MATLAB Addendum on Fourier Stuff

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 DFT,
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-frequency transform we would not be able to
compute the Fourier transform with a microprocessor based
system.

It is the speed and discrete nature of the FFT that allows us to
analyze a signal's spectrum with MATLAB.

2. Review of Transforms

The useful transforms are the following:

Laplace Transform: $x(t) \Leftrightarrow X(s)$ $where$ $X(s) = \int_{-\infty}^{\infty} x(t)e^{-st}dt$

Continuous-Time Fourier Transform:

$$x(t) \Leftrightarrow X(i\omega)   where   X(i\omega) = \int_{-\infty}^{\infty} x(t)e^{-i\omega t}dt$$

z Transform: $x[n] \Leftrightarrow X(z)$ $where$ $X(z) = \sum_{n=-\infty}^{\infty} x[n]z^{-n}$

Discrete-Time Fourier Transform:

$$x[n] \Leftrightarrow X(i\Omega)   where   X(e^{i\Omega}) = \sum_{n=-\infty}^{\infty} x[n]e^{-i\Omega n}$$

The **Laplace transform** is used to find a pole/zero representation
of a continuous-time signal or system, x(t), in the s-plane. 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-tine Fourier transform (CTFT) can be found by
evaluating the Laplace transform at $s = i\omega$. The discrete-time
Fourier transform (DTFT) can be found by evaluating the z
transform at $z = e^{i\Omega}$.

3. Understanding the DFT

How does the discrete Fourier transform relate to 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 $X(e^{i\Omega})$, is periodic. One period extends from $f = 0$ to $f = f_s$, 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, $0.5 f_s$, the Nyquist frequency. This symmetry adds redundant information. Figure 1 shows the DFT(using MATLAB's FFT function) of a cosine with a frequency 1/10 the sampling frequency. Note that the data between $0.5 f_s$ and f_s is a mirror image of the data between 0 and $0.5 f_s$.
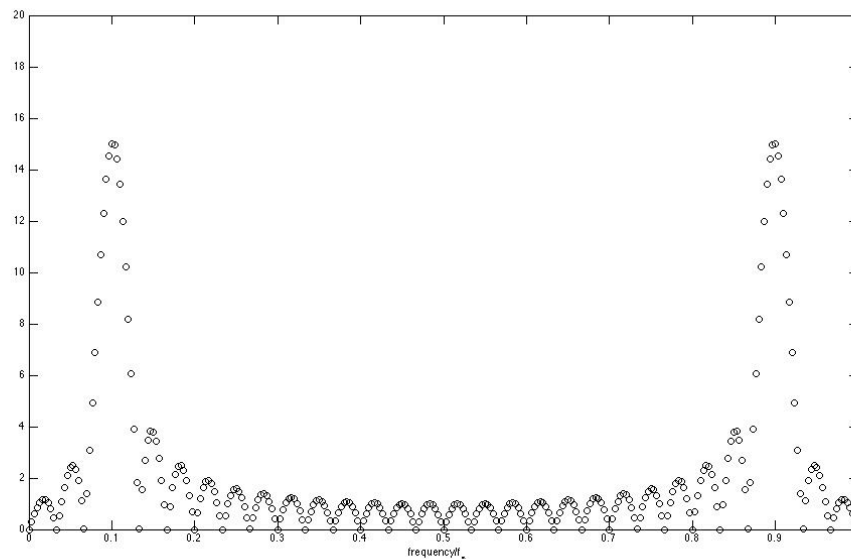

Figure 1: Plot showing the symmetry of a DFT

The code used to generate this figure was

```
n=[0:29];
x=cos(2*pi*n/10);
N=300;
X=abs(fft(x,N));
F=[0:N-1]/N;
plot(F,X,'ko');
axis([0 1 0 20]);
xlabel('frequency/f_s');
```

The figure window was then SAVED as a JPEG file.

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, we synthesize 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 concerned 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 1-1/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])
```
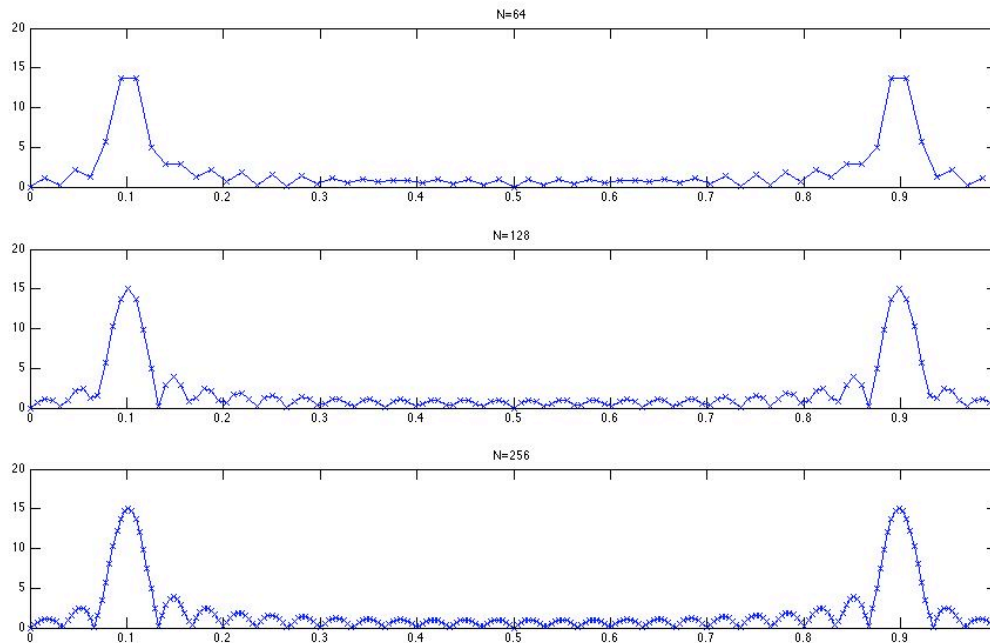

Figure 2: FFT of a cosine for N = 64, 128, and 256

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 extract that shape. The full code is:

```
n=[0:29];
x=cos(2*pi*n/10);
N1=64;N2=128;N3=256;
X1=abs(fft(x,N1));X2=abs(fft(x,N2));X3=abs(fft(x,N3));
F1=[0 : N1-1]/N1;F2=[0 : N2-1]/N2;F3=[0 : N3-1]/N3;
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])
```

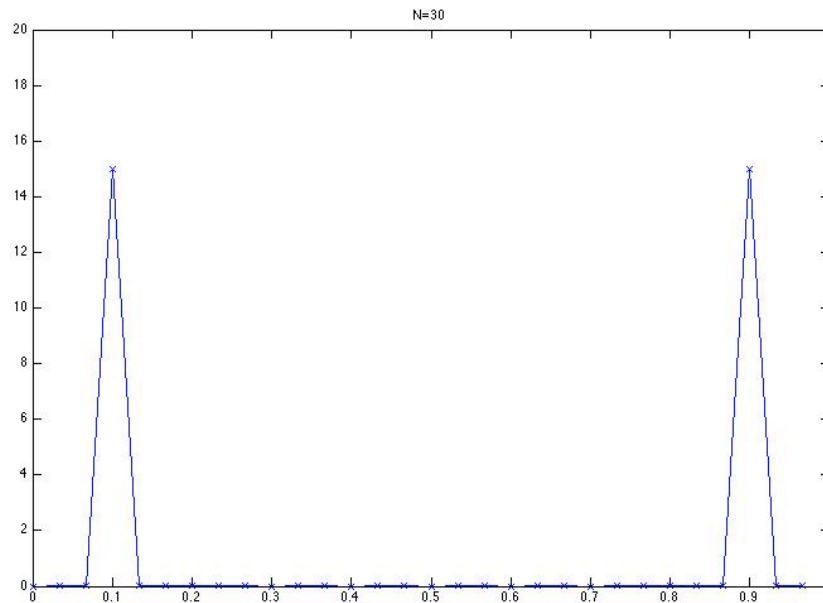If we choose N as the same number of samples in x[n], i.e., N = 30. We get the result in Figure 3.

Figure 3: N=30 = number of samples in x[n]

Why does the picture look like this?

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

```
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])
```
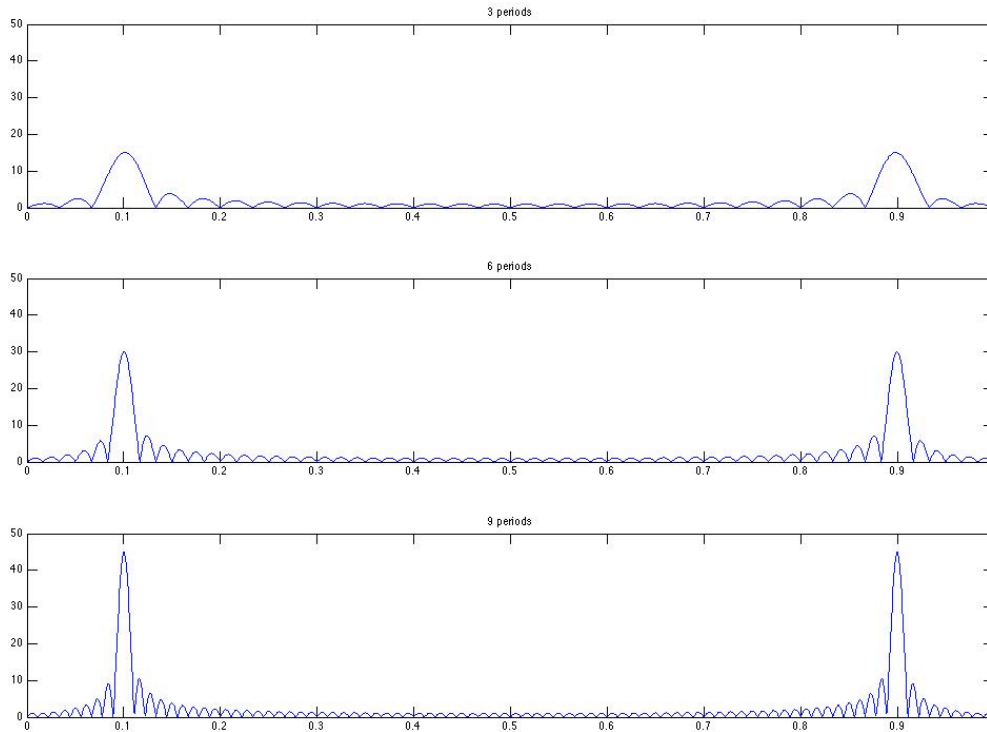
Figure 4: FFT of a cosine of 3, 6, and 9 periods

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 $0.9f_s$.

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 a 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 multiplying it with a rectangular box of length 30. A multiplication of a box and a sinusoid in the time domain results 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. Example 1 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 examples, 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 entirely below $f_s/2$, the Nyquist frequency.

Recall also that a real signal should have a transform magnitude that is symmetrical 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 $-f_s/2$ to $f_s/2$. This can be accomplished by using MATLAB's **fftshift** function as shown in the following code.

```
n=[0:149];
x1=cos(2*pi*n/10);
N=2048;
X=abs(fft(x1,N));
X=fftshift(X);
X3=abs(fft(x3,N));
F=[-N/2 : N/2-1]/N;
plot(F,X)
xlabel('frequency/f_s')
```
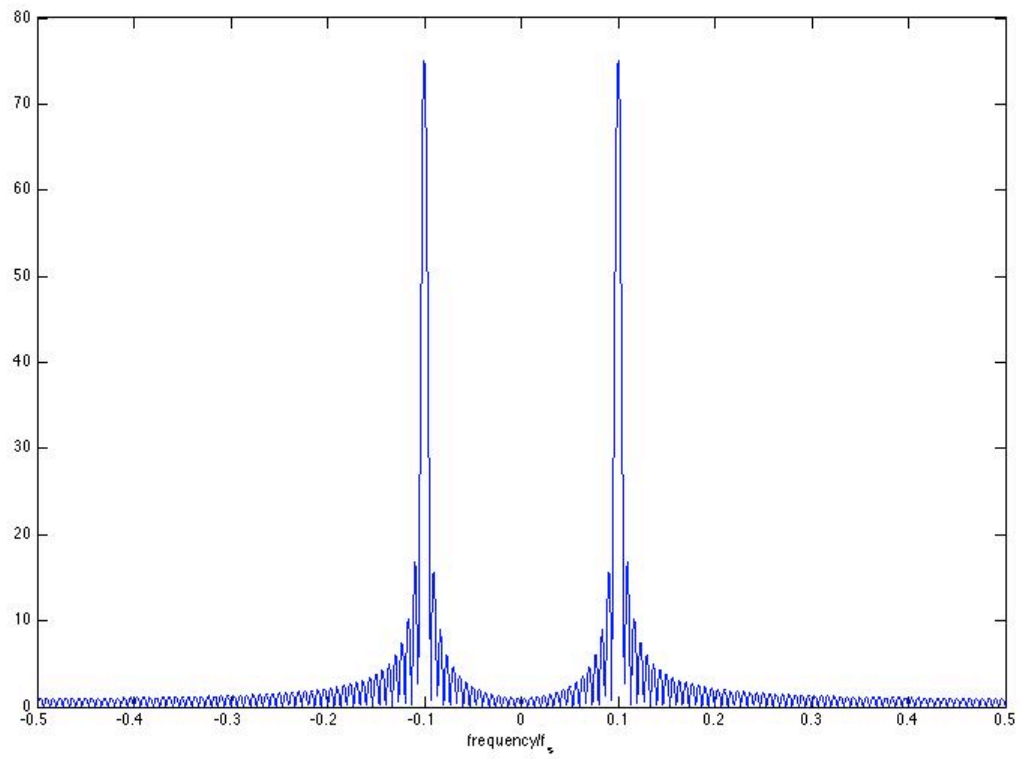
Figure 5: Approximate spectrum of a sinusoid with the FFT