Note 04: Conversion Into Frequency Domain | DFT | FFT

Author: Sadat Rafi
https://www.linkedin.com/in/sadat-rafi-7855a2a5/

Introduction

The discrete Fourier transform of a discrete-time domain signal is defined as:

$$X_K = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N}kn}$$

Where, X_K is the frequency domain representation of x_n and $k = 0,1,2, \dots, N-1$. For digital signal processing, the absolute values are typically required. So, the absolute values can be calculated as shown below:

$$X_K = \sum_{n=0}^{N-1} x_n \cdot e^{-\frac{2\pi i}{N}kn} \quad \dots\dots (01)$$

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

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

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

$$\Rightarrow |X_K| = \sqrt{\left(\sum_{n=0}^{N-1} x_n \cos\left(\frac{2\pi}{N}kn\right)\right)^2 + \left(\sum_{n=0}^{N-1} x_n \sin\left(\frac{2\pi}{N}kn\right)\right)^2} \quad \dots\dots (02)$$

Let's check a GNU Octave code to visualize what exactly this has done. Octave code is 99% similar to Matlab. I'm avoiding Matlab so that everyone can try this code.

```
N = 50;
t = 0.01*[0:N-1];
f = 10;  %Hz

x = sin(2*pi()*f*t);
figure 01;
stem([0:N-1],x,'b');

X_real = 0;
X_imaginary = 0;
step=0;
for k=0:N-1
```

```
        for n=0:N-1
                X_real = X_real + (x(n+1)* (cos((2*pi()/N)*k*n)));
                X_imaginary = X_imaginary + (x(n+1)* (sin((2*pi()/N)*k*n)));
                step ++;
        end
        X(k+1) = (X_real^2 + X_imaginary^2)^0.5;
        X_real = 0;
        X_imaginary = 0;
end

figure 02;
stem([0:N-1],X,'b');
```

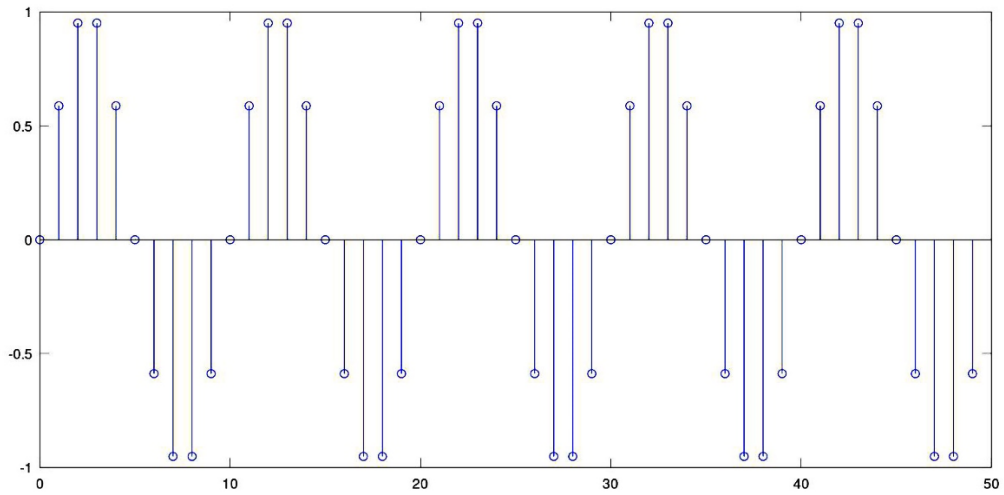

Figure 01: 50 samples of 10 Hz signal having a sampling frequency of 100 Hz.

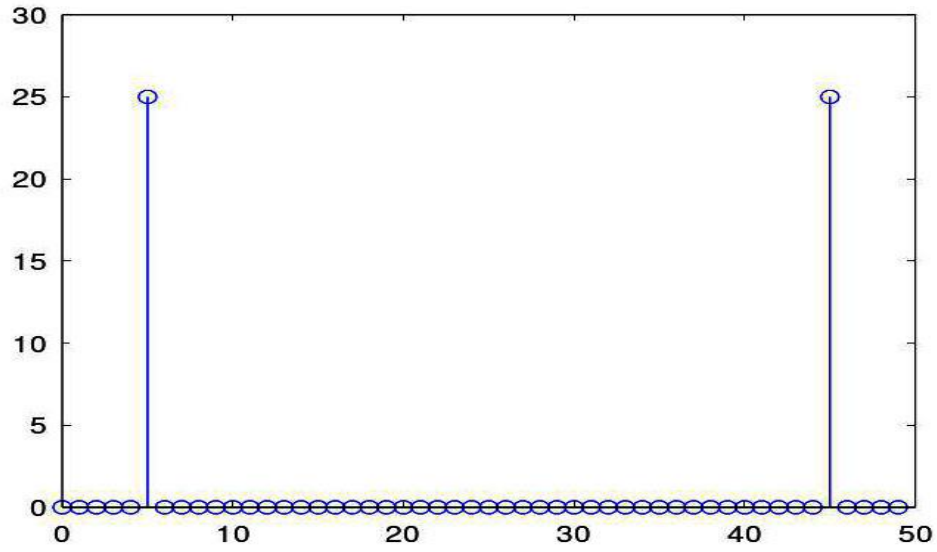

Figure 02: The DFT sequence of the time domain signal of figure 01

In figure 02, there is a large peak at k=5, having a value of 25. 'k' represents a frequency, which can be obtained by multiplying it by the step size defined as:

$$f_{step} = \frac{Sampling\ frequency}{Number\ of\ samples} = \frac{100}{50} = 2\ Hz$$

So, k=5 represents

$$k \cdot f_{step} = 5 \cdot 2\ Hz = 10\ Hz$$

And the amplitude of the signal that is represented by k=5 is:

$$\frac{25}{\left({}^{N}/_{2}\right)} = \frac{25}{\left({}^{50}/_{2}\right)} = 1$$

Thus the 10 Hz signal having an amplitude of 1 unit of figure 01 is converted into a frequency-domain signal using equation 02.

However, the process consumes a tremendous amount of clock cycles as the program looped 2500 times, and each time it has to go through two trigonometric functions and some multiplications. This large amount of clock cycle can be minimized using some algorithms commonly known as FFT.

The radix-2 DIT FFT

Equation 1 can be split into two parts based on the odd and even-numbered samples.

$$X_K = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cdot e^{-\frac{2\pi i}{N}(2n)k} + \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cdot e^{-\frac{2\pi i}{N}(2n+1)k}$$

$$\Rightarrow X_K = \sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cdot e^{-\frac{2\pi i}{N}(2n)k} + e^{-\frac{2\pi i}{N}k} \sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cdot e^{-\frac{2\pi i}{N}(2n)k}$$

$$\Rightarrow X_K = \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x_{2n} \cdot e^{-\frac{2\pi i}{\frac{N}{2}}nk}}_{\text{DFT of Even Samples}} + e^{-\frac{2\pi i}{N}k} \underbrace{\sum_{n=0}^{\frac{N}{2}-1} x_{2n+1} \cdot e^{-\frac{2\pi i}{\frac{N}{2}}nk}}_{\text{DFT of Odd Samples}} \quad \ldots\ldots (03)$$

$$\Rightarrow X_K = E_K + \left(e^{-\frac{2\pi i}{N}k}\right) O_K$$

Because of the periodicity of the complex exponentials $X_{K+\frac{N}{2}}$ can also be computed from X_K.

$$X_{K+\frac{N}{2}} = E_K - \left(e^{-\frac{2\pi i}{N}k}\right) O_K = E_K - \left(\cos\left(\frac{2\pi}{N}k\right) + i \sin\left(\frac{2\pi}{N}k\right)\right) O_K$$

Now the question arises that how does this improving operating speed. To understand that, have a closer look at the blue segment of equation 03, which computes the DFT of even-numbered samples. For that, compute the DFT components up to $n = \frac{N}{4} - 1$ and copy and paste the values to find the DFT components of the remaining points. Thus a tremendous amount of trigonometric calculation has been avoided.

As there are many efficient library functions on FFT, I'm not going to elaborate on this topic. It's wise to use any of those libraries for practical purposes unless you are highly interested in making FFT more efficient.