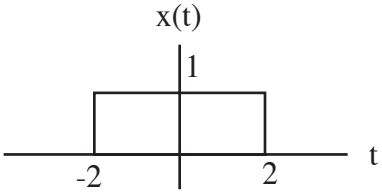
ECE 460 – Introduction to Communication Systems
MATLAB Tutorial #2

Evaluating Fourier Transforms with MATLAB

In class we study the analytic approach for determining the Fourier transform of a continuous time signal. In this tutorial numerical methods are used for finding the Fourier transform of continuous time signals with MATLAB are presented.

Using MATLAB to Plot the Fourier Transform of a Time Function

The aperiodic pulse shown below:



has a Fourier transform:

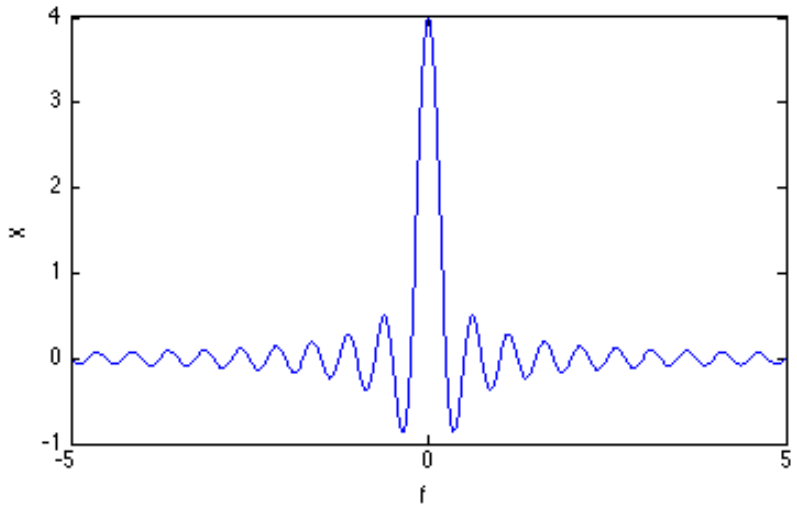
$$X(jf) = 4\,\mathrm{sinc}(4\pi f)$$

This can be found using the Table of Fourier Transforms. We can use MATLAB to plot this transform. MATLAB has a built-in *sinc* function. However, the definition of the MATLAB *sinc* function is slightly different than the one used in class and on the Fourier transform table. In MATLAB:

$$\mathrm{sinc}(x) = \frac{\sin(\pi x)}{\pi x}$$

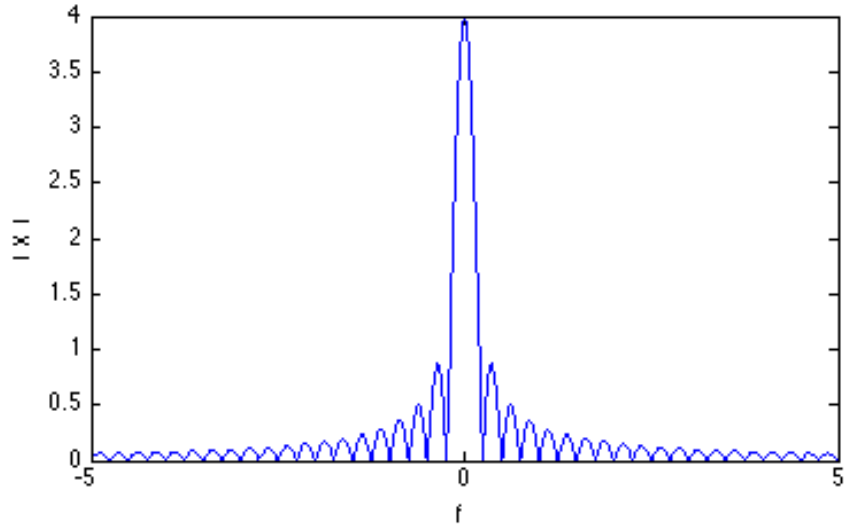
Thus, in MATLAB we write the transform, X, using sinc(4f), since the π factor is built in to the function. The following MATLAB commands will plot this Fourier Transform:

```
>> f=-5:.01:5;
>> X=4*sinc(4*f);
>> plot(f,X)
```

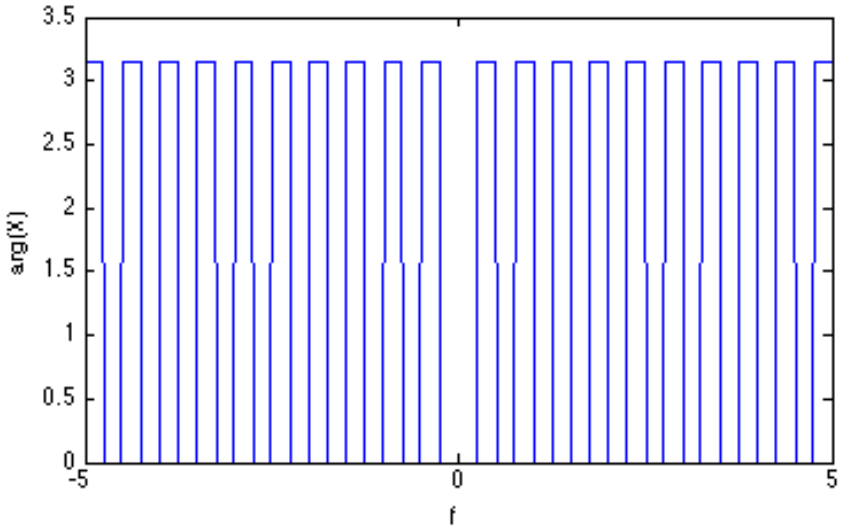


In this case, the Fourier transform is a purely real function. Thus, we can plot it as shown above. In general, Fourier transforms are complex functions and we need to plot the amplitude and phase spectrum separately. This can be done using the following commands:

```
>> plot(f,abs(X))
```




```
>> plot(f,angle(X))
```



Note that the angle is either zero or π. This reflects the positive and negative values of the transform function.

Performing the Fourier Integral Numerically

For the pulse presented above, the Fourier transform can be found easily using the table. However, for some functions, an integration will need to be performed to find the transform using:

$$X(jf) = \int_{-\infty}^{\infty} x(t)e^{-j2\pi ft}dt$$

or, for this example:

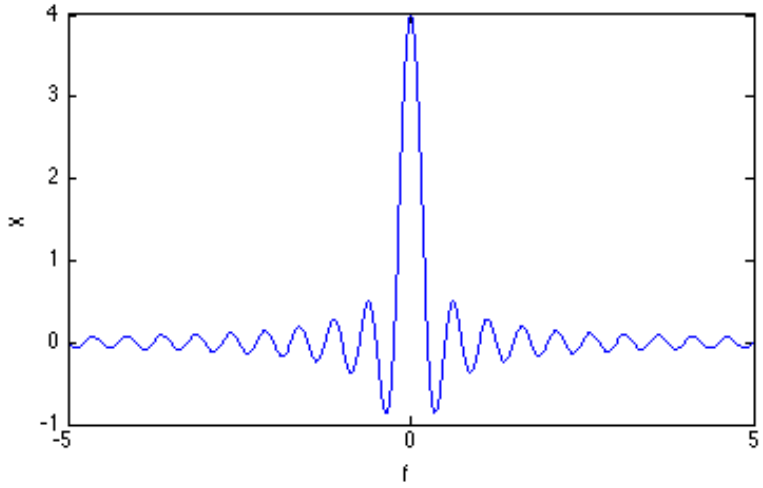
$$X(jf) = \int_{-2}^{2} 1e^{-j2\pi ft}dt$$

This integration can be performed using the *trapz* command in MATLAB. This command has the form: trapz(x,y) where the integral of the function y is found with respect to the variable of integration, x. Or, using MATLAB:

```
>> clear
>> f=0;
>> t=-2:.01:2;
>> trapz(t,exp(-j*2*pi*f*t))
ans =
    4.0000
```

This is consistent with our earlier results. The value of the transform at f = 0 was found to be 4 using the transform from the table. In order to find the complete transform over a range of frequencies we can use a for loop as shown below.

```
>> clear
>> t=-2:.01:2;
>> k=0;
>> for f=-5:.01:5
   k=k+1;
   X(k)=trapz(t,exp(-j*2*pi*f*t));
   end
>> f=-5:.01:5;
>> plot(f,X)
```



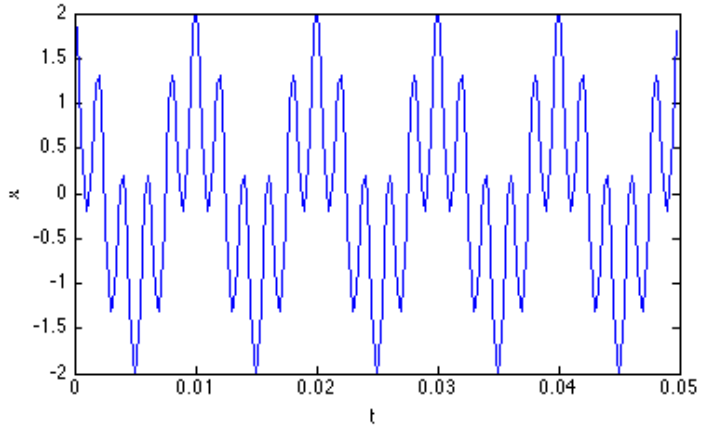
This should match the transform plotted above. This technique can also be used to approximate the Fourier transform of an infinite duration signal. Say we want to find the amplitude spectrum of the two-frequency signal:

$$x(t) = \cos(2\pi 100t) + \cos(2\pi 500t)$$

We begin by creating a vector, x, with sampled values of the continuous time function. If we want to sample the signal every 0.0002 seconds and create a sequence of length 250, this will cover a time interval of length 250*0.0002 = 0.05 seconds. A plot of this signal is generated using the following MATLAB code:

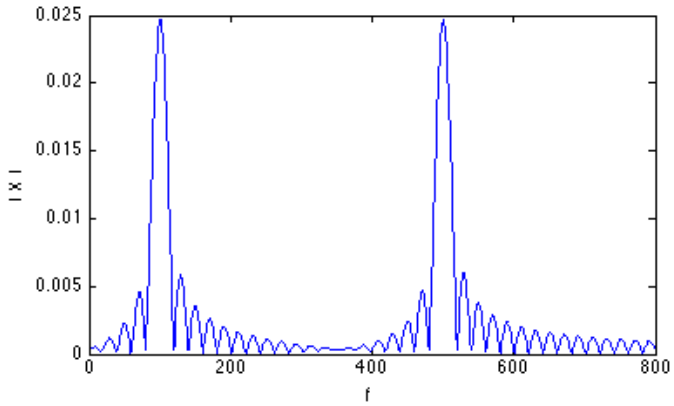
```
>> clear
>> N=250;
>> ts=.0002;
>> t=[0:N-1]*ts;
>> x=cos(2*pi*100*t)+cos(2*pi*500*t);
```

```
>> plot(t,x)
```



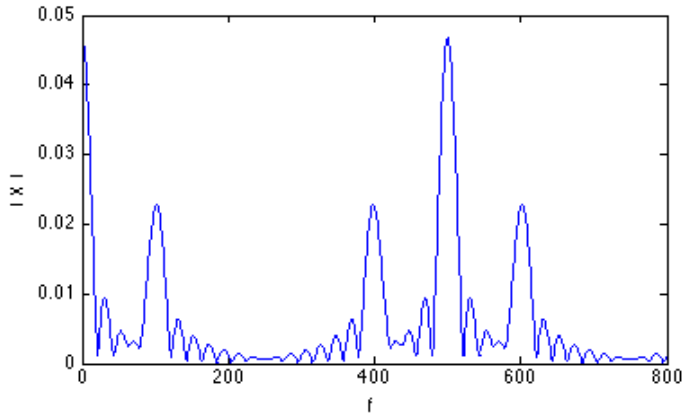
We can find approximate the Fourier transform integral for $0 \le f \le 800$ Hz using:

```
>> k=0;
>> for f=0:1:800
k=k+1;
X(k)=trapz(t,x.*exp(-j*2*pi*f*t));
end
>> f=0:800;
>> plot(f, abs(X))
```

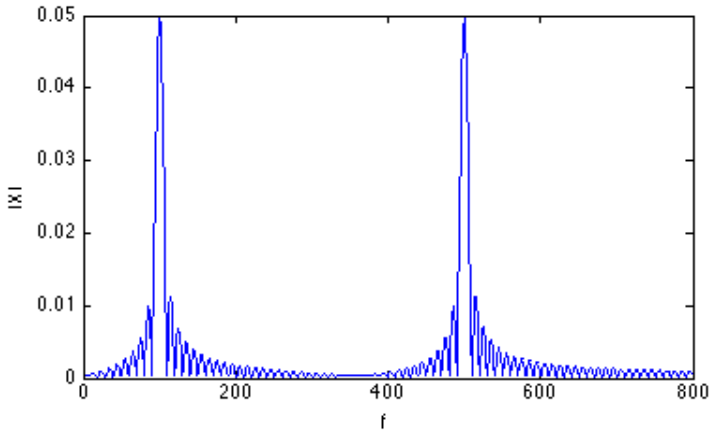


As expected the peaks in the spectrum are at 100 and 500 Hz the two frequencies contained in the signal. Theoretically, we expect to see impulse functions at these two frequencies and zero at every other frequency. This is not what we observe. This is because the MATLAB code only approximates the transform. There are three elements that make the results approximate. The sequence used to compute the transform is a sampled version of a continuous signal. The value of the sampling interval (Ts) and number of samples taken (N) affect the approximation. Additionally, the *trapz* command is using a summation to approximate the integral. We can observe the effects of the sampling interval and number of samples by changing their values.

For example, if we increase the sampling interval by a factor of 10 (to .002) and decrease the number of samples to 25 (so that we still cover the same range total time interval), and repeat the MATLAB code above we get:



In this case the approximation is less accurate. If we return the sampling interval to the original value of .0002 and increase the number of samples to 500, the approximation to the transform is found to be:



Note that the transform is more accurate than the original. This is expected because we are included more cycles of the waveform in the approximation (increasing the limits of integration).

The Discrete Fourier Transform (DFT)

An alternative to using the approximation to the Fourier transform is to use the Discrete Fourier Transform (DFT). The DFT takes a discrete signal in the time domain and transforms that signal into its discrete frequency domain representation. This transform is generally the one used in

DSP systems. The theory behind the DFT is covered in ECE 351. In that course you will find that the DFT of a signal can be used to approximate the continuous time Fourier transform.

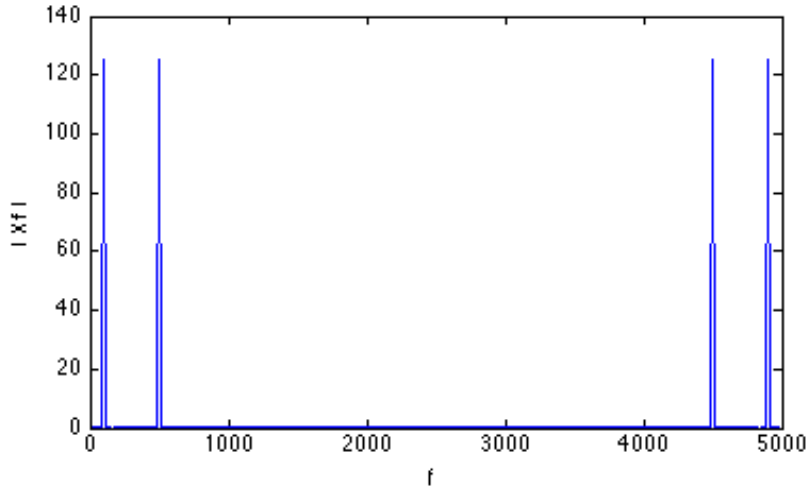
The Fast Fourier Transform (FFT)

Depending on the length of the sequence being transformed with the DFT the computation of this transform can be time consuming. The Fast Fourier Transform (FFT) is an algorithm for computing the DFT of a sequence in a more efficient manner. MATLAB provides a built in command for computing the FFT of a sequence. In this section we will discuss the use of the FFT to approximate the Fourier transform of signals. Recall that the DFT and FFT are discrete frequency domain representations of a discrete time sequence. In our examples, these sequences will be obtained by sampling continuous time signals.

In general, if a continuous time function, x(t), is sampled every T_s seconds until N samples are collected, the DFT/FFT of this sequence of length N is also of length N. The components of the resulting transform correspond to frequencies spaced every $1/(N*T_s)$ Hz. For example, using the same two-frequency signal x(t) used above we can produce a sequence of samples of length N = 250 spaced every T_s = .0002 seconds as shown previously.

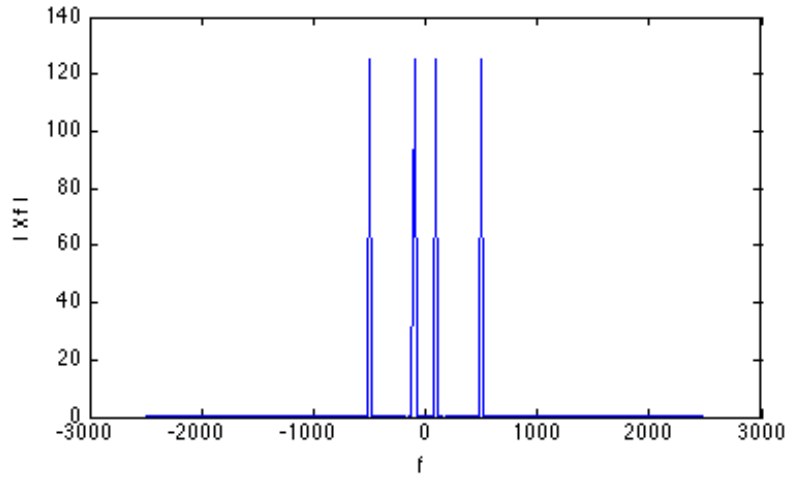
However, this time we will find the amplitude spectrum of this signal using the *fft* command. The resulting transform will contain N = 250 values. Since the frequency components are spaced every $1/(N*T_s)$ Hz these correspond to frequency values from 0 to $(N-1)/(N*T_s)$ Hz as shown below.

```
>> clear
>> N=250;
>> ts=.0002;
>> deltaf=1/(N*ts);
>> t=[0:N-1]*ts;
>> x=cos(2*pi*100*t)+cos(2*pi*500*t);
>> Xf=fft(x);
>> f=[0:N-1]*deltaf;
>> plot(f,abs(Xf))
```



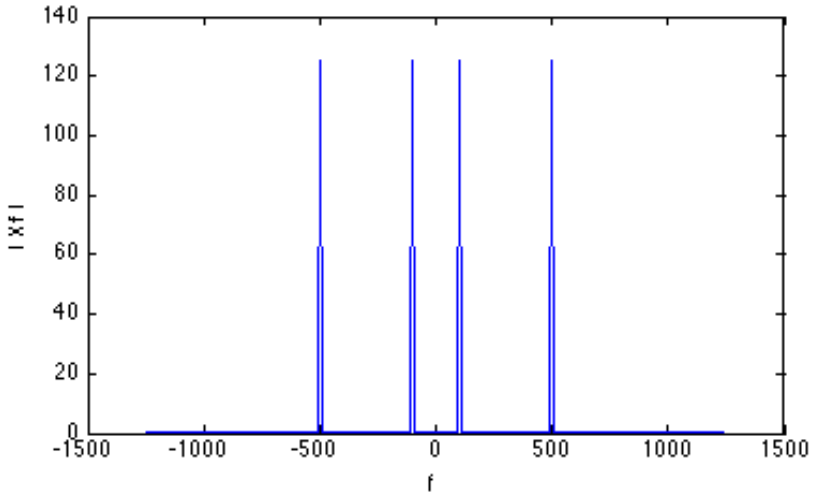
Note that the spectrum shows four components. Two are at the expected frequencies of 100 and 500 Hz. The other two are at 4500 and 4900 Hz, frequencies that do not appear in the signal. This is due to the periodic nature of the DFT. Only frequencies up to $0.5/T_s$ correspond to the actual frequencies in the Fourier transform. We can produce the spectrum plot that only shows these frequencies and shows the negative frequency components by applying the *fftshift* function as shown.

```
>> Xf_shift=fftshift(Xf);
>> plot([-N/2:N/2-1]*deltaf, abs(Xf_shift))
```



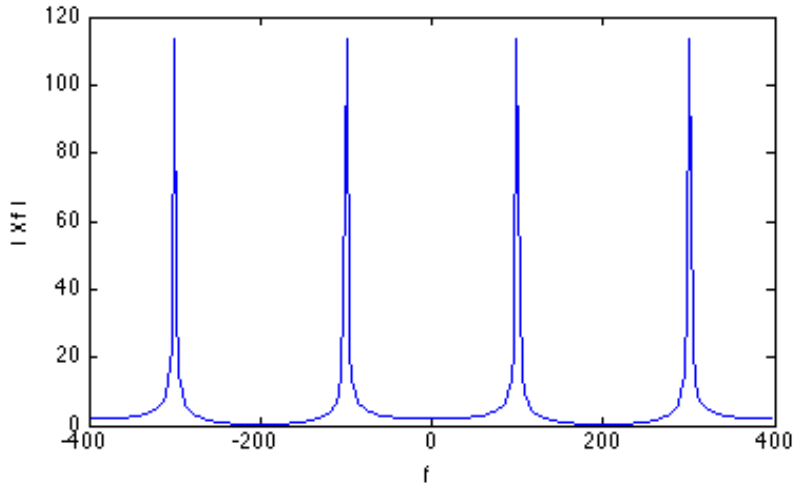
The values of the sequence length, N, and the time sampling interval, T_s, will have an effect on the accuracy of the spectrum that is calculated. First, we will increase the spacing between samples by a factor of two, or $T_s = 0.0004$. From the sampling theorem we know that the slower we sample, the lower the frequency that we can accurately represent. Repeating the FFT:

```
>> clear
>> N=250;
>> ts=.0004;
>> deltaf=1/(N*ts);
>> t=[0:N-1]*ts;
>> x=cos(2*pi*100*t)+cos(2*pi*500*t);
>> Xf=fft(x);
>> Xf_shift=fftshift(Xf);
>> plot([-N/2:N/2-1]*deltaf, abs(Xf_shift))
```



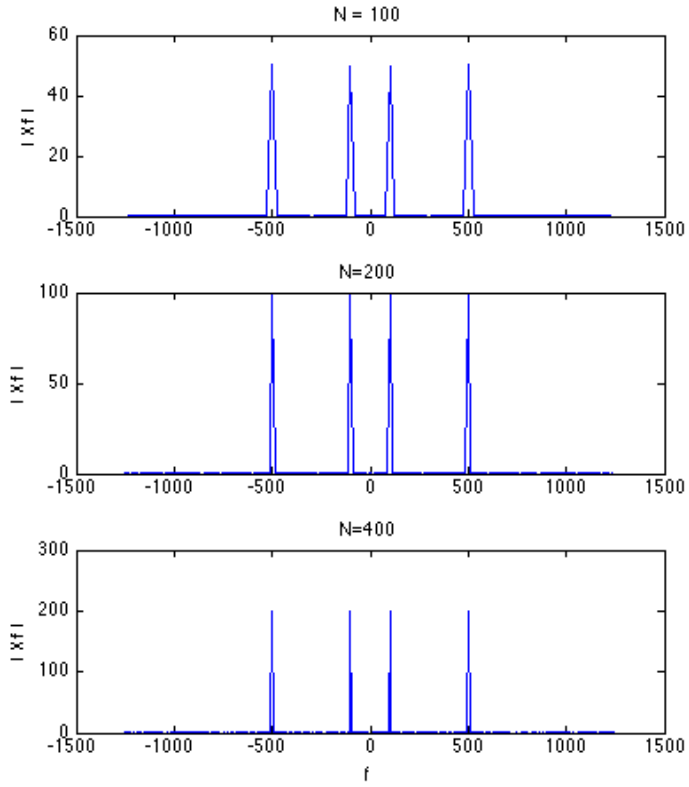
We observe that the maximum frequency is now 1250 Hz, instead of 2500 Hz as in the original computation and the frequency components of 100 and 500 Hz are still represented correctly. In general, the maximum frequency represented is given by $0.5/T_s$. Say we increase the sampling interval to $T_s = 0.00125$ as shown below:

```
>> clear
>> N=250;
>> ts=.00125;
>> deltaf=1/(N*ts);
>> t=[0:N-1]*ts;
>> x=cos(2*pi*100*t)+cos(2*pi*500*t);
>> Xf=fft(x);
>> Xf_shift=fftshift(Xf);
>> plot([-N/2:N/2-1]*deltaf, abs(Xf_shift))
```



It appears as if the signal, x, has frequency content at 100 Hz (correct) and 300 Hz (incorrect). This incorrect component is due to the aliasing effect and the fact that the signal has been sampled at too low of a frequency. The sampling frequency ($1/T_s$) always needs to be at least two times the highest frequency component in the signal being transformed, or in our example at least 2*500 Hz = 1000Hz or $T_s < 1/1000 = .001$.

Next we will observe the effect of N, the number of samples taken. With $T_s = .0004$ seconds we repeat the computation of the FFT with N = 100, 200, and 400 samples. The results are shown below.



Note that the effect of larger N is to increase the resolution of graph. This is due to the fact that the frequency spacing is given by $1/NT_s$, or in these three cases 25 Hz, 12.5 Hz, and 6.25 Hz, respectively. For a given sampling interval, T_s, as N is increased, the length of time that the continuous time signal is sampled increased (NT_s). Thus, we would expect that the resulting spectrum would have greater accuracy as shown here.

Guidelines for Using the *fft* Command

In general, when the *fft* command is used to produce the amplitude and/or phase spectrum of a continuous time signal values for N and T_s must be selected. The following guidelines should be helpful in this process:

1. Select T_s as large as possible but so that the highest frequency component in your signal is less than $1/2T_s$.
2. After determining the value of T_s, select N so that $1/NT_s$, the frequency resolution is small enough to accurately display your frequency components.