Lab 7

Convolution and the FFT

In this lab we will be playing around with the FFT as well as using two different methods of convolution. We will also use the `tic` and `toc` commands in Matlab to keep track of how long our code takes to execute.

The FFT

As you may know, the Discrete Fourier Transform, or DFT is a way of finding the frequency spectrum of a signal. It has an enormous range of uses from performing signal analysis to being an integral part of perceptual codecs such as MP3 and AAC. The Fast Fourier Transform, or FFT is simply a more efficient way of calculating the DFT, the results are identical. While there are many FFT algorithms in existence, and it is outside the scope of this class to explain how they work, the basic idea is that they take advantage of mathematical symmetry to reduce the number of calculations required. A few things to know about the DFT/FFT:

1. The number of samples given as the input is the number of frequency bins obtained at the output.
2. The frequency bins are equally spaced around the unit circle (including both positive and negative frequencies.
3. The first bin is always 0 Hz, or the DC component.
4. The spacing (in Hz) between frequency bins is: (sampling frequency) / (length of input in samples)
5. The results should be interpreted with care, as the FFT always operates on a linear frequency scale, yet our ears operate on a logarithmic frequency scale.
6. There is an inherent trade off between frequency resolution and time resolution. Long FFT's give good frequency resolution but poor time resolution. Short FFT's are vice versa.
7. FFT's are much faster when the input is a power of 2 in length (i.e. 1024, 2048, 4096, etc). The `nextpow2()` function can be very helpful in finding the best size for your signal.

Convolution

Convolution, as we will use it in this class is a way of applying a linear system to a signal. A linear system in this case could be an equalizer, a speaker, a room, or dozens of other things. The important property of a linear system is that it can be completely characterized by its impulse response. We encountered the idea of the impulse response in the previous lab when we used an impulse to excite the Karplus-Strong model. In the world of digital audio, an impulse is simply one sample at maximum amplitude (+1.0) followed by all zeros. The output of the system in response to an impulse is called, unsurprisingly, it's impulse response. It's a very convenient tool to use, because it tells us everything we need to know* about the system. Thus, if we want to hear how a recording would sound if it had been recorded in a particular room, all we need to do is convolve this signal with the impulse response of the room. In this lab we will be doing just that. A few things to know about convolution:

1. Your book will give you a more detailed description of how convolution works. It is important to keep in mind that convolution requires a **large** number of computations, and grows exponentially with the lengths of the inputs.
2. Convolution operates on two signals, which we will think of as the input and the impulse response we are applying to that input. In actuality though, the order of inputs doesn't matter. A convolved with B is the same as B convolved with A.
3. The convolution of two signals is always longer than either of its inputs. An input of length L convolved with an impulse response of length M will have an output of length L+M-1.

*** yes there are some caveats to this statement, but for our purposes it is true**

The Convolution Theorem

Simply stated, the Convolution Theorem tells us that multiplication in the frequency domain is the same as convolution in the time domain. Consider two things we know:

1. Convolution is a very computationally intensive task.
2. The FFT gives us a tremendous saving in computation when converting between the time domain and frequency domain.

Hopefully you see where I'm going. If we have large sets of data to convolve (such as audio signals), we can do it much faster by taking the FFT of both signals, multiplying them together, and then taking the IFFT of the result. Done correctly, this will give us the same result as good old-fashioned convolution but in much less time.

Timing Scripts in Matlab

When we start to do more complicated things on longer bits of audio, the time needed to do the processing can become non-trivial. Two of the very helpful commands in Matlab are `tic` and `toc`, which make it easy to see how long it took to run a series of commands. It's quite simple: when you want to start timing something, you give the `tic` command. When you want to stop the timer, you give the `toc` command. Matlab will then print the elapsed time to the command window.

```
tic;
% ...do a bunch of stuff
toc;
```

Display:

```
Elapsed time is 3.153909 seconds.
```

Assignment:

Problem 1:

One of the most important skills as a programmer is learning to read and understand existing code. Look again at the magPlot function. Copy it into your write-up and explain what each line does. Be as descriptive as you can, and do your best to explain "why" instead of just "what." (Note: you do not need to explain the "`set`" commands)

Problem 2:

Read in a **few seconds** of audio from your favorite song (mono signal only) and use magPlot to identify what the most prominent frequencies are. How do these relate to the key or chords of the song? Include the plot in your write up, along with the wav file of the audio.

Problem 3:

a) Write a convolution function based on equation 7.6 on page 186 in the Steiglitz book. Use small sets of numbers and test it against the built in function `conv`.

Hint: Think of making delayed and scaled versions of the "impulse response" input which are superimposed to create the output.

b) Write a convolution function utilizing FFT's and the Convolution Theorem. Test it against the `conv` function. **Remember that each input will need to be zero padded to be at least as long as the expected output.**

c) Use this impulse response and convolve it with this snare hit using the functions you wrote in parts A and B respectively. Use the `tic` and `toc` commands to time how long each one takes. Include these times in your write up, as well as one of the output wav files (the output **should** be identical from the two functions).

Note that for the non-FFT case this could take a minute or more for your machine to process, so be patient.

d) Given your results from part C, what is the advantage of using the FFT to perform convolution? Are there any disadvantages?

Useful commands:

`fft()` Compute the FFT (DFT) of an input. Note that you can give it a second argument with the length of FFT you want to use. It will then do any zero-padding or truncation necessary. Very convenient!

`ifft()` Inverse FFT.

`nextpow2()` Find the nearest 2^X which is greater than or equal to the input, check the documentation for the correct usage.

`tic;` Start timer.

`toc;` Stop timer and report time elapsed.

`conv()` Convolve two signals.

11/17/13 MMI502

Lab 7 – Convolution and the FFT

Nate Paternoster

Part 1 – Analyzing the *magPlot* function

```
function magPlot(input, fs, plotTitle)
% function magPlot(input, fs, plotTitle)
% magPlot takes in an input vector (one channel only!)
% and plots the magnitude spectrum in dB
%
%    input:       input signal
%    fs:          sampling frequency of input signal
%    plotTitle:   string to use as the title of the plot
%
% Example:  magPlot(mySignal, 44100, 'Frequency Spectrum');

fftLen = 2^nextpow2(length(input));
```

The Fast Fourier Transform works most efficiently with 2^N amount of data. It works similarly to mergesort, which is most efficient when it can divide all the data exactly evenly. Having the Fourier series length be the next highest power of 2 after the input matrix length will allow all of the input data to be represented while being most efficient to process.

```
halfLen = fftLen/2;
bin1Freq = fs/fftLen;
```

Bins are the equivalent of samples of the input waveform. Each bin represents a frequency component of the waveform. The very first bin will be the input waveform's sample rate (for example 48k) divided by the total length of the Fourier series representing the input data (for example 24k). In this example the first bin would be 2Hz and would allow us to see the amount of 2Hz sinusoid present in the waveform.

```
freqVec = 0:bin1Freq:(halfLen-1)*bin1Freq;
```

We can create the entire vector of bins to be shown on the frequency domain graph. We are only concerned about representing the first half of the frequency domain because the second half is past the Nyquist frequency and won't be able to be represented or heard in the waveform. The first bin's frequency will be the space between each bin; therefore, this FFT will use evenly spaced bins.

```
inputFFT = fft(input, fftLen);
```

This will perform the transform on the input waveform vector data up to the Fourier series length calculated earlier.

```
inputFFT = abs(inputFFT(1:halfLen));
```

This will take the magnitude of each bin.

```
inputFFT = inputFFT/halfLen;
```

This will remove the second half of the frequency domain vector. All the bins past the Nyquist frequency will be removed.

```
inputMagDB = 20.*log10(inputFFT);
```

The magnitudes of each bin will be converted to dB values.

```
inputMagDB(inputMagDB < -100) = -100;
```

Any bins that have a magnitude greater than -100 will be set to -100. This is to keep the graph readable since there may be some bins that have –infinity magnitude.

```
figure;
plot(freqVec, inputMagDB);
```

```
axis([20 fs/2 -60 0]);
grid on;
xlabel('Frequency (Hz)');
ylabel('Amplitude (dBFS)');
title(plotTitle);
set(gca, 'XScale', 'log');

set(gca, 'XTick', [20, 50, 100, 200, 500, 1e3, 2e3, 5e3, 10e3, 20e3]);
```

Part 2 – Identifying the frequency content of a wav file

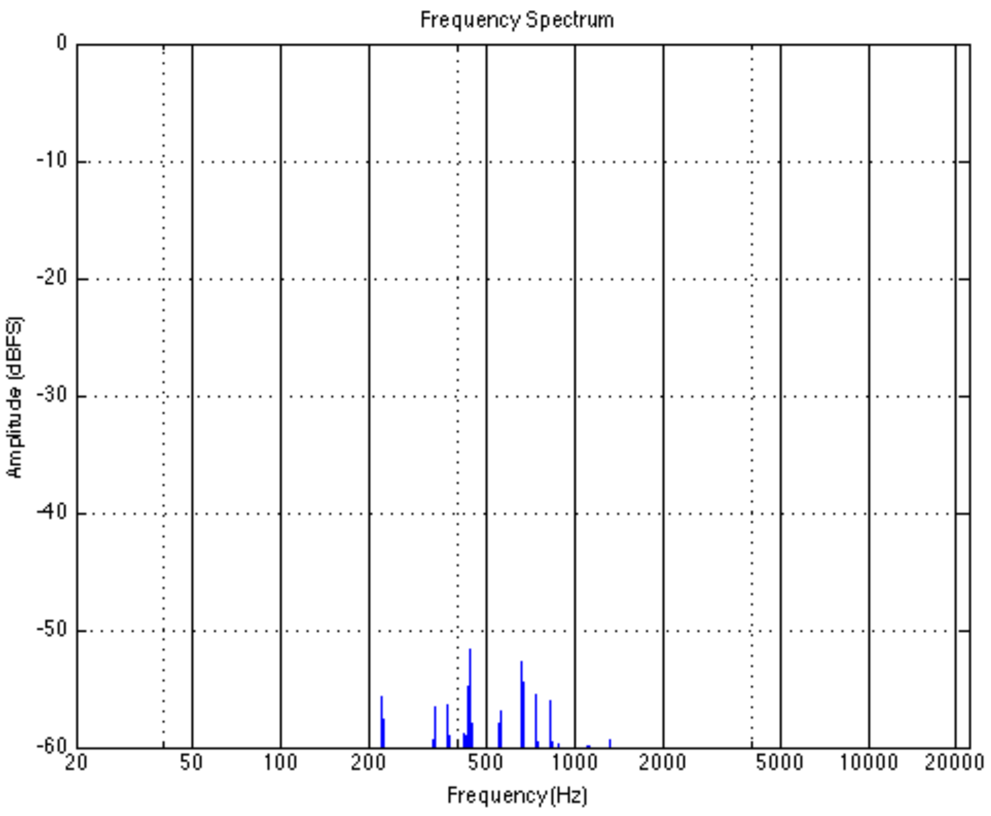

Figure 1: The frequency spectrum of the wav file read in.

The wav file was read in using wavread and plotted using magPlot. The corresponding bins are shown on the x-axis and the amplitude in dB is shown on the y-axis. The most prominent frequencies appear to be 440Hz, 660Hz, 220Hz, 740Hz, and 830Hz. These correspond to the pitches A4, E5, A3, F#5, and G#5. The beginning of this song is an Amaj7 chord.

Part 3 – Writing convolution functions

a. *Write a convolution function based on equation 7.6 on page 186 in the Steiglitz book. Use small sets of numbers and test it against the built in function conv.*

The function here was based on the equation:

$$y_t = \sum_{k=0}^{t} x_k h_{t-k}$$

To test this function I used [0, 0, 0, 1] as the input vector and [1, 1, 1, 1] as the impulse response vector. Using both my function and Matlab's built-in conv function the resultant vector ended up being [0, 0, 0, 1, 1, 1, 1] in both cases.

b. *Write a convolution function utilizing FFT's and the Convolution Theorem. Test it against the conv function.* ***Remember that each input will need to be zero padded to be at least as long as the expected output.***

The convolution theorem states that convolution in the time domain is equivalent to multiplication in the frequency domain. In order to make this convolution function work we will first pad the input and impulse vectors to be equal lengths. Then they will undergo the FFT. The vectors will then be multiplied together and finally the resultant vector will undergo the inverse FFT to be brought back into the time domain.

After testing this function against Matlab's conv using the same input and impulse response vectors as before, I arrived at roughly the same result as before.

c. *Use this impulse response and convolve it with this snare hit using the functions you wrote in parts A and B respectively. Use the tic and toc commands to time how long each one takes. Include these times in your write up, as well as one of the output wav files (the output* ***should*** *be identical from the two functions).*

The function from part (a) took 306.1796 seconds to convolve the snare hit with the impulse result. The function from part (b) took only 0.0163 seconds in contrast.

d. *Given your results from part C, what is the advantage of using the FFT to perform convolution? Are there any disadvantages?*

Using the FFT provides a much quicker method to perform convolution. However using the FFT also produces quantization errors because the signal can only be analyzed at discrete, evenly spaced bins and cannot be analyzed continuously. In addition, the FFT must contain 2^N amount of data to be used efficiently.