University of Rhode Island Department of Electrical and Computer Engineering

ELE 436: Communication Systems

# FFT Tutorial

## 1 Getting to Know the FFT

**What is the FFT?** FFT = Fast Fourier Transform. The FFT is a faster version of the Discrete Fourier Transform (DFT). The FFT utilizes some clever algorithms to do the same thing as the DTF, but in much less time.

**Ok, but what is the DFT?** The DFT is extremely important in the area of frequency (spectrum) analysis because it takes a discrete signal in the time domain and transforms that signal into its discrete frequency domain representation. Without a discrete-time to discrete-frequency transform we would not be able to compute the Fourier transform with a microprocessor or DSP based system.

It is the speed and discrete nature of the FFT that allows us to analyze a signal's spectrum with Matlab or in real-time on the SR770

## 2 Review of Transforms

**Was the DFT or FFT something that was taught in ELE 313 or 314?** No. If you took ELE 313 and 314 you learned about the following transforms:

| | | |
|---|---|---|
| Laplace Transform: | $x(t) \Leftrightarrow X(s)$ | where $X(s) = \int_{-\infty}^{\infty} x(t)e^{-st}dt$ |
| Continuous-Time Fourier Transform: | $x(t) \Leftrightarrow X(j\omega)$ | where $X(j\omega) = \int_{-\infty}^{\infty} x(t)e^{-j\omega t}dt$ |
| z Transform: | $x[n] \Leftrightarrow X(z)$ | where $X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$ |
| Discrete-Time Fourier Transform: | $x[n] \Leftrightarrow X(e^{j\Omega})$ | where $X(e^{j\Omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-j\Omega n}$ |

The **Laplace transform** is used to to find a pole/zero representation of a continuous-time signal or system, $x(t)$, in the s-plane. Similarly, The **z transform** is used to find a pole/zero representation of a discrete-time signal or system, $x[n]$, in the z-plane.

The continuous-time Fourier transform **(CTFT)** can be found by evaluating the Laplace transform at $s = j\omega$. The discrete-time Fourier transform **(DTFT)** can be found by evaluating the z transform at $z = e^{j\Omega}$.

# 3 Understanding the DFT

**How does the discrete Fourier transform relate to the other transforms?** First of all, the DFT is NOT the same as the DTFT. Both start with a discrete-time signal, but the DFT produces a *discrete* frequency domain representation while the DTFT is *continuous* in the frequency domain. These two transforms have much in common, however. It is therefore helpful to have a basic understanding of the properties of the DTFT.

**Periodicity:** The DTFT, $X(e^{j\Omega})$, is periodic. One period extends from $f = 0$ to $f_s$, where $f_s$ is the sampling frequency. Taking advantage of this redundancy, The DFT is only defined in the region between 0 and $f_s$.

**Symmetry:** When the region between 0 and $f_s$ is examined, it can be seen that there is even symmetry around the center point, $0.5f_s$, the Nyquist frequency. This symmetry adds redundant information. Figure 1 shows the DFT (implemented with Matlab's FFT function) of a cosine with a frequency one tenth the sampling frequency. Note that the data between $0.5f_s$ and $f_s$ is a mirror image of the data between 0 and $0.5f_s$.

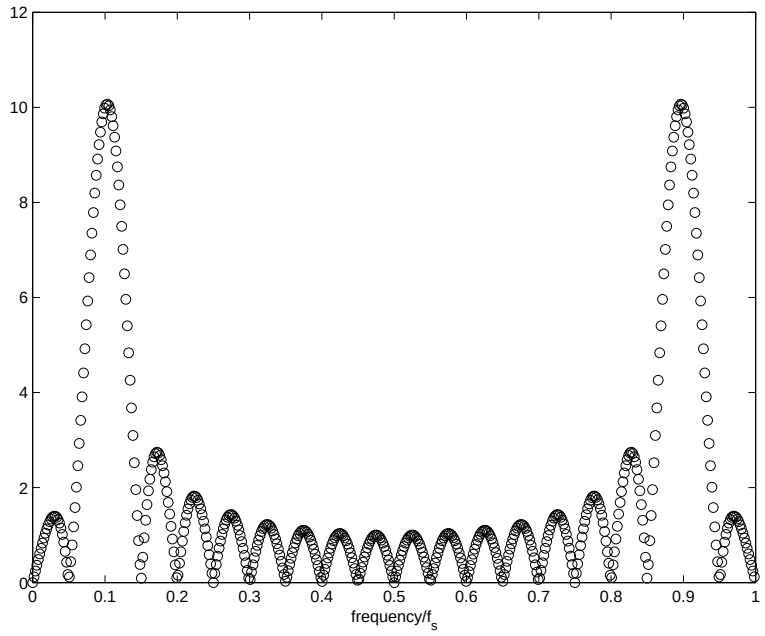


Figure 1: Plot showing the symmetry of a DFT

# 4 Matlab and the FFT

Matlab's FFT function is an effective tool for computing the discrete Fourier transform of a signal. The following code examples will help you to understand the details of using the FFT function.

**Example 1:** The typical syntax for computing the FFT of a signal is FFT(x,N) where x is the signal, $x[n]$, you wish to transform, and N is the number of points in the FFT. N must be at least as large as the number of samples in $x[n]$. To demonstrate the effect of changing the value of N, sythesize a cosine with 30 samples at 10 samples per period.

```
n = [0:29];
x = cos(2*pi*n/10);
```

Define 3 different values for N. Then take the transform of $x[n]$ for each of the 3 values that were defined. The *abs* function finds the magnitude of the transform, as we are not concered with distinguishing between real and imaginary components.

```
N1 = 64;
N2 = 128;
N3 = 256;
X1 = abs(fft(x,N1));
X2 = abs(fft(x,N2));
X3 = abs(fft(x,N3));
```

The frequency scale begins at 0 and extends to $N-1$ for an N-point FFT. We then normalize the scale so that it extends from 0 to $1-\frac{1}{N}$.

```
F1 = [0 :  N1 - 1]/N1;
F2 = [0 :  N2 - 1]/N2;
F3 = [0 :  N3 - 1]/N3;
```

Plot each of the transforms one above the other.

```
subplot(3,1,1)
plot(F1,X1,'-x'),title('N = 64'),axis([0 1 0 20])
subplot(3,1,2)
plot(F2,X2,'-x'),title('N = 128'),axis([0 1 0 20])
subplot(3,1,3)
plot(F3,X3,'-x'),title('N = 256'),axis([0 1 0 20])
```

Upon examining the plot (shown in figure 2) one can see that each of the transforms adheres to the same shape, differing only in the number of samples used to approximate that shape. What happens if N is the same as the number of samples in $x[n]$? To find out, set N1 = 30. What does the resulting plot look like? Why does it look like this?

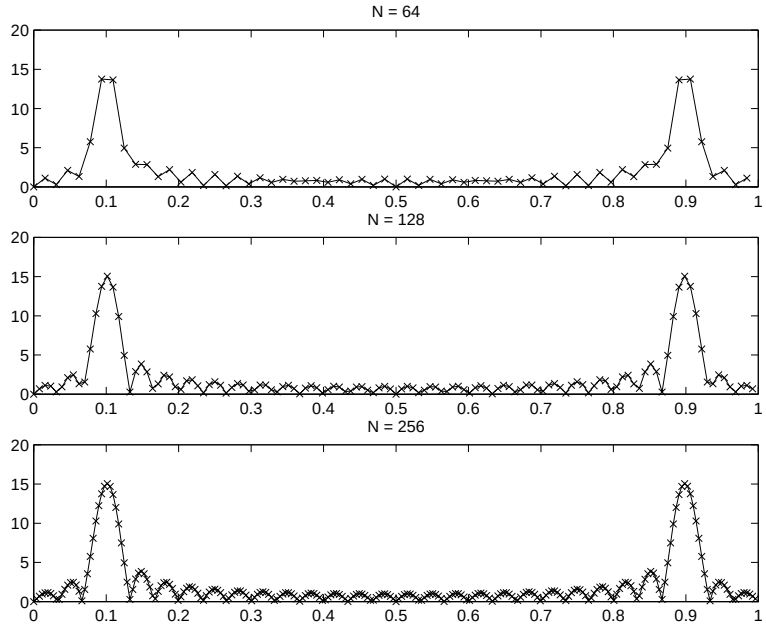


Figure 2: FFT of a cosine for N = 64, 128, and 256

**Example 2:** In the the last example the length of $x[n]$ was limited to 3 periods in length. Now, let's choose a large value for N (for a transform with many points), and vary the number of repetitions of the fundamental period.

```
n = [0:29];
x1 = cos(2*pi*n/10); % 3 periods
x2 = [x1 x1]; % 6 periods
x3 = [x1 x1 x1]; % 9 periods

N = 2048;

X1 = abs(fft(x1,N));
X2 = abs(fft(x2,N));
X3 = abs(fft(x3,N));

F = [0:N-1]/N;

subplot(3,1,1)
plot(F,X1),title('3 periods'),axis([0 1 0 50])
subplot(3,1,2)
plot(F,X2),title('6 periods'),axis([0 1 0 50])
subplot(3,1,3)
plot(F,X3),title('9 periods'),axis([0 1 0 50])
```

The previous code will produce three plots. The first plot, the transform of 3 periods of a cosine, looks like the magnitude of 2 sincs with the center of the first sinc at $0.1f_s$ and the second at $0.9f_s$.

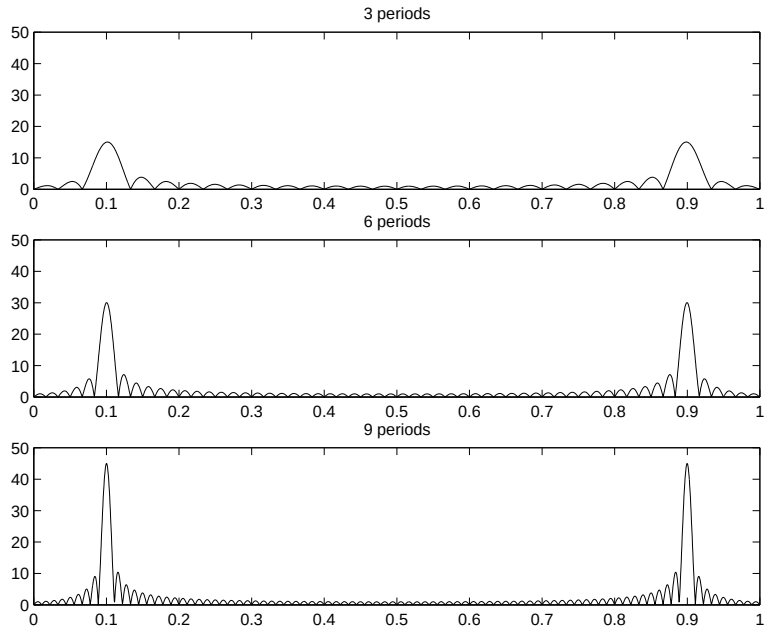


Figure 3: FFT of a cosine of 3, 6, and 9 periods

The second plot also has a sinc-like appearance, but its frequency is higher and it has a larger magnitude at $0.1f_s$ and $0.9f_s$. Similarly, the third plot has a larger sinc frequency and magnitude. As $x[n]$ is extended to an large number of periods, the sincs will begin to look more and more like impulses.

**But I thought a sinusoid transformed to an impulse, why do we have sincs in the frequency domain?** When the FFT is computed with an N larger than the number of samples in $x[n]$, it fills in the samples after $x[n]$ with zeros. Example 2 had an $x[n]$ that was 30 samples long, but the FFT had an N = 2048. When Matlab computes the FFT, it automatically fills the spaces from n = 30 to n = 2047 with zeros. This is like taking a sinusoid and mulitipying it with a rectangular box of length 30. A multiplication of a box and a sinusoid in the time domain should result in the convolution of a sinc with impulses in the frequency domain. Furthermore, increasing the width of the box in the time domain should increase the frequency of the sinc in the frequency domain. The previous Matlab experiment supports this conclusion.

## 5 Spectrum Analysis with the FFT and Matlab

The FFT does not directly give you the spectrum of a signal. As we have seen with the last two experiments, the FFT can vary dramatically depending on the number of points (N) of the FFT, and the number of periods of the signal that are represented. There is another problem as well.

The FFT contains information between 0 and $f_s$, however, we know that the sampling frequency

must be at least twice the highest frequency component. Therefore, the signal's spectrum should be entirly below $\frac{f_s}{2}$, the Nyquist frequency.

Recall also that a real signal should have a transform magnitude that is symmetrical for for positive and negative frequencies. So instead of having a spectrum that goes from 0 to $f_s$, it would be more appropriate to show the spectrum from $\frac{-f_s}{2}$ to $\frac{f_s}{2}$. This can be accomplished by using Matlab's *fftshift* function as the following code demonstrates.

```
n = [0:149];
x1 = cos(2*pi*n/10);

N = 2048;

X = abs(fft(x1,N));
X = fftshift(X);

F = [-N/2:N/2-1]/N;

plot(F,X),
xlabel('frequency / f_s')
```

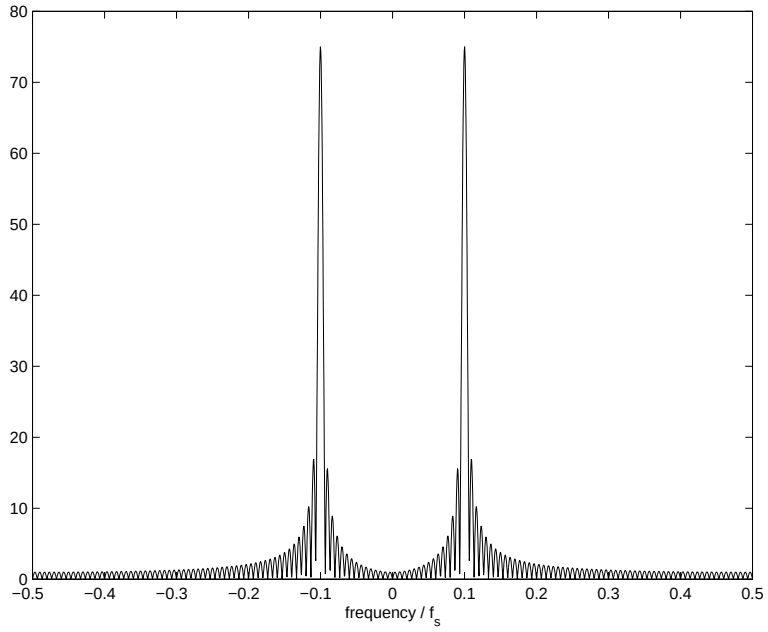


Figure 4: Approximate Spectrum of a Sinusoid with the FFT