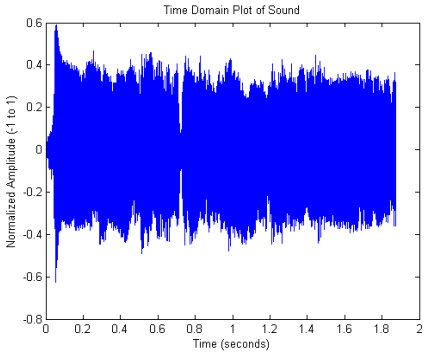
Fast Fourier Transform in MATLAB

In sound, frequency (pitch) and amplitude (related to volume) are the principle elements perceived by humans—both of which usually vary with time. Most sounds are composed of more than one frequency at a time. In music, this is known as a chord. If this were not the case, then we would be left with those annoying monophonic ringtones from the first cell phones. You know, the ones where there were only 10 ringtones from which to choose and the odds of more than one person having that annoying beeping rendition of Für Elise were very high. But I digress.

In MATLAB, if we plot the amplitude data vector against time we get a plot similar to the one below.




```
[data fs] = wavread('sweetchild2.wav');
t = linspace(0,length(data)/fs,length(data));
plot(t,data)
xlabel('Time (seconds)')
ylabel('Normalized Amplitude (-1 to 1)')
title('Time Domain Plot of Sound')
```

Time domain plot of the intro to "Sweet Child of Mine."

This plot shows us how long the sound is (2 seconds) and how the amplitude (effectively related to volume) varies over time. But it also can be very useful to know how each frequency component of the sound contributes to the overall amplitude. For example, in most music the melody line has larger amplitude than the other frequency components of the sound. Think of it as if 3 different instruments played 3 different notes (frequencies) each at a different volume. We can distinguish which instrument was playing which frequency at what volume using MATLAB. This can be done by doing a Fourier Analysis of the sound data—a transform from the time domain to the frequency domain.

Magnitude of the Complex Amplitude

In MATLAB, there is a built in function, `fft( )`, which can be used to analyze the frequency components of a sound. More specifically, for each frequency we can measure two values: amplitude and phase. Phase is a more complicated topic covered in other courses. It is very common for these two values be represent together as one complex value called a *complex amplitude* or *phasor*. This representation is derived from Euler's Relation (not very relevant to this course, but for arguments sake it is shown below).

$$\mathrm{A}\cos(2\pi f) = \frac{1}{2}Ae^{i2\pi f} + \frac{1}{2}Ae^{-i2\pi f}$$

Don't worry if you don't understand this. It is not too important.

Because it is a complex number, it has both real and imaginary parts. For this course, we generally are only concerned with the overall magnitude of the complex amplitude. Note that the real and imaginary parts are not amplitude and phase, respectively. Therefore, it is incorrect to use the `real( )` function instead of `abs( )` to achieve the magnitude of the complex amplitude. Additionally, the `fft()` function returns the complex amplitudes scaled by the overall length of the data. Thus, we'll need to divide by the length of the data. Finally, we note from the above equation that we need to multiply the magnitude by a factor of 2 to achieve the original amplitude, A. All of this combines to give us the following line of code.

```
y=2*abs(fft(data))/length(data);
```

From this code, `y` will be a vector of the magnitudes of each frequency. However, the second half of this vector contains data not relevant to the course. So the next line we run is:

```
y = y(1:end/2);
```

Nyquist Frequency

The frequencies that correspond to these magnitudes can be made by the following line of code (where `Fs` is the sampling frequency of the sound).

```
x=linspace(0,Fs/2,length(y));
```

These frequencies vary from 0 hertz to Fs/2 Hz. Fs/2 is also known as the Nyquist Frequency—or the maximum frequency that can be perceived in the sound at a sampling frequency of Fs. In other words, if you try to record a sound with a frequency of 10 kHz in it at a sampling frequency of 15 kHz, the sound will not be accurately played back, according to Nyquist's Theorem.

We are now ready to plot frequency versus magnitude:

```
plot(x,y)
```

Principle (Strongest) Frequency

It is often useful to know what the strongest (loudest) frequency in a sound is. In our example of the three musicians playing 3 different notes this would indicate which musician was playing the loudest. This can be done by taking the `max( )` of the `y` vector which gives us the value of the strongest magnitude and the index at which it occurs. We can then look up the frequency that goes along with that magnitude by indexing the `x` vector with that index:

```
[y_max index] = max(y);
f_principle = x(index);
```

We can modify our plot command above to include a circle around the Principle Frequency:

```
plot(x,y,f_principle,y_max,'o')
```

Summary

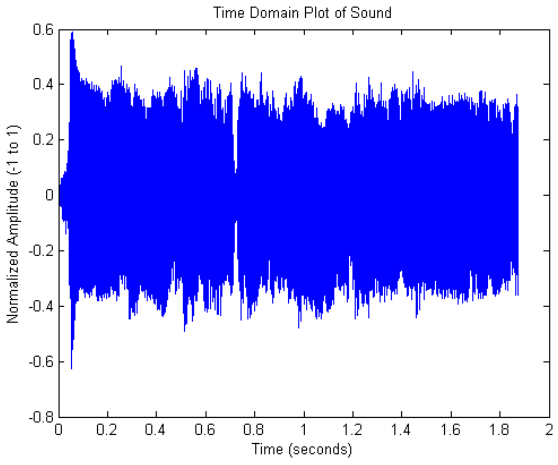
All of the above can be difficult to understand. Some of it is even beyond the scope of our course. However, it provides a basis for understanding frequency domain analysis of sounds in MATLAB. Below is an outline of what you should "take home" about the FFT:

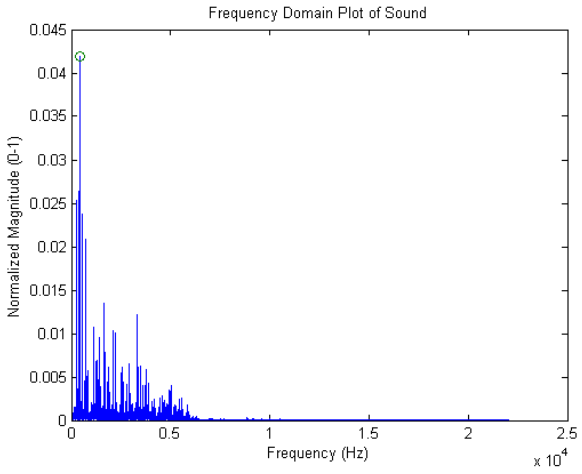
- Sounds often contain more than one frequency at once.
- Each frequency component can contribute a different magnitude to the overall amplitude of the sound.
- Time domain plots say nothing about the frequency components of a sound. A Fourier transform is needed to do frequency domain analysis. The `fft( )` function in MATLAB does this for us.
- The output of the `fft( )` function by itself is a vector of complex numbers.
- ```
  y=2*abs(fft(data))/length(data);
  y = y(1:end/2);
  ```
  - returns a vector of the magnitudes of each of the frequencies' contributions to the sound's amplitude (for the frequencies we care about in this course)
- The highest frequency that can be perceived in a sound is given by the Nyquist Frequency:
  ```
  f_nyquist = Fs/2;
  ```
- The frequencies that correspond to the `y` vector range from 0 Hz to the Nyquist Frequency and can be generated by:
  ```
  x = linspace(0,f_nyquist,length(y));
  ```
- The strongest (loudest) frequency (`f_principle`) and its corresponding magnitude (`y_max`) can be determined by the following:
  ```
  [y_max index] = max(y);
  f_principle = x(index);
  ```
- Frequency versus magnitude can be plotted (along with the Principle Frequency circled) by:
  ```
  plot(x,y,f_principle,y_max,'o')
  ```
- The following code will make 2 plots: time versus total amplitude and frequency versus frequency component magnitude.
  ```

Completed Code

```
%Read in the sound
[data Fs] = wavread('sweetchild2.wav');
%Generate a time vector that corresponds with data
t = linspace(0,length(data)/Fs,length(data));
%Plot time versus amplitude data
plot(t,data)
xlabel('Time (seconds)')
ylabel('Normalized Amplitude (-1 to 1)')
title('Time Domain Plot of Sound')

%Analyze the frequency components.
y=2*abs(fft(data))/length(data);
%Remove non-relevant data.
y = y(1:end/2);
%Calculate the maximum frequency that can be perceived.
f_nyquist = Fs/2;
%Generate the frequency vector that corresponds with y
x=linspace(0,f_nyquist,length(y));
%Find strongest magnitude
[y_max index] = max(y);
%Look up the frequency that corresponds with the strongest magnitude
f_principle = x(index);
%Plot frequency versus frequency componant magnitude, circle f_max
figure
plot(x,y,f_principle,y_max,'o')
xlabel('Frequency (Hz)')
ylabel('Normalized Magnitude (0-1)')
title('Frequency Domain Plot of Sound')
```





Note: the sampling frequency for this sound was 44,100 Hz. The Principle Frequency was about 415Hz which is approximately a "G#" note, the loudest note played in this recording of the intro to "Sweet Child of Mine" by Guns N' Roses.