Connor McGwire, Mariana Chagoyan
Special thanks to project group members Griffin and Geoff for help with first section
CSS 457 – Multimedia Signals
Lab 07

Handling Feedback Filters

Introduction

In this lab, we analyzed how DFT (Discrete Fourier Transform) works by computing all N values to 4N^2 complex multiplications and 4N^2 -2N complex additions resulting an O (N^2) algorithm. Based on the inefficiency of this computation, FFT (Fast Fourier Transform) is used to improve the efficiency of the algorithm by decomposition an N point DFT into successively smaller DFT's with a length N as the sum of two DFT'S each of length N/2, in a manner similar to merge sort. One of the two is formed from the even-numbered points of the original N, while the other uses the odd numbered points.

In addition, we also used the spectrogram tool to analyze a signal's contributing frequencies over time, by computing DFT windows over the length of the signal. We also identified problems that may occur using the spectrogram to analyze signals.

Build it Faster

To implement our own FFT function, we consulted Algorithm 6.2 in the lab book for the skeleton, then determined how to work in a real indexed array. The final code is thus (for a non-padded FFT, and thus brittle function):

```
function [Y] = myFFT(x)

  indices = [0 : length(x)-1];
  revIndices = bin2dec(fliplr(dec2bin(indices, 8))); % bit reversed indices
  revX = x(revIndices+1);
  temp = zeros(1, length(revX));
  N=2;

  while (N <= length(revX))
    for n = (0:length(revX)/N - 1)
      for k = (0:N - 1)
        subOff = (N * n);
        even = (mod(k,N/2)) + subOff + 1;
        temp(k + subOff + 1) = revX(even) + (exp(-1i * k * 2 * pi / N) * revX(even + (N/2)));
      end
    end
    revX = temp;
    N = N * 2;
  end
  Y = revX;
end
```

When checked with a standard sinusoid against MATLAB's standard FFT function, we achieved nigh on identical results. This can be seen in Figure 1. As for complexity, our while loop executes log N times, the outer for loop executes N / size times in a while loop, and the inner most for loop executes size times. Multiplied together we get:

$$O\left(\frac{N}{size} size\ logN\right) = O(NlogN)$$

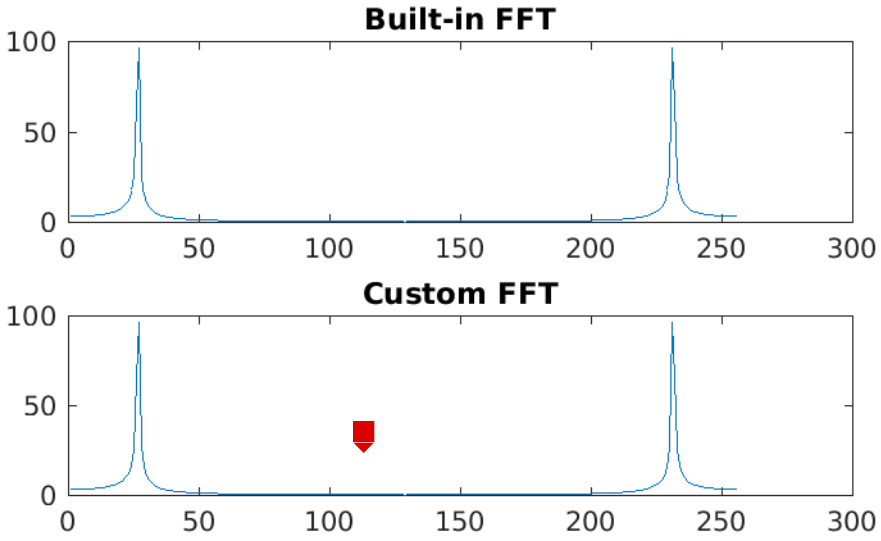

Figure 1

Examining the Results

To test the FFT function, we created analog signals with frequencies $\frac{256}{2}0.19 = 24.32Hz$ and $\frac{256}{2}0.13 = 16.64Hz$. Combining them together and taking their Fourier Transforms with the fft() function, we get an array of values with indices at 18 and 25 (Figure 2; upper graph) which, taking some computational error into account, corresponds pretty closely to the composing frequencies of the wave. Doing it again with a 0.13pi and 0.14pi frequency, we instead get one slightly thicker peak (Figure 2; lower graph) likely due to having too low of a resolution to properly distinguish frequencies that are so close together. Taking the FFT of various DTMF 'buttons' we can get the composing frequencies as sharp peaks on the graph (Figure 3).

It should be noted that we finally figured out how to properly display our results as one sided graphs of amplitude at frequency!

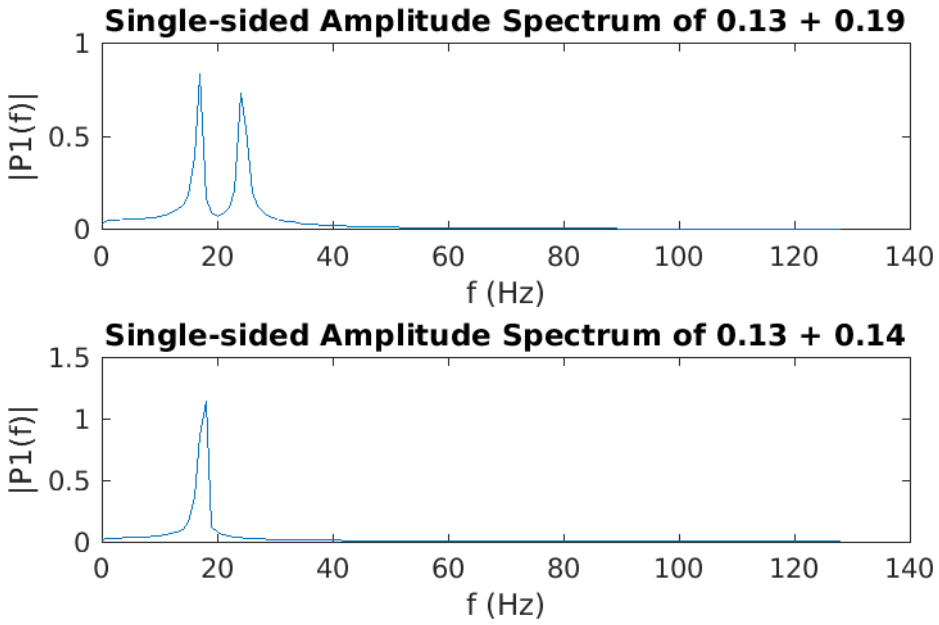

Figure 2

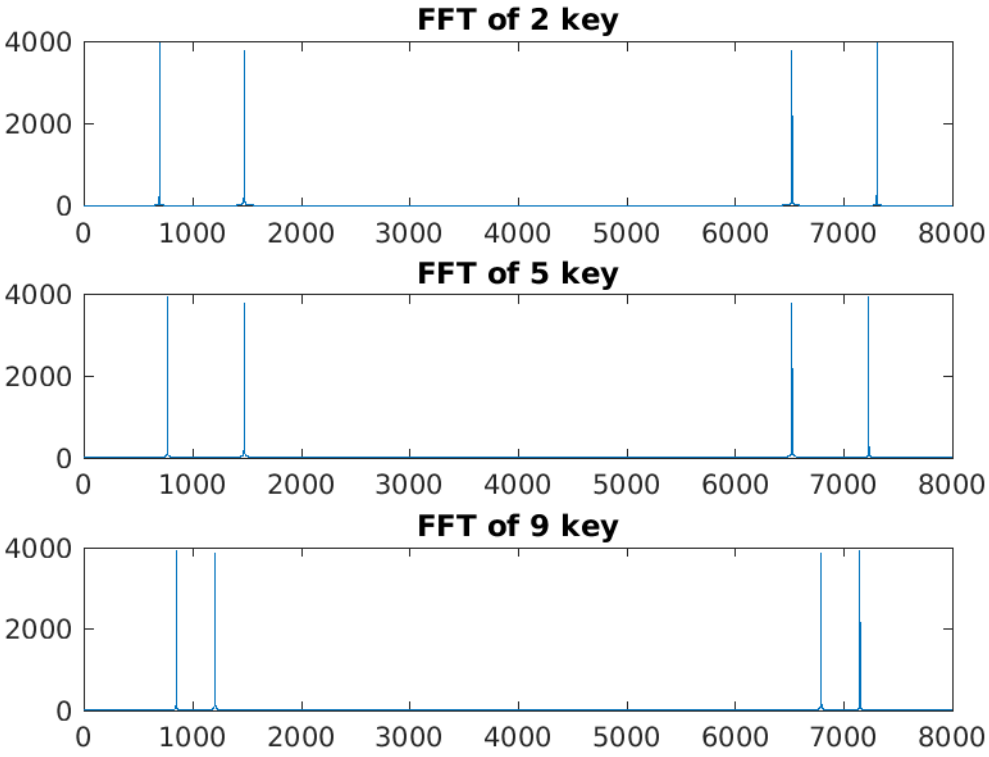

Figure 3: Two sided absolute values of raw output

Adding Some Color

To take the concept of a DFT to create a spectrogram, we made use of the myspecgram function and fed in some values from our previously created DTMFCoder function. We started by giving it a single wave created from pressing the '1' key on the DTMF creating the spectrogram in Figure 4. We can very clearly see both frequencies that compose the wave as thin orange lines going down the image. When concatenated together, the outputs of multiple DTMF 'buttons' creates a spectrogram consisting of the individual images concatenated together (Figure 5). One matter of note are the vertical streaks at the concatenation points. This is likely an artifact of windowing over a very sharp change in data.

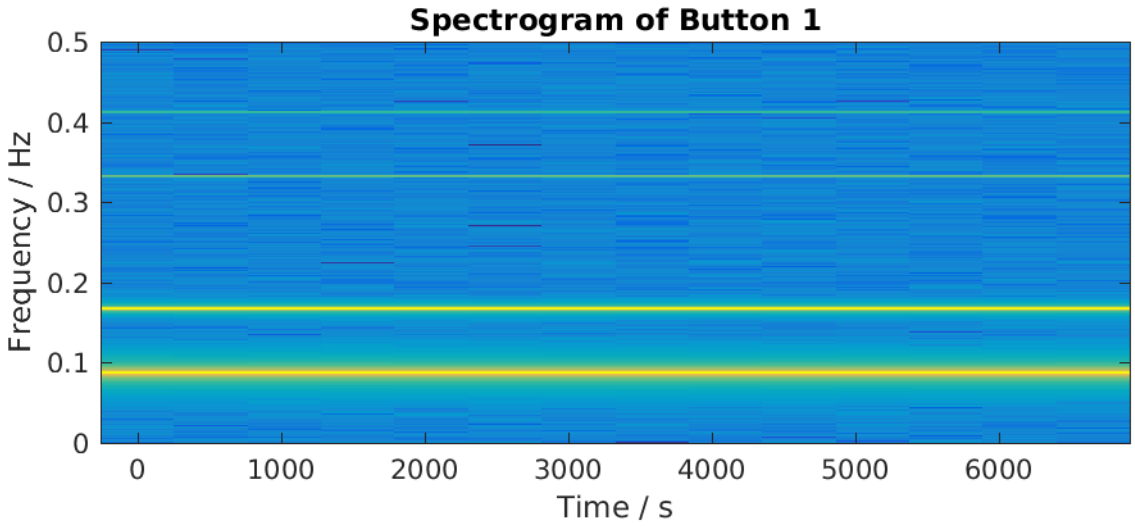

Figure 4

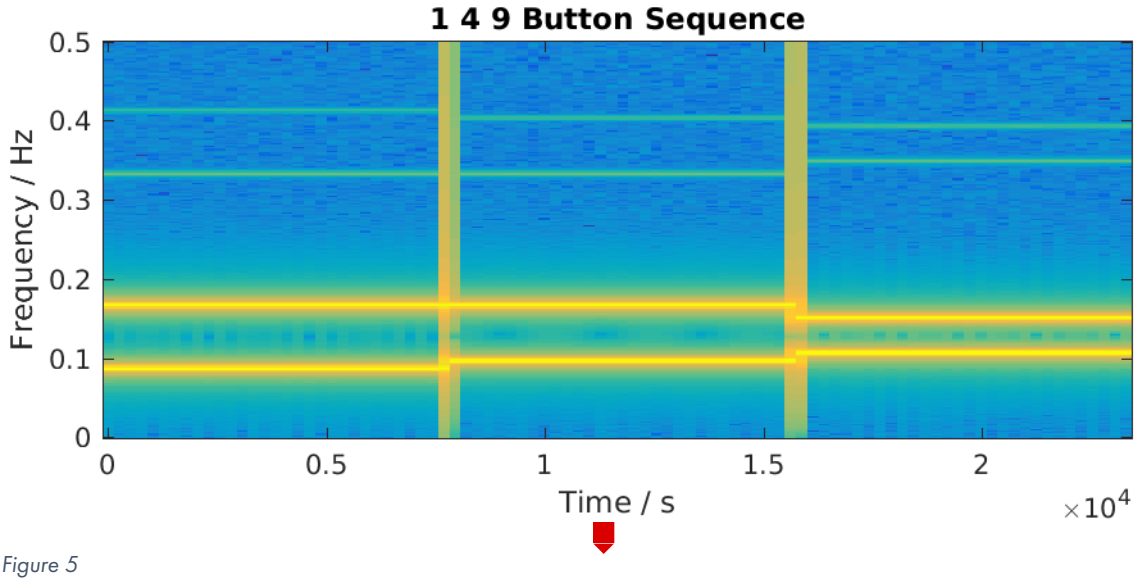

Figure 5

Conclusion

We learned that the FFT algorithm is a faster way of doing DFT because it divides the problem in smaller, but equal, parts and then it applies that strategy to all the divided parts, rather than working with the whole signal repeatedly. This process of dividing reduces the complexity from N^2 to N*logN. When identifying problems using the DFT to analyze signals, it's not always good to extend the window length to get a better frequency resolution because, depending of the signal, a longer segment might provide worse spectral information. The spectrogram method deals with signals that vary over time and gives us a compromised result for both the time and frequency domain. Large windows grab more frequency information, but obscure the time component of those frequencies. Whereas grabbing small time frames gets a less accurate picture of the composing frequencies.