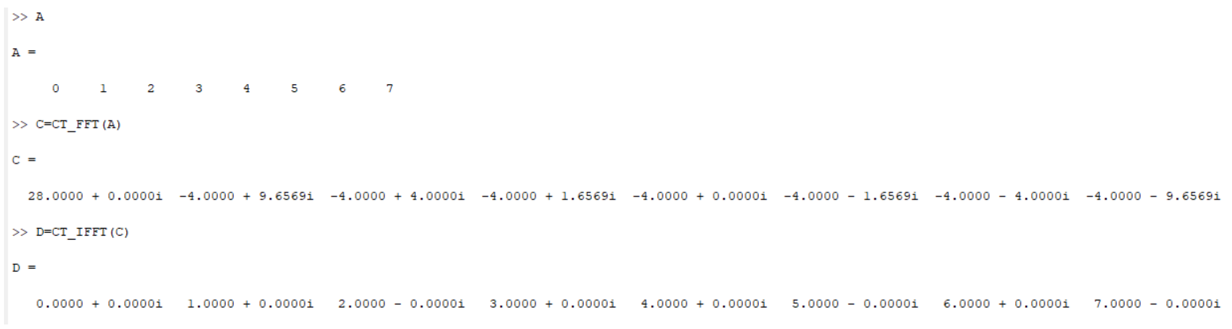
To test the code, an 8-element vector was created as A=[0,1,2,3,4,5,6,7] where the zeroth element was 0, first element was 1, second element was 2, and so on. The result of C=CT_FFT(A) is shown below and when compared to the MATLAB DFT computation B=fft(A), they were found to be identical.



```
>> A

A =

     0     1     2     3     4     5     6     7

>> C=CT_FFT(A)

C =

  28.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i  -4.0000 + 0.0000i  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i

>> B=fft(A)

B =

  28.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i  -4.0000 + 0.0000i  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i
```

To test the IFFT function CT_IFFT, the function was executed on the previously calculated DFT vector called C. Shown below is the result and it can be observed that D=CT_IFFT(C) is identical to the initial 8-element vector A. With both functions now verified, a noise removal signal processing application can be demonstrated [4].

[redacted]



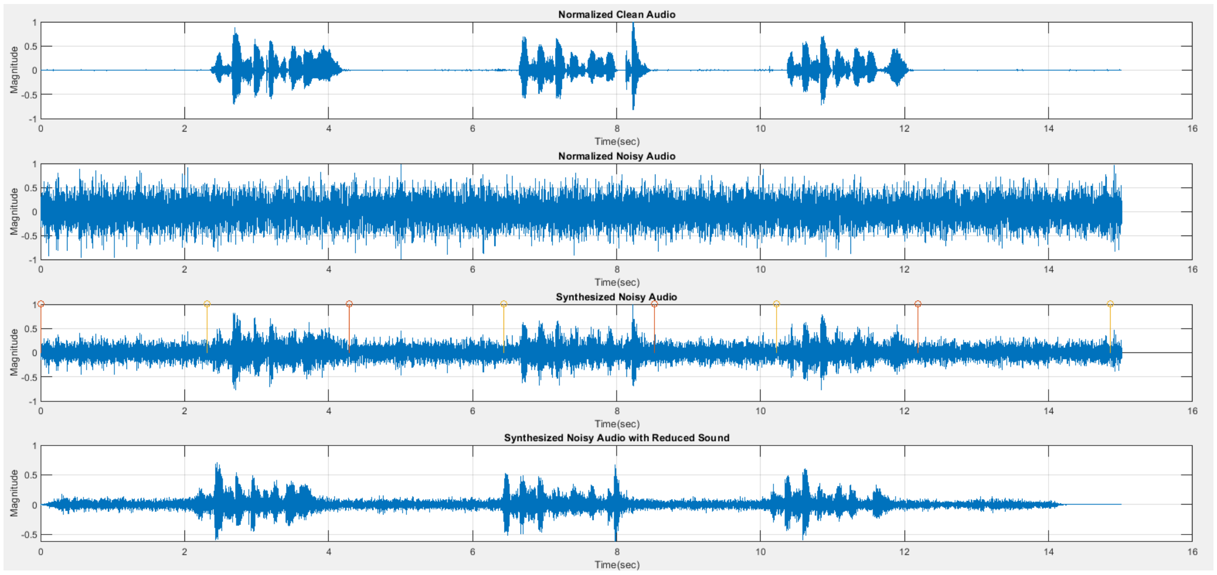
```
>> A

A =

     0     1     2     3     4     5     6     7

>> C=CT_FFT(A)

C =

  28.0000 + 0.0000i  -4.0000 + 9.6569i  -4.0000 + 4.0000i  -4.0000 + 1.6569i  -4.0000 + 0.0000i  -4.0000 - 1.6569i  -4.0000 - 4.0000i  -4.0000 - 9.6569i

>> D=CT_IFFT(C)

D =

   0.0000 + 0.0000i   1.0000 + 0.0000i   2.0000 - 0.0000i   3.0000 + 0.0000i   4.0000 + 0.0000i   5.0000 - 0.0000i   6.0000 + 0.0000i   7.0000 - 0.0000i
```

The result of the noise removing application is shown below. The top graph is the clean

[redacted]

[redacted]

graphs but when comparing both audio segments by ear, there is no question that the noise has been reduced.



[redacted]

Conclusion

All in all, the Fast Fourier Transform is a computationally efficient and fast method to calculate the discrete Fourier Transform of a signal with N samples. Despite the lowered calculation complexity, it was still impractical to utilize the FFT algorithm and this result went largely unnoticed until the fast-paced developments of the digital computer in the 20th century. Throughout the course of the paper, the Cooley-Tukey FFT algorithm was derived from the original discrete time Fourier Transform to the sample based discrete Fourier Transform and finally led to the Cooley-Tukey FFT discrete Fourier Transform equation. The divide and conquer method from the Cooley-Tukey FFT was found to have a complexity of $O(N\log_2 N)$ as opposed to the direct discrete Fourier Transform complexity of $O(N^2)$.

Fourier analysis is important for many applications spanning a variety of fields including signal processing, software-defined radio, control theory, etc. As technology begins to integrate machine learning processes and artificial intelligence, there is no doubt that the Fast Fourier Transform will remain relevant due to its fast computation speed.

Bibliography

[1] Burrus, Sidney C., Frigo, et al. *Fast Fourier Transforms*. OpenStax CNX, 2012.

[2] “The Scientist and Engineer's Guide To Digital Signal Processing By Steven W. Smith, Ph.D.” *How the FFT Works*, www.dspguide.com/ch12/2.htm.

[3] Proakis, John G., and Dimitris G. Manolakis. *Digital Signal Processing*. Pearson Education Limited, 2007.

[4] Osgood, Brad. *Lecture Notes for EE 261: The Fourier Transform and its Applications*. Stanford University, 2007.