Semester Project

Phase II

Stuart Miller
EE 3410
Missouri University of Science & Technology

The following is an exploration of the discrete-time sampling of continuous-time samples, utilizing MATLAB as a means to process the signals and generate a plotted output. To begin with, the assigned signals (Figure 1) consisted of a cosine function and a triangle impulse function.

$$\cos(2pif_o t)\,, where f_o = 356.5Hz \qquad tri\left(\frac{t}{a}\right), where\ a = 0.06$$

Figure 1

To begin with, let's look at a simple sampling of the cosine function at a frequency well above the Nyquist frequency. Here (Figure 2), we see what we want the function at a rather high sampling rate (20 samples per period). The discrete time range has been set to show 6 full periods and the frequency spacing is set to the ideal case of 0.001. This represents the ideal case in all aspects and will set the stage for our analysis. In the resulting plot We can clearly see the periodic nature of the cosine wave in both continuous and discrete time.

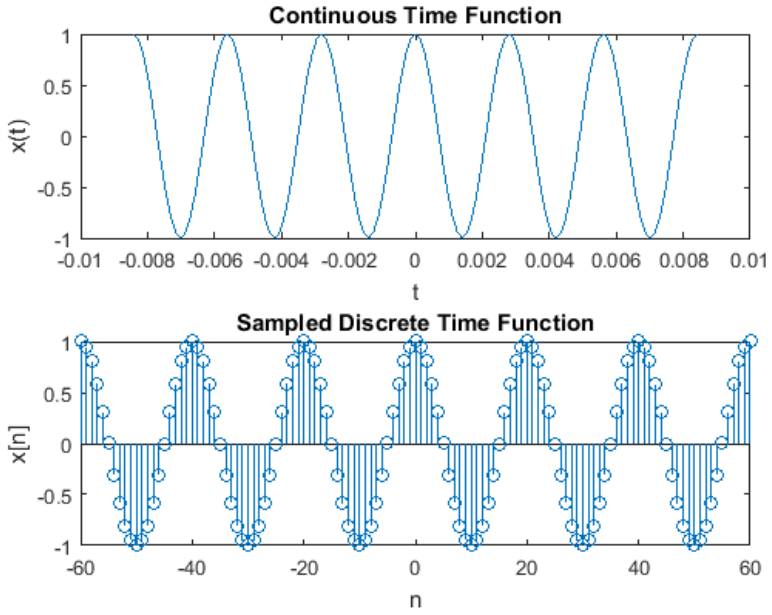

Figure 2

Now that we have an idea of what the signal should look like, we can begin a frequency analysis. The following code (Figure 3) was used to generate a discrete-time Fourier transform (hereafter referred to as "DTFT"). The algorithm implements the standard DTFT summation based on the sampled array of input data. The algorithm produces a resulting array in the frequency domain. To begin with, the algorithm will show one period of the DTFT, ranging from F=-0.5 to F=0.5. (This will be modified later). The two output arrays may be plotted against each other and the plot can be examined to find the original frequency at the peaks along the 'F' axis.

```
function [ x_dtft, f_dtft ] = DTFT( x_in, n_in, delta_f )

f_dtft = -0.5:delta_f:0.5;
x_dtft = zeros(size(f_dtft));
j = sqrt(-1);

for F = 1:1:length(f_dtft)
    sum = 0;
    for n = 1:1:length(n_in)-1
       sum = sum + x_in(n)*exp(-j*2*pi*n_in(n)*f_dtft(F));
    end
    x_dtft(F) = sum/length(n_in);
end

end
```

Figure 3

Running our sampled function from Figure 1 through the DTFT algorithm yields the following result for the DTFT (Figure 4). The parameters are as follows: Fs=20*Fo, frequency spacing= 0.001, and the n-array was generated from -60 to 60 (showing 6 periods in discrete time). This represents the numerical DTFT for ideal frequency spacing. Utilizing the datatip tool, we can clearly see that the DTFT produced our original frequency of 356.5Hz. It is also interesting to note that with this much detail, there is quite a bit of leakage. The plot on the left (Figure 4) shows the periodicity of the DTFT, while Figre 5 focuses in on just one period.

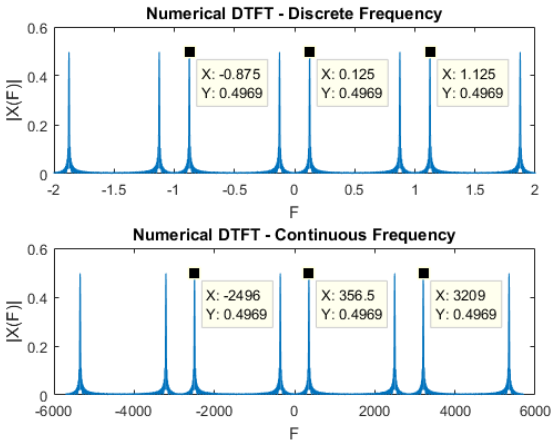

Figure 4

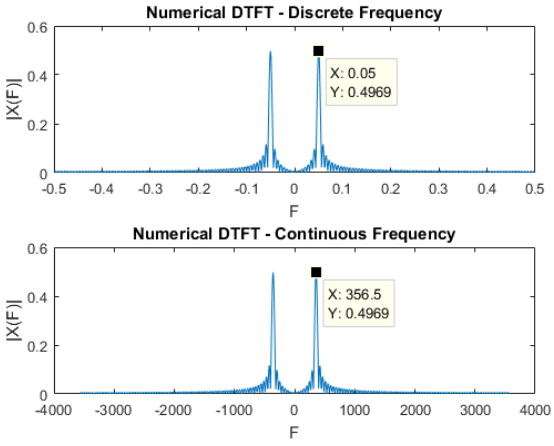

Figure 5

We can also repeat the analysis for a more realistic frequency spacing of 1/N. Decreasing the frequency spacing will, of course, reduce the accuracy of the resulting frequency. The accuracy is now entirely dependent on N (how many discrete samples we take). Here are Figure 4 and 5 repeated with a frequency spacing of 1/N. Because the frequency spacing is so much higher, we can no longer get as close to the target frequency of 365.5Hz.

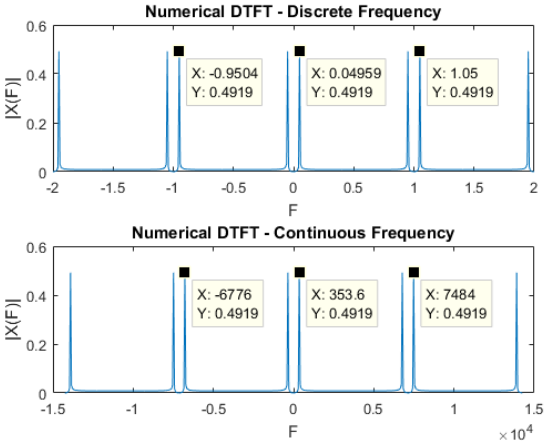

Figure 6

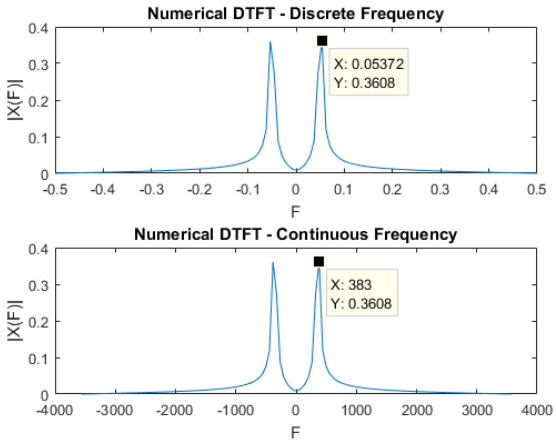

Figure 7

Of course, because our frequency spacing is now dependent on the number of samples taken (N), we can increase the frequency resolution by taking more samples and get a better result. In Figures 8 and 9, five times as many samples were taken (DT range -300 to 300). As you can see, we get much

close in both of these plots. It is also interesting to note that showing a greater frequency range produces more accurate results that simply one period of the frequency plot.

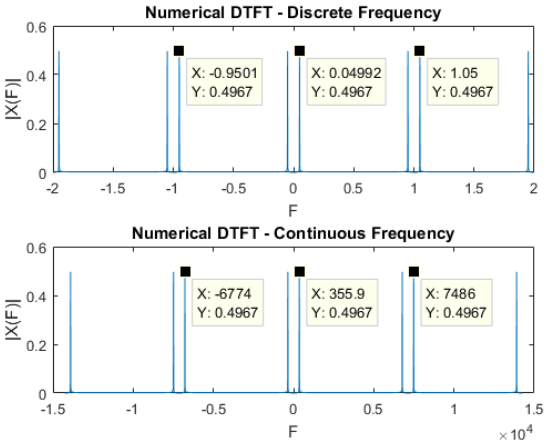

Figure 8

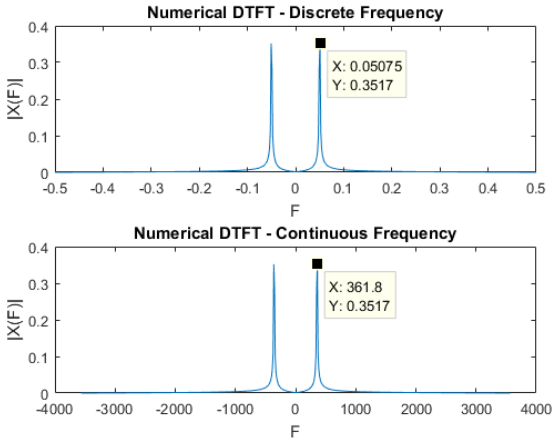

Figure 9

For the next step, we will compare our numerical plots to the theoretical DTFT to see if it matches the math. The definition for the theoretically DTFT of a cosine function is shown in Figure 6. The result of the theoretical CTFT is plotted in Figure 10. This is what all our numerical analyses should show! As we can see in the plot, we get a frequency of 356.5 Hz, exactly what we should see (and what matches our DTFT in Figures 4 and 5). Note that the impulse function given by the theoretical DTFT is much cleaner than the numerical DTFT and presents no leakage that we see in numerical analyses.

$$\cos(2\pi F_o n) \overset{F}{\leftrightarrow} \frac{1}{2}\big(comb(F - F_o) + comb(F + F_o)\big)$$

$$\cos(2\pi F_o n) \overset{F}{\leftrightarrow} \frac{1}{2}\big(\delta(F - F_o) + \delta(F + F_o)\big)$$

(for one period only)

Figure 6

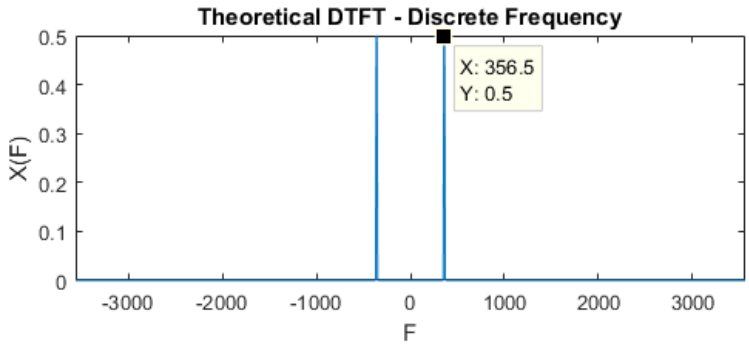

Figure 10

Another method we can use to obtain the DTFT is to get one period of the DTFT is to individually transform both the cosine function and a window function and convolve them.

$$rect\left[n,\frac{N}{2}\right] \overset{F}{\leftrightarrow} \left(2\left(\frac{N}{2}\right)+1\right) drcl\left(F,\left(2\left(\frac{N}{2}\right)+1\right)\right)$$

$$\cos(2\pi F_o n) \overset{F}{\leftrightarrow} \frac{1}{2}\left(comb(F-F_o)+comb(F+F_o)\right)$$

Convolution Yields…

$$\left(\frac{1}{2}\right)(N+\frac{1}{2})(drcl(F+F_o,(2N+1))+drcl(F-F_o,(2N+1))$$

Figure 11

Plotting the convolution yields the following result (Figure 11 with realistic frequency spacing and Figure 12 with ideal frequency spacing). Note how closely the frequency at the datatip matches that in the DTFT plots, for each frequency spacing.

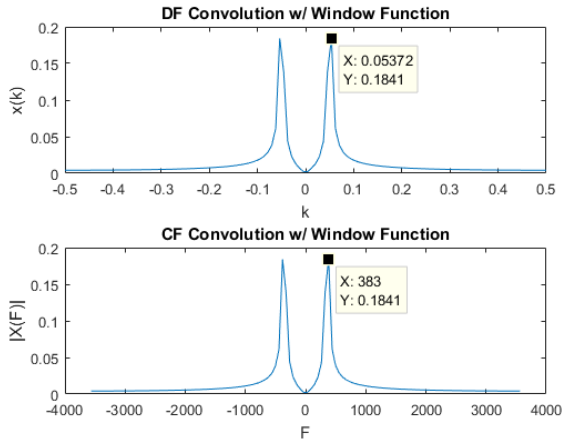

Figure 12

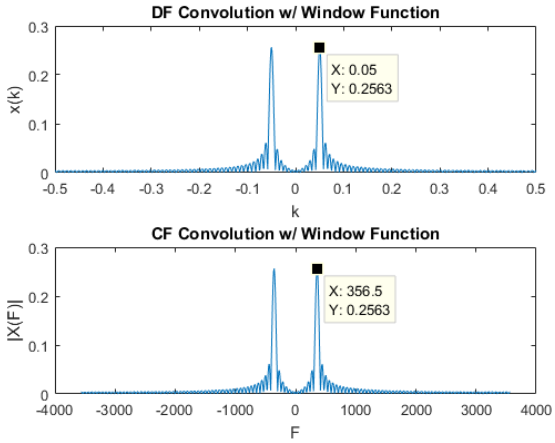

Figure 13

Next, we will take the Discrete Fourier Transform (DFT) of the signal (Figure 8). Here, we will be able to observe the full periodic nature of the DFT. This is because DFT is, by nature a periodic series. For this implementation of the algorithm, it was decided to show 6 full periods of the DFT. The code for the DFT algorithm can be seen in Figure 14.

```
function [ x_dft, n_dft ] = DFT( x_in, N )

n_dft = -3*N:1:3*N;
x_dft = zeros(size(n_dft));

for k=1:length(n_dft)
    sum = 0;
    for n=0:N-1
        sum = sum+(x_in(n+1)*exp(-sqrt(-1)*2*pi*n*n_dft(k)/N));
    end
    x_dft(k) = sum/N;
end

end
```

Figure 14

For plotting the DFT, we will, once again, sample at 20 times the source frequency, and keep a sample range of -60 to 60. Frequency spacing does not affect the DTF because it uses a discrete frequency series as part of its generation. Also, to show that the DFT is discrete, we will use a stem plot this time. As a result, placing the datatip on the first positive peak tells us that we got fairly close to our source frequency.

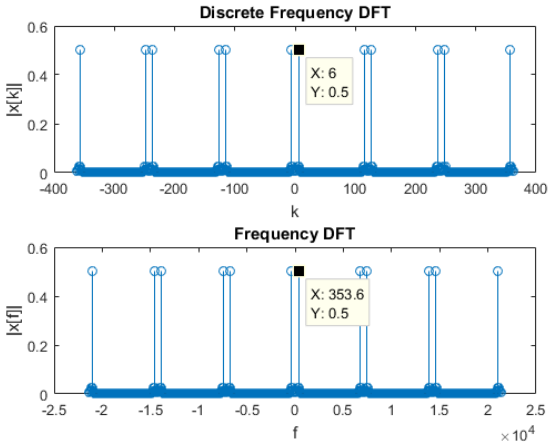

Figure 15

Finally, we will utilize MATLAB's fft() function to calculate the fast Fourier transform (FFT). Once more, the function was sampled at twenty times the source frequency over a discrete array from -60 to 60 (showing 12 waveforms). As we can see in the plotted output (Figure 10), we once again get the expected frequency of 356.5 Hz.

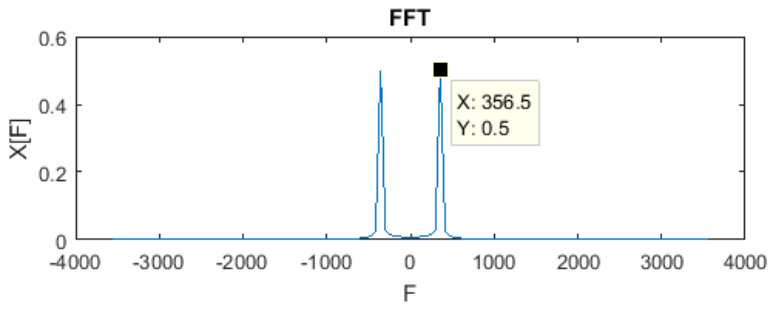

Figure 16

All of these outputs can be summed up in one cumulative comparison plot. Figure 17 shows our first comparison plot, generated using a DT array of -60 to 60, a sampling frequency of twenty time the source frequency, and ideal frequency spacing. Note how all the amplitudes match and the frequency peaks are in relatively the same location for each plot.

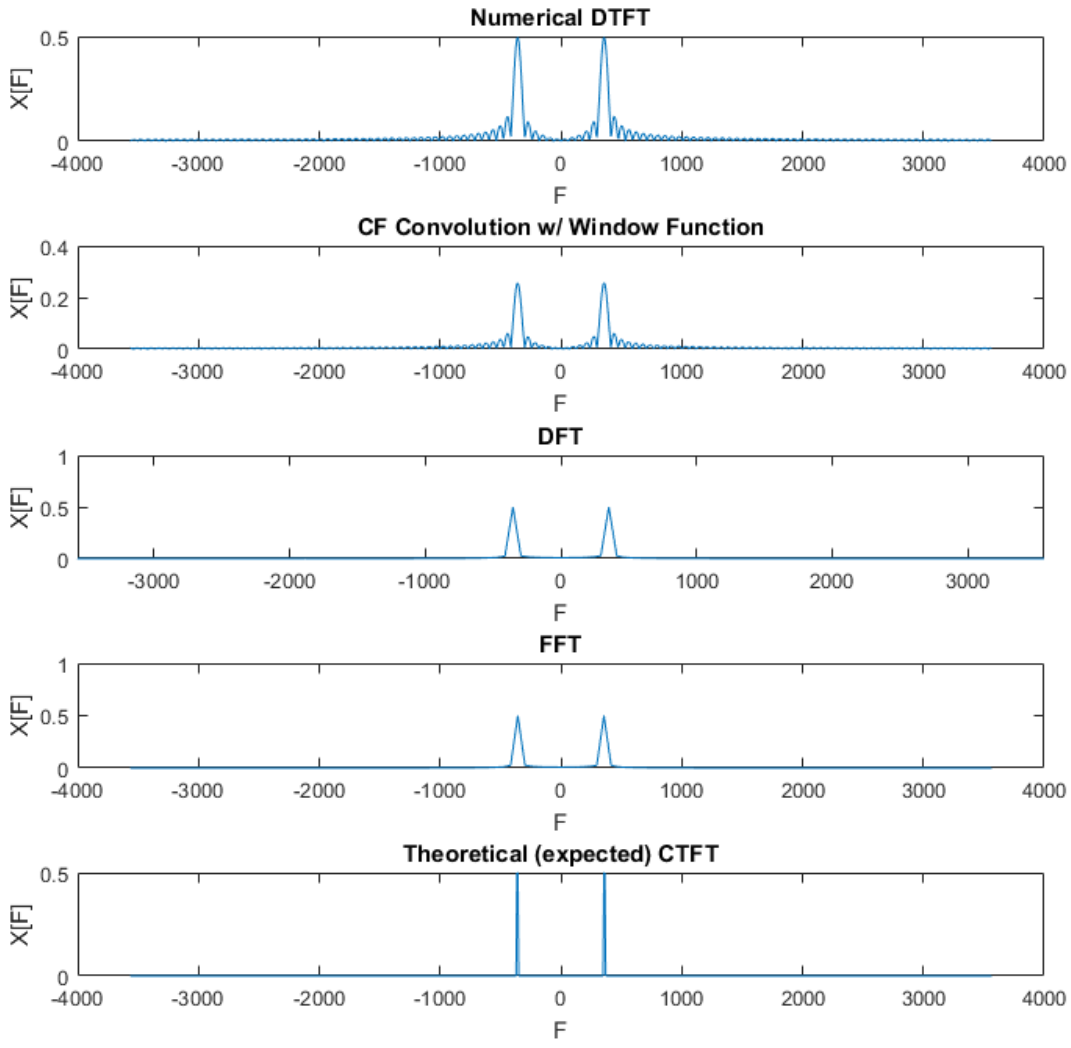

Figure 17

Now, we will move on to the triangle impulse function. The triangle function provides for some interesting analysis as it itself is not periodic and is time-limited. As with the cosine function, we will start with a simple CT vs. DT plot.

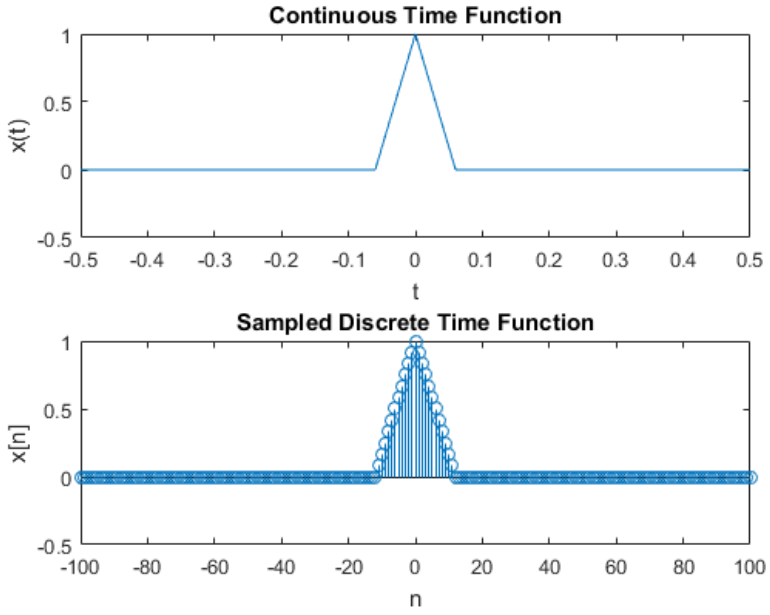

Figure 18

Moving along, we will again start with the DFT. Figure 19 shows the modified algorithm for the DFTF of an aperiodic signal. The amplitude-scaling factor has been removed.

```
function [ x_dtft, f_dtft ] = DTFT_tri( x_in, n_in, delta_f )

f_dtft = -0.5:delta_f:0.5;
x_dtft = zeros(size(f_dtft));
j = sqrt(-1);

for F = 1:1:length(f_dtft)
    sum = 0;
    for n = 1:1:length(n_in)
       sum = sum + x_in(n)*exp(-j*2*pi*n_in(n)*f_dtft(F));
    end
    x_dtft(F) = sum;
end

end
```

Figure 19

Figures 20 and 21 shows the plotted output of the DTFT. Figure 20 is the ideal frequency spacing, while Figure21 is the realistic frequency spacing. Both were taken with a DT range of -100 to 100 and a sampling frequency of 200Hz. As you can see, both have a frequency of about zero and are aperiodic. The amplitude is that of the dividing factor *a* of the function. Note how similar the plots are, despite the differing frequency spacing, and also that because of the realistic frequency spacing, we are not able to see a sample right at F=0.

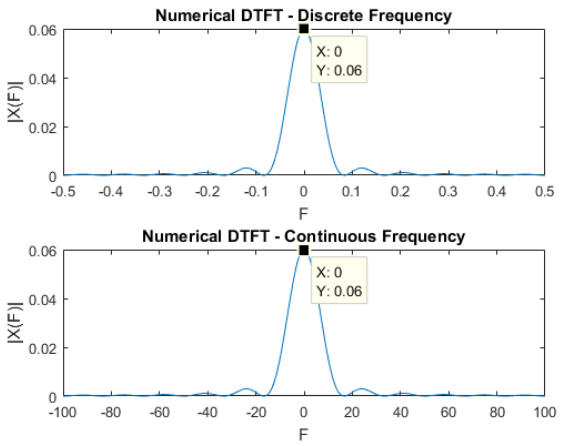

Figure 20

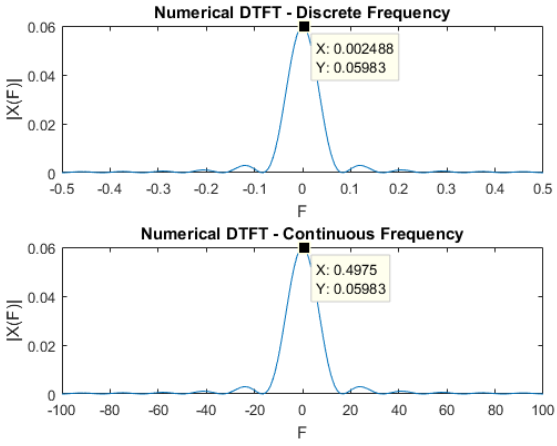

Figure 21

Next, we will examine the theoretical DTFT, calculated mathematically. The DTFT transform of a tri() function is shown in Figure 22. This was used to generate the continuous frequency version of the theoretical plot. The discrete version of the theoretical plot was generate using the linearity and stime-shifting properties of the DTFT, impletmented as a summing loop in MATLAB. This algorithm is shown in Figure 23.

$$tri\left(\frac{n}{a}\right) \overset{F}{\leftrightarrow} a * drcl^2(F, a)$$

Figure 22

```
x_dt_dtft = zeros(size(F));
j=sqrt(-1);
for i=1:1:N
    x_dt_dtft = x_dt_dtft + x(i)*exp((-j*2*pi*n(i)*F));
end
x_dt_dtft = x_dt_dtft / Fs;
```

Figure 23

The plotted output of the DTFT is shown below that (Ideal frequency spacing in Figure 24 and realistic in Figure 25, both with DT range -100 to 100 and Fs = 200Hz). As you can see, each matches the numerical DTFT closely.

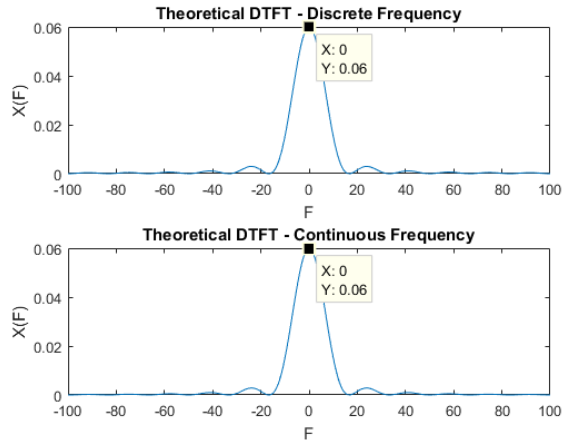

Figure 24

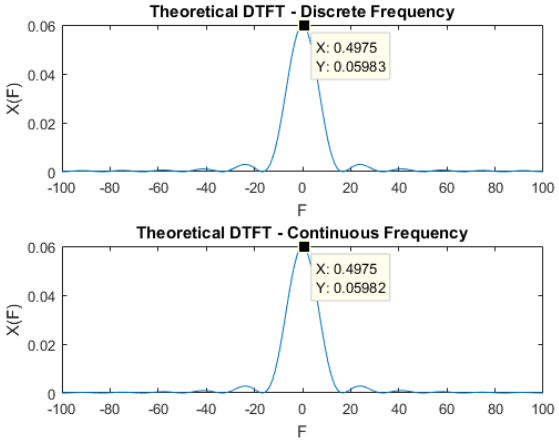

Figure 25

Finally, we will examine the DFT and FFT of the tri function. These are very similar to the process used for the cosine function, so I will only touch on them briefly. Figures 26 and 27 show the DFT output and Figures 28 and 29 show the FFT output. As before the plots on the left feature the ideal frequency spacing, while the plots on the right feature the realistic (1/N) frequency spacing. Note the periodic nature of the DFT.

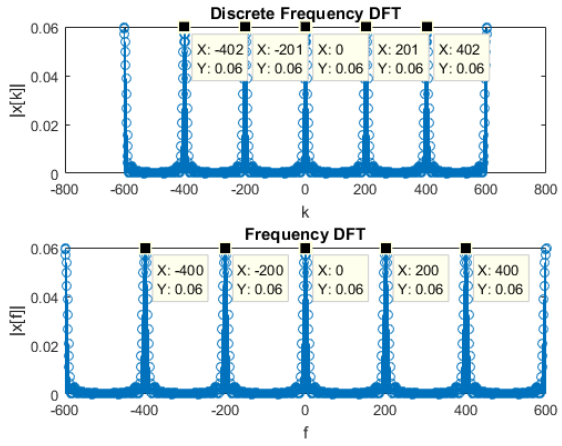

Figure 26

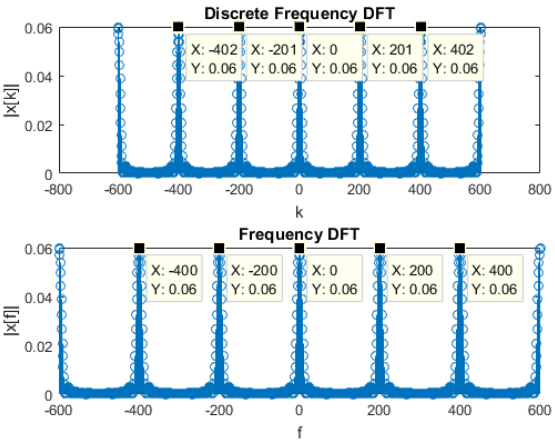

Figure 27

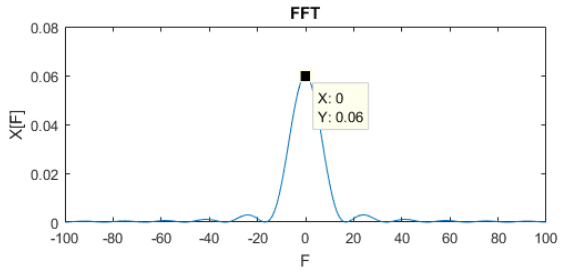

Figure 28

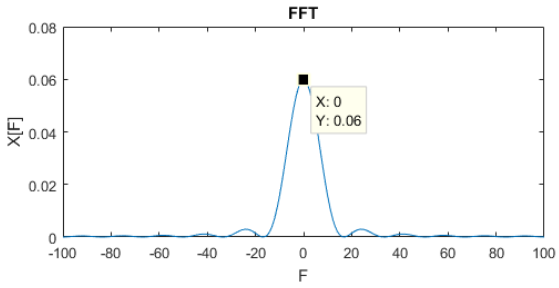

Figure 29

Finally, we have another comparison plot, this time for all the aggregated tri functions. Again, these were generated using a DT range of -100 to 100 and a sampling frequency of 200Hz. The ideal case frequency spacing is shown here for clarity. Note the consistent amplitude and shape of all of the plots.

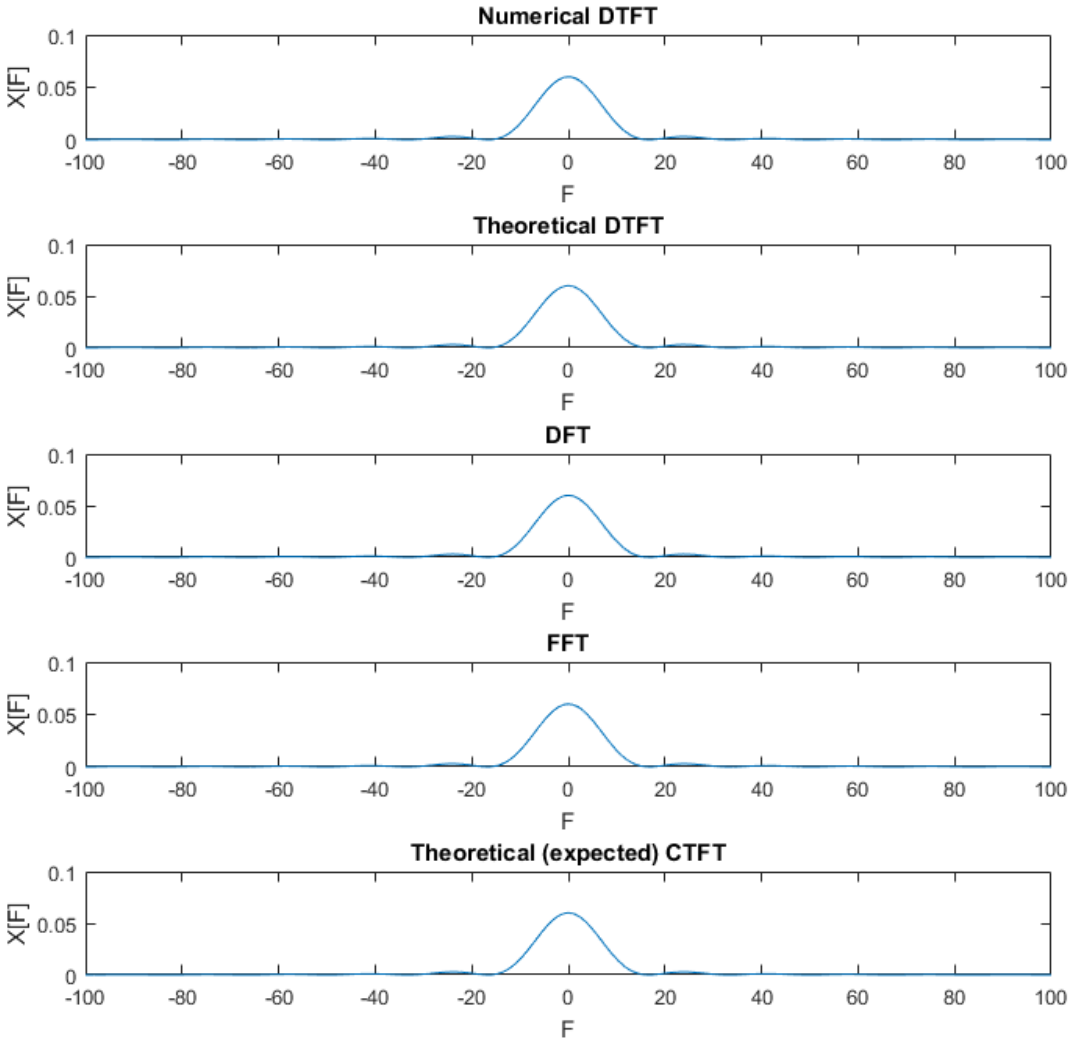

Figure 30

The remaining portion of this report will focus on varying the input parameters to produce more interesting plots. In practice, the sampling frequency may be limited by either the programmer' knowledge of the source signal, of the limitations of the sampling hardware. Furthermore, changing the sampling parameters can reveal interesting properties of the signal. In any case, it is worthwhile to perform an exploration of these signals from alternative perspectives. For this portion, we will be relying on the DFT transform. For each plot, the sampling parameters will be shown on the top line of the figure.

To begin, we will sample the cosine function at a sub-Nyquist rate of half the source frequency. This will produce a comb-like function containing only the peaks of the cosine function. It is still periodic, though so it will produce an interesting DFT. The transform for a comb function theoretically produces another comb function, and this is what we see in Figure 31. As you can see, the DFTF frequency matches the the frequency of the comb function we produced by sampling. See how the amplitude is

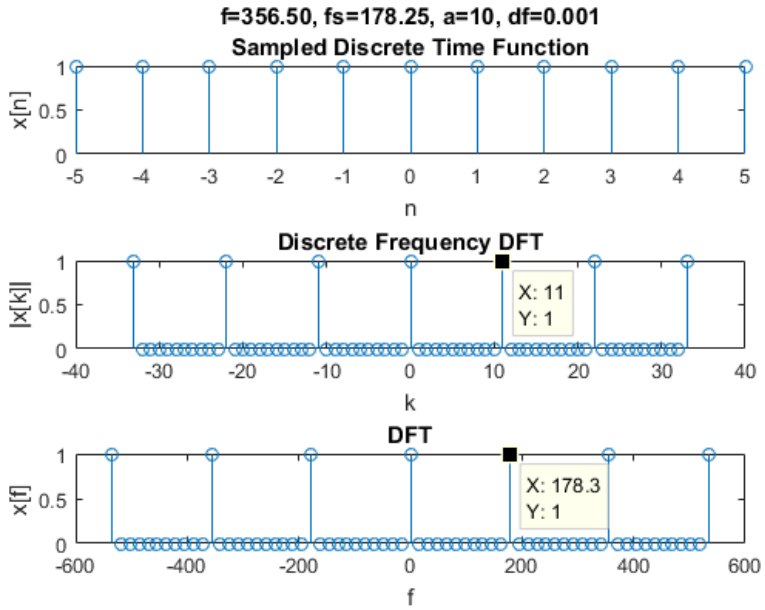

Figure 31

Similarly, we can get the same type of result by sampling at the same rate as the source frequency (half the minimum Nyquist rate). This will also produce a sampled camb function, although at twice the frequency of the previous plot. Figure 32 shows this same resulting comb function, but at twice the frequency.

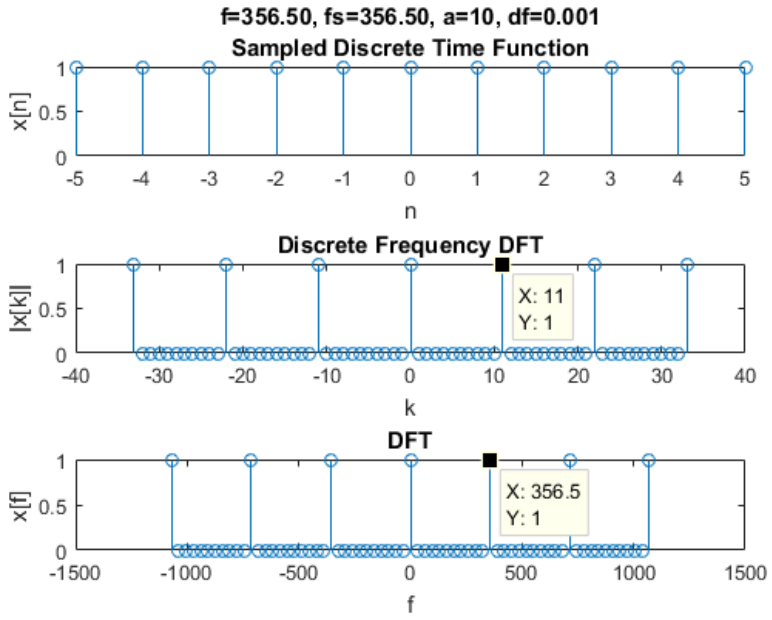

Figure 32

It is not until we sample at the Nyquist frequency that we begin to observe the sinusoidal nature of the function (Figure 33); note the peaks and troughs in the x[n]. Also see that the amplitude is now smaller. The resulting frequency is less accurate than before, yet it's still quite close to our source. While not ideal, this plot shows results that are not too far off from what we expect. This is evidence that the Nyquist rate is, indeed, the absolute lowest frequency that will produce usable results. Increasing the frequency from here will produce more and more detailed plots, approaching those shown earlier in Figure 15.

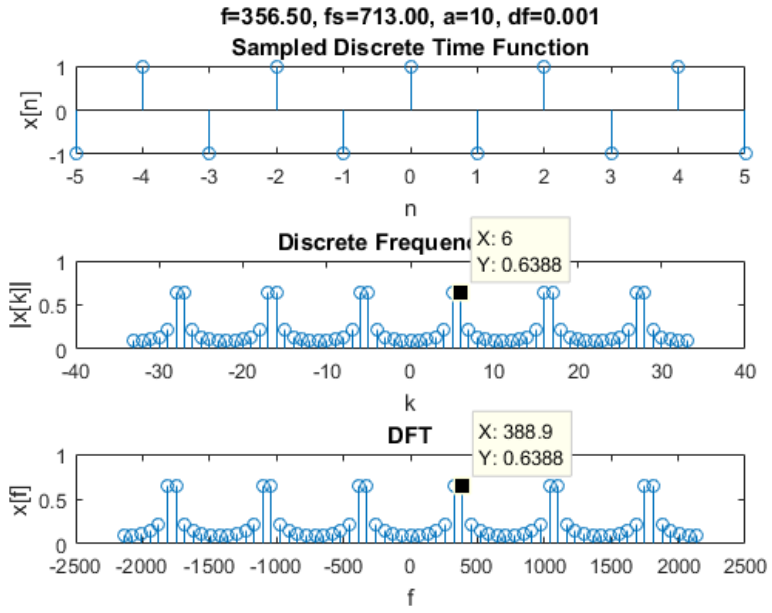

Figure 33

So far, we have taken for granted that we have an expectation of the source frequency and can therefore sample at a fraction or multiple of it. However, this is not likely to be the case in a realistic application. For the next examples, we will examine some frequencies that are chosen independent of the cosine's (presumably unknown) source frequency.

A logical choice of starting frequency for any signal is 500Hz. Seeing that 500Hz does not provide enough resolution, a programmer may decide to double that to 1kHz. Both examples are shown below. (500Hz in Figure 34, 1kHz in Figure 35) In order to make the under-sampling evident, I have overlaid the DT function with the continuous cosine function. When sampling at 500Hz, see how the under-sampling produces a DT function that looks like it peaks at about half the frequency. You can see how the DTF actually picks up on this as it also features a peak at half the true frequency. Sampling at 1kHz has the opposite effect. Its DT stems could have a cosine wave drawn between each one, plausibly allowing for a wave at twice the frequency.

Figures 36 and 37 take the secondary frequencies for the DFT results and plot them over the sampled stems in order to show where the frequencies originate from. As you can see, the stems align pretty well with the secondary frequencies. This shows the dangers of choosing too low of a sampling frequency!

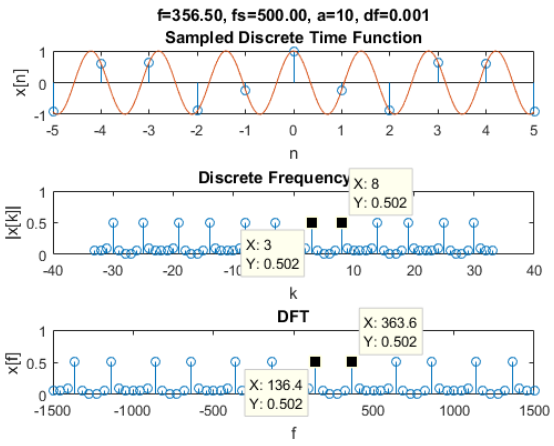

Figure 34

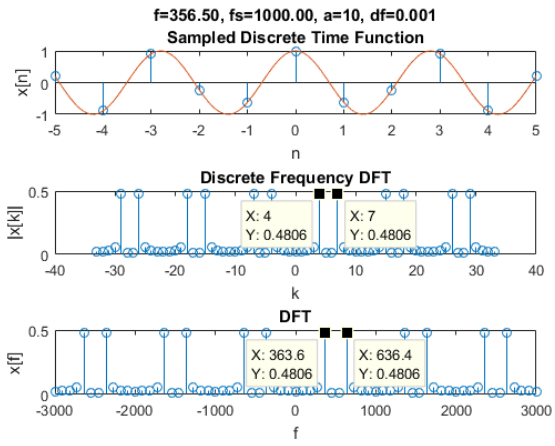

Figure 35

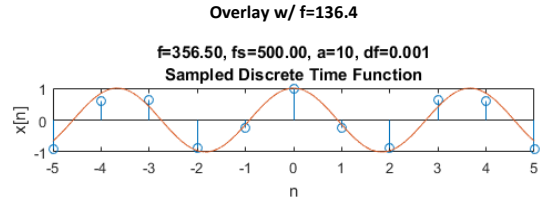

Figure 36

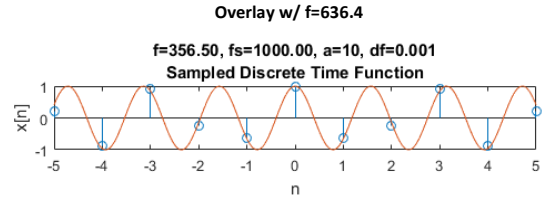

Figure 37

Finally, before we leave the cosine function, we'll take a look at the DTFT and FFT. In general, the FFT and DFT do not present well for anything less than the Nyquist frequency.

Figures 38 and 39 show the DTFT and FFT of the cosine sampled at half the Nyquist rate. Notice how they present as aperiodic. This does not change if we were to increase the DT range and is why they are unreliable at sub-Nyquist.

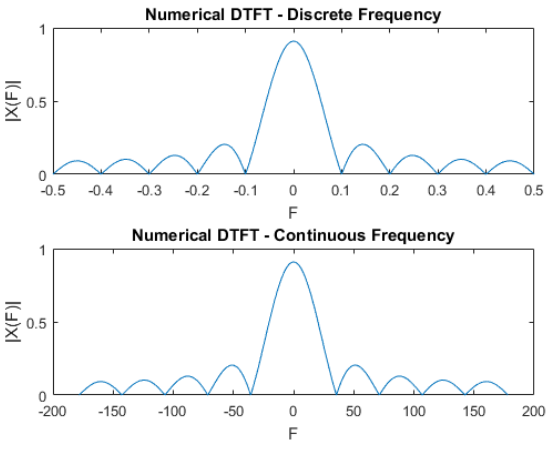

Figure 38

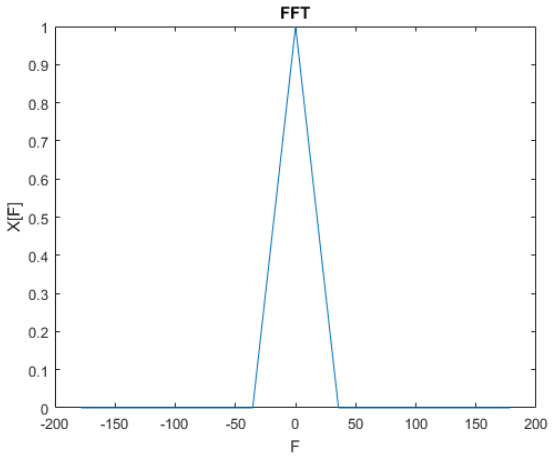

Figure 39

Figure 40 and 41 show the DFTF and FFT of the cosine sampled at the Nyquist rate. DTFT begins to be functional if the discrete frequency range is double to 1.0, whereas FFT still fails to see the periodicity of the function. Increasing the sampling rate from here produces plots closer to that of Figures 9 and 16.

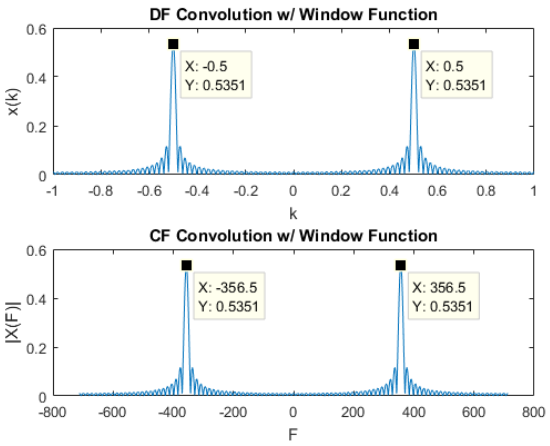

Figure 40

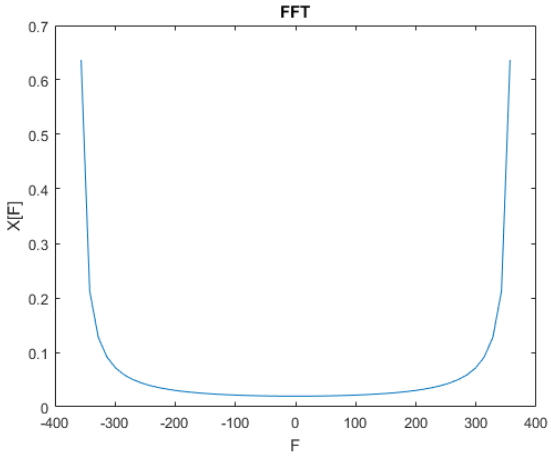

Figure 41

Next, we will focus in on alternative sampling parameters for the triangle impulse function. This function is not periodic like the cosine.For a starting point, I have chosen a sampling frequency that allows for only one nonzero sample. This is NOT satisfying the Nyquist criteria, as this only allows for one sample per waveform. The resulting DT waveform appears as a unit impulse (or delta) function. As we known from the definition of DTFT transform, a delta function should result in a DTFT of constant 1. Removing the amplitude scaling factor gives us exactly this, as shown in Figure 42.

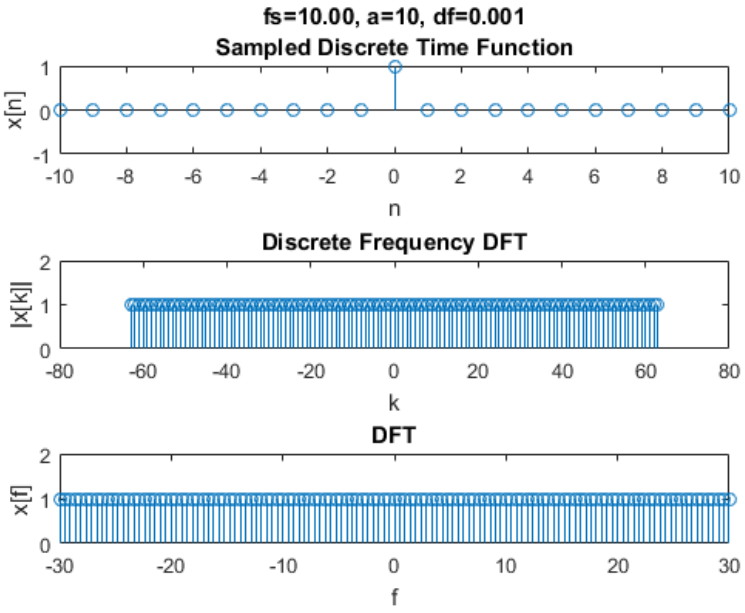

Figure 42

Next, we can increase the sampling frequency to allow for a vaguely triangular DT plot (only 3 nonzero stems). This represents satisfying the Nyquist criteria, as we have more than one sample per waveform. The theoretical DTFT of a triangle function is a dirichlet (as we known from the definition in Figure 22). Since this triangle function has such a small width, the second term of the dirichlet will correspondingly be very small. This contributes to very few fluctuations between each peak of the dirichelt. This is seen in the DFT plot as the dirichlet appears to be purely sinusoidal, with no fluctuations. Also see how the amplitude is now corrected to the expected value of source a = 0.06.

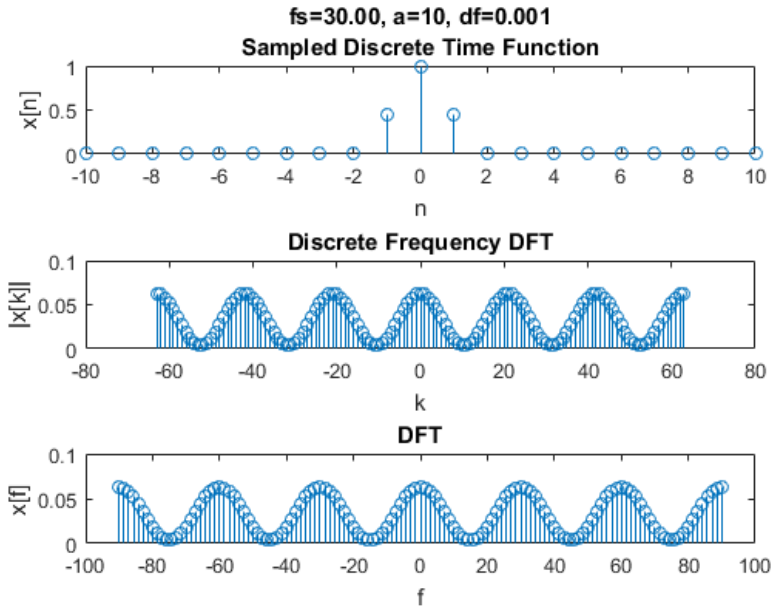

Figure 43

Increasing the sampling frequency once more to a little bit above the Nyquist frequency produces a DTFT that looks much more like a dirichelt. While still somewhat under-sampled, this gets us much closer to our expected result. Not the fluctuations between peaks. This is characteristic of a dirichelt, even though squaring it diminishes them significantly.

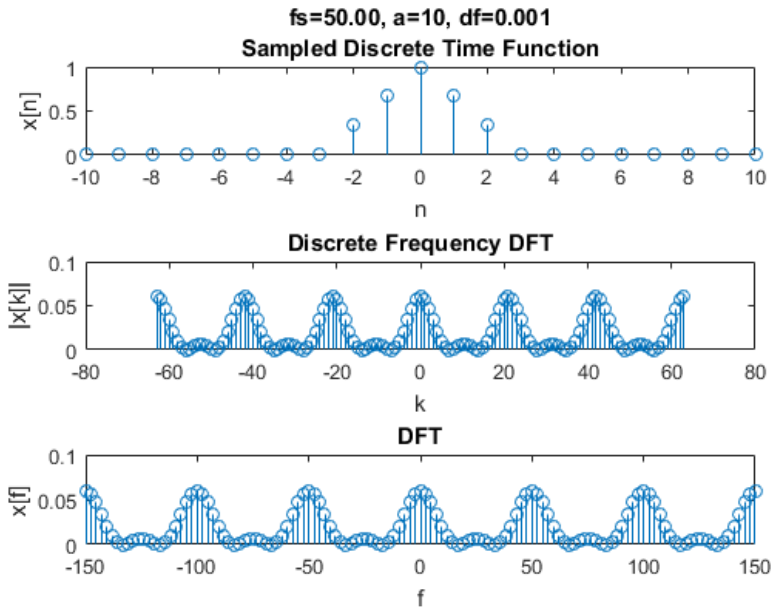

Figure 44

Another interesting case would be to choose a sampling frequency that is too high (above Nyquist, but over-sampled). Say we were to approach this the same way as the cosine function earlier

and assume a sampling frequency of 1kHz. At this frequency the triangle function would appear to be almost a constant. The theoretical DTFT of a constant is a comb function. As expected, this is what we see in Figure 45.

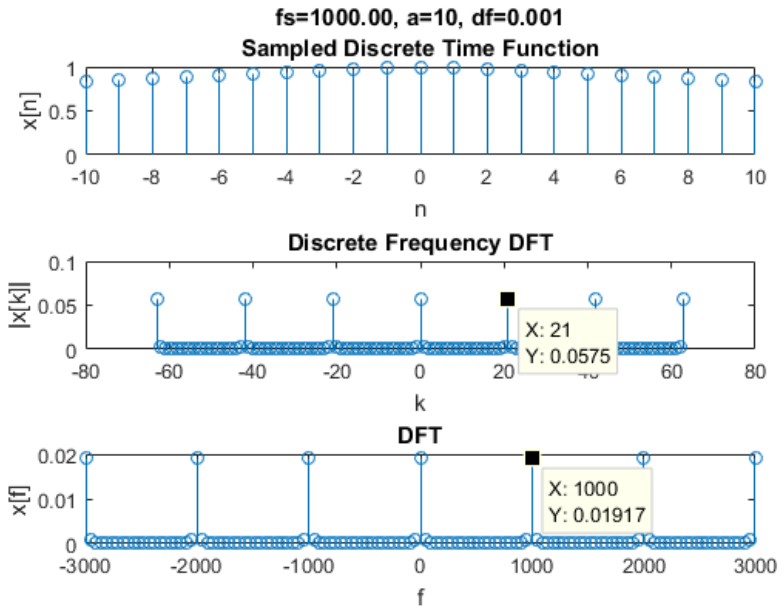

Figure 45

Remember how undersampling the cosine function produced exactly the same result; a DTFT of a comb function. These two completely different function produced almost exactly the same DTFT when sampled incorrectly. Once again, this shows why a programmer must be very careful in what sampling frequency they choose! Choosing incorrectly can be very misleading, as we have seen here.

Finally, we can examine the DFT and FFT of an under/oversampled triangle function. It is a little less exciting as its fairly accurate as soon as the triangle shape emerges. Figures 46 and 47 show the DTFT and FFT when the tri() is sampled to look like a delta. Again, observe the constant result.

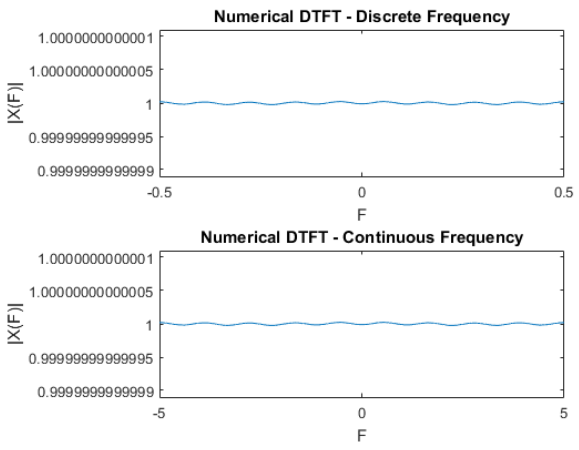

Figure 46

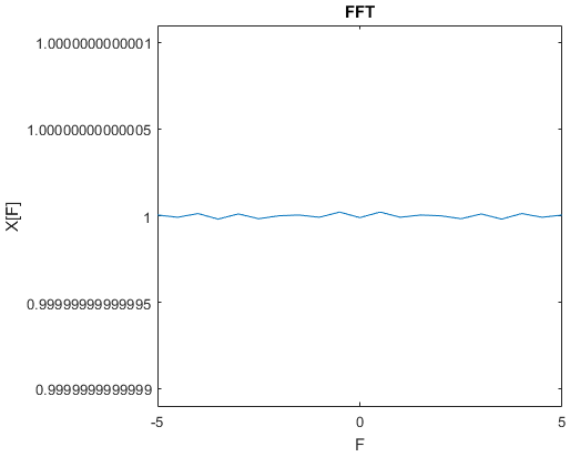

Figure 47

Figures 48 and 49 show the DFTF and FFT at the oversampled rate of 1kHz. While the DFT failed here and produced a comb instead, the DTFT and FFT work differently and produce a pulse at zero as a result (properly showing the result of an aperiodic function)

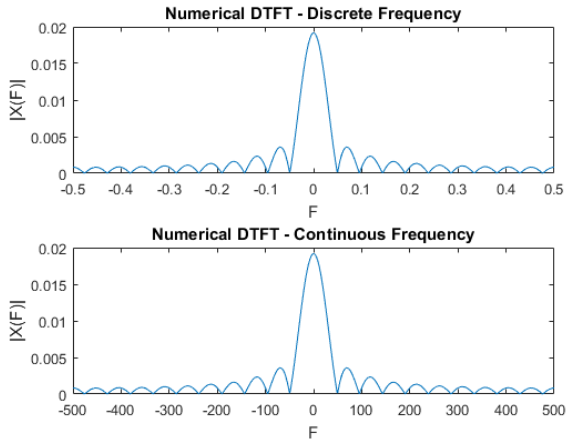

Figure 48

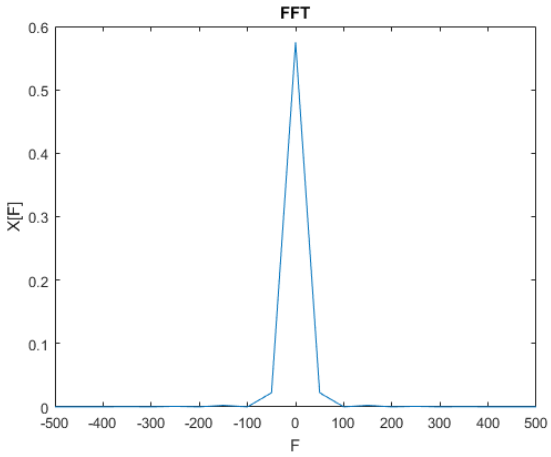

Figure 49

To conclude this analysis of sampling and Fourier transforms, it is important to remember that the sampling parameters can drastically change the way a function is processed. Choosing a sampling frequency that is too far off can completely change the signal and produce that is completely different from what is to be expected.

As we saw in the creative analysis portion for each function, sampling at below the Nyquist rate can have disastrous results and can completely mangle your signal. A programmer must be sure that

they sampling frequency is not below or too far above the Nyquist rate either, or they may receive erroneous results.

Throughout all the observations, we saw that sampling at the Nyquist is often sufficient to produce a roughly accurate frequency. Sampling at anywhere above that is perfectly feasible and often produces exact results. I based my starting analysis off of Fs=20*F. This is more than sufficient, and perhaps a little bit too high. Any rate in between these two should be sufficient, depending on the arracy required for the specific application. One would have to consider the hardware constraints of their embedded system and compare the desired accuracy to this when choosing a sampling frequency.

Furthermore, special consideration is required for aperiodic signals such as the triangle function. As we saw above, amplitude scaling is not required for an aperiodic function and this is a modification that must be accounted for should a programmer be expecting an aperiodic function.

Undoubtable, signal processing requires a large amount of attention on the part of the programmer and is a field that demand both precision and creativity.

Appendix – Complete MATLAB source code

run_cos.m

```
clc; clear; close all;

% Given fo from project assignment
Fo = 356.5;
To = 1/Fo;
% Choose an Fs
Fs = Fo;          % WILL ALTER THIS FIELD FOR EACH RUN
Ts = 1/Fs;
% Choose a DT sample range
a = 50;
n = -a/2:1:a/2;       % WILL ALTER THIS FIELD FOR EACH RUN
N = length(n);
% Choose a delta f (freq. spacing)
df = 0.001;          % WILL ALTER THIS FIELD FOR EACH RUN
% Source Function
x = cos(2*pi*n*Fo/Fs);

% Sample in CT and DT
figure();
subplot(2,1,1)
plot(n*Ts,x);
title('Continuous Time Function');
xlabel('t');
ylabel('x(t)');
subplot(2,1,2);
stem(n,x);
title('Sampled Discrete Time Function');
xlabel('n');
ylabel('x[n]');

% Numerical DTFT
[x_dtft,F] = DTFT(x,n,df);
figure();
subplot(2,1,1);
plot(F,abs(x_dtft));
title('Numerical DTFT - Discrete Frequency');
xlabel('F');
ylabel('|X(F)|');
subplot(2,1,2);
plot(F*Fs,abs(x_dtft));
title('Numerical DTFT - Continuous Frequency');
xlabel('F');
ylabel('|X(F)|');

% Theoretical DTFT
x_t_dtft = 0.5*(unitImpulse(F-Fo/Fs)+unitImpulse(F+Fo/Fs));
figure();
plot(F*Fs,x_t_dtft);
xlim([-Fs/2 Fs/2]);
title('Theoretical DTFT - Discrete Frequency');
xlabel('F');
ylabel('X(F)');

% Conv w/ window funct
Nw = floor(N/2);
x_win_dtft_conv = 0.5*(Nw/Fs+0.5)*(drcl(F+Fo/Fs,(2*Nw+1))+drcl(F-Fo/Fs,(2*Nw+1)));
figure();
subplot(2,1,1);
plot(F,abs(x_win_dtft_conv));
title('DF Convolution w/ Window Function');
xlabel('k');
ylabel('x(k)');
subplot(2,1,2);
plot(F*Fs,abs(x_win_dtft_conv));
title('CF Convolution w/ Window Function');
xlabel('F');
ylabel('|X(F)|');
```

```
% DFT
[x_dft,n_dft] = DFT(x,N);
f = n_dft.*Fs/N;
figure();
subplot(2,1,1);
stem(n_dft,abs(x_dft));
title('Discrete Frequency DFT');
xlabel('k');
ylabel('|x[k]|');
subplot(2,1,2);
stem(f,abs(x_dft));
title('Frequency DFT');
xlabel('f');
ylabel('|x[f]|');

% FFT
y = fft(x,N)/(length(x));
x_fft = fftshift(y);
f_fft = linspace(-Fs/2,Fs/2,length(x));
figure();
plot(f_fft,abs(x_fft));
title('FFT');
xlabel('F');
ylabel('X[F]');

% Comparison
figure();
subplot(5,1,1);
plot(F*Fs,abs(x_dtft));
title('Numerical DTFT');
xlabel('F');
ylabel('X[F]');
subplot(5,1,2);
plot(F*Fs,abs(x_win_dtft_conv));
title('CF Convolution w/ Window Function');
xlabel('F');
ylabel('X[F]');
subplot(5,1,3);
plot(f,abs(x_dft));
title('DFT')
xlabel('F');
ylabel('X[F]');
xlim([-Fs/2 Fs/2]);
subplot(5,1,4);
plot(f_fft,abs(x_fft));
title('FFT')
xlabel('F');
ylabel('X[F]');
subplot(5,1,5);
plot(F*Fs,x_t_dtft);
title('Theoretical (expected) CTFT')
xlabel('F');
ylabel('X[F]');

% Creative portion plots
figure();
subplot(3,1,1);
stem(n,x); hold on
n_full = min(n):0.001:max(n);
x_full = cos(2*pi*n_full*Fo/Fs);
plot(n_full,x_full);
title(sprintf('f=%.2f, fs=%.2f, a=%u, df=%.3f\n%s',Fo,Fs,a,df,'Sampled Discrete Time
Function'));
xlabel('n');
ylabel('x[n]');
subplot(3,1,2);
stem(n_dft,abs(x_dft));
title('Discrete Frequency DFT');
xlabel('k');
ylabel('|x[k]|');
subplot(3,1,3);
stem(f,abs(x_dft));
title('DFT')
xlabel('f');
ylabel('x[f]');
```

run_tri.m

```
clc; clear; close all;

% Given a from project assignment
a = 0.06;
% Choose a sampling frequency
Fs = 10;                % WILL ALTER THIS FIELD FOR EACH RUN
Ts = 1/Fs;
% Choose a DT sample range
n = -10:1:10;           % WILL ALTER THIS FIELD FOR EACH RUN
N = length(n);
% Choose a delta f (freq. spacing)
%ideal = 0.001, actual = 1/N
df = 0.001;             % WILL ALTER THIS FIELD FOR EACH RUN
% Source Function
x = tri(n*Ts/a);

% Sample in CT and DT
figure();
subplot(2,1,1)
plot(n*Ts,x);
title('Continuous Time Function');
xlabel('t');
ylabel('x(t)');
subplot(2,1,2);
stem(n,x);
title('Sampled Discrete Time Function');
xlabel('n');
ylabel('x[n]');

% Numerical DTFT
[x_dtft,F] = DTFT_tri(x,n,df);
x_dtft = x_dtft / Fs; % amplitude scale
figure();
subplot(2,1,1);
plot(F,abs(x_dtft));
title('Numerical DTFT - Discrete Frequency');
xlabel('F');
ylabel('|X(F)|');
subplot(2,1,2);
plot(F*Fs,abs(x_dtft));
title('Numerical DTFT - Continuous Frequency');
xlabel('F');
ylabel('|X(F)|');

% Theoretical  DTFT (using linearity and time-shifting)
x_dt_dtft = zeros(size(F));
j=sqrt(-1);
for i=1:1:N
    x_dt_dtft = x_dt_dtft + x(i)*exp((-j*2*pi*n(i)*F));
end
x_dt_dtft = x_dt_dtft / Fs;

% Theoretical CTFT (using equation from project sheet)
x_ct_dtft = a*(sinc(a*F*Fs)).^2;

figure();
subplot(2,1,1);
plot(F*Fs,abs(x_dt_dtft));
title('Theoretical DTFT - Discrete Frequency');
xlabel('F');
ylabel('X(F)');
subplot(2,1,2);
plot(F*Fs,abs(x_ct_dtft));
title('Theoretical DTFT - Continuous Frequency');
xlabel('F');
ylabel('X(F)');
```

DTFT.m

```
function [ x_dtft, f_dtft ] = DTFT( x_in, n_in, delta_f )

f_dtft = -0.5:delta_f:0.5;
x_dtft = zeros(size(f_dtft));
j = sqrt(-1);

for F = 1:1:length(f_dtft)
    sum = 0;
    for n = 1:1:length(n_in)-1
       sum = sum + x_in(n)*exp(-j*2*pi*n_in(n)*f_dtft(F));
    end
    x_dtft(F) = sum/length(n_in);
end

end
```

DTFT_tri.m

```
function [ x_dtft, f_dtft ] = DTFT_tri( x_in, n_in, delta_f )

f_dtft = -0.5:delta_f:0.5;
x_dtft = zeros(size(f_dtft));
j = sqrt(-1);

for F = 1:1:length(f_dtft)
    sum = 0;
    for n = 1:1:length(n_in)
       sum = sum + x_in(n)*exp(-j*2*pi*n_in(n)*f_dtft(F));
    end
    x_dtft(F) = sum;
end

end
```

DFT.m

```
function [ x_dft, n_dft ] = DFT( x_in, N )

n_dft = -3*N:1:3*N;
x_dft = zeros(size(n_dft));

for k=1:length(n_dft)
    sum = 0;
    for n=0:N-1
        sum = sum+(x_in(n+1)*exp(-sqrt(-1)*2*pi*n*n_dft(k)/N));
    end
    x_dft(k) = sum/N;
end

end
```