Eugene Davis
CPE 381
04/21/14
Project Bonus

CPE 381 Project Bonus

1. Problem Description and Solution

The bonus portion of the project is to apply a Fast Fourier Transform to the input WAV files, and then interpret the results programmatically to provide the dominant frequency of the file. This was performed by taking an existing Fast Fourier Transform created by Jon Harrop and calling it on a series of time windows (measured in number of samples) dividing the input into equal pieces. Each window was 4096 samples long (2^{12}) allowing it to meet the power of two input requirement of the Fast Fourier Transform. All the samples within each window were multiplied by the Welch window function, and each window overlaps the second half of the previous one.

2. Design

In order to process the file, a mostly procedural C++ program was written with the following major features.

1. Input Handling

The first thing the program does is handle user input from the command line. Unlike previous phases, this program has few input constraints, since the Fast Fourier Transform algorithm does not have to know about the sample rate. As a result the only checks here are for sample size, and to ensure that the WAV filename input is for a valid WAV file.

2. Timing and Opening Summary File

Before the timer is started, the summary file is opened and has the first few pieces of information written. This is to accommodate the fact that the summary file will be written to during the processing in order to output data for each channel as it is calculated. After this, the start time is recorded.

3. Reading Samples and Applying Fast Fourier Transform

Before starting the main loop of the program, the array to store the values of the Welch window is filled with the appropriate values. This is done by simply looping through the array, which is size 4096 and of type double, and applying the equation for Welch's on each location in said array.

$$w(n) = 1 - \left(\frac{n - \frac{N-1}{2}}{\frac{N+1}{2}} \right)^2$$

Welch Window Equation (a parabolic section)

With this done, the main loop is entered. The outer loop iterates through all the samples in the WAV file, incrementing the counter in steps that are half the size of the Fast Fourier Transform length (4096, therefore steps of 2048). Each iteration of this loop fetches the all the samples that will fit within the Fast Fourier Transform length, padding out the final iteration with zeros. Each sample (on a channel by channel basis) is multiplied by the relevant portion of the Welch Window array, and then stored in a vector object. There is one vector object per channel, allowing each channel to be processed individually.

Once all the samples for a window are read in, each channel's vector object is run through the Fast Fourier Transform function. The results of calling this function are further processed, with the maximum index for each window for each channel being stored in another vector object.

4. Finding the Dominant Frequency

Now that the maximum index of the Fast Fourier Transform for each time window has been found for all the channels, a loop is entered to find the dominant frequency. This is performed in two ways, by finding the median of the maximum indices and by finding the average value per channel for the maximum indices. In this section, both ways of finding the dominant frequency are written out immediately to the summary file, rather than having to buffer them for output later.

5. Completion of Summary File

Finally, the sampling frequency, recording length, and processing time are all written out to the summary file.

Design Decisions

Several major trade-offs were made in the course of this portion of the project. The major ones were related to maintainability of the code versus development time, calculation of the dominant frequency and finally optimization versus development time. All are described below.

1. Maintainability vs. Development Time

Since all the required information for making the Bonus portion of the project was not available until near its due date, the code for this final section is not as modular as in Phases 1 and 2 of the project. For example, the calculation of the Welch Window array is done in the main function rather than its own function, as is other complex functionality such as the processing to discover the dominant frequency. With fewer upcoming assignments and exams, this would be the first area to target for improvement.

2. Calculation of the Dominant Frequency

There are multiple ways to calculate the dominant frequency from the maximum indices of the Fast Fourier Transform. Two ways were selected here, the median and the mean, with the median yielding better results (discussed in the Program Performance section). The median is a more complex formula, and thus requires longer time to calculate than the mean, but seems to provide a much more reliable result.

The median provides the better result because it effectively disregards the outliers that drive the mean very high or very low (generally very high).

3. Optimization (for speed) vs. Development Time

As in the first trade-off section, optimization was also sacrificed due to limited time. In particular, built-in C++ data structures (objects) were used, which can dramatically decrease time over using arrays. This made code development much quicker, and given that the performance time was still real-time (discussed in the Program Performance section) this was an acceptable trade-off.

4. Padding Last Window with Zeros

In order to meet the power of two requirement, the last window had to be padded with zeros. This approach was taken, since it allowed the window sizes to remain uniform, and since it only affects the final time window in a file where the total number of samples is much larger than the window size, it should have limited effect on the end result.

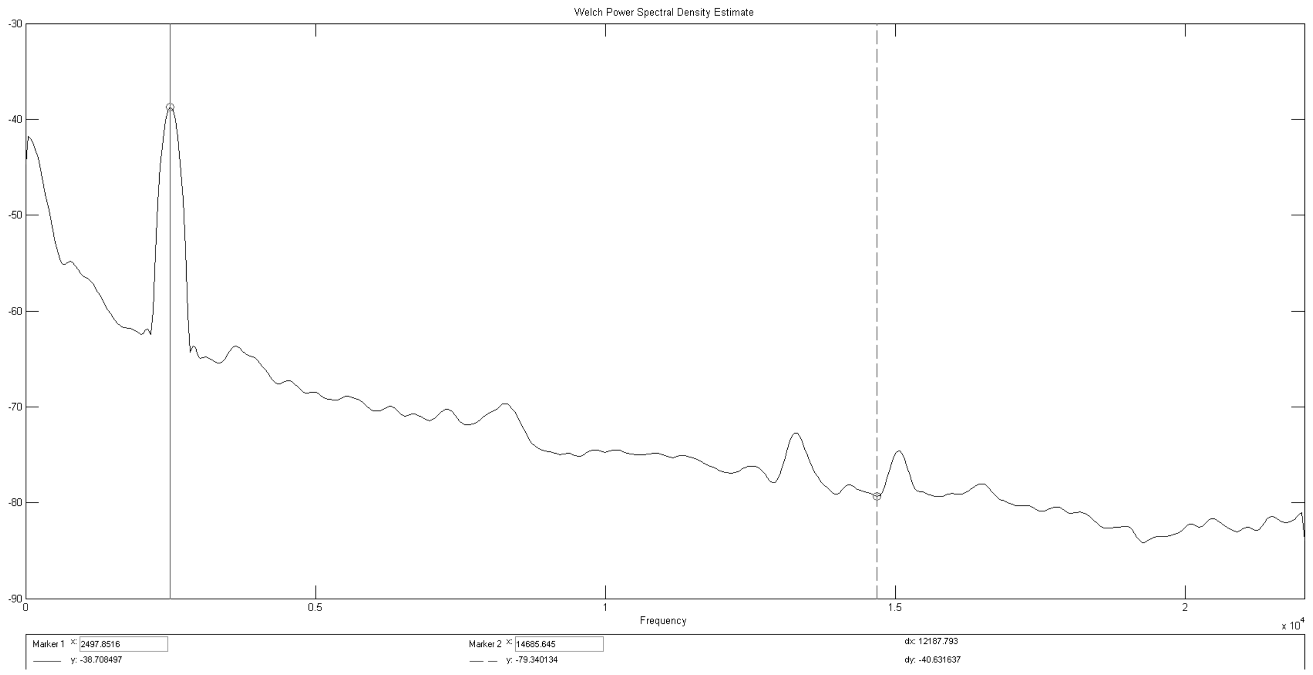
5. Length of Time Window

Although the time window must be a power of two, it can be either long or short. Short windows provide better time resolution, but a longer one gives better frequency resolution. This is because a shorter window will have more frequency leakage, because it is further from having infinite support. The time window selected here provided a reasonably accurate result for periodic signals, without slowing the program down too much.

3. Results

The program was able to handle periodic signals well, for example the result from running the noisy (with 2500 Hertz) WAV file was a 2497.85 Hertz dominant frequency for both channels (for median, mean proved a much poorer approach, yielding 8409.68 and 9148.66 Hertz for the different channels). This value is quite close to the result provided by Matlab for the Welch spectra, which yielded a dominant frequency of 2497.8516 Hertz, suggesting that the approach Matlab is using for periodic signals is similar.

Eugene Davis
CPE 381
04/21/14
Project Bonus

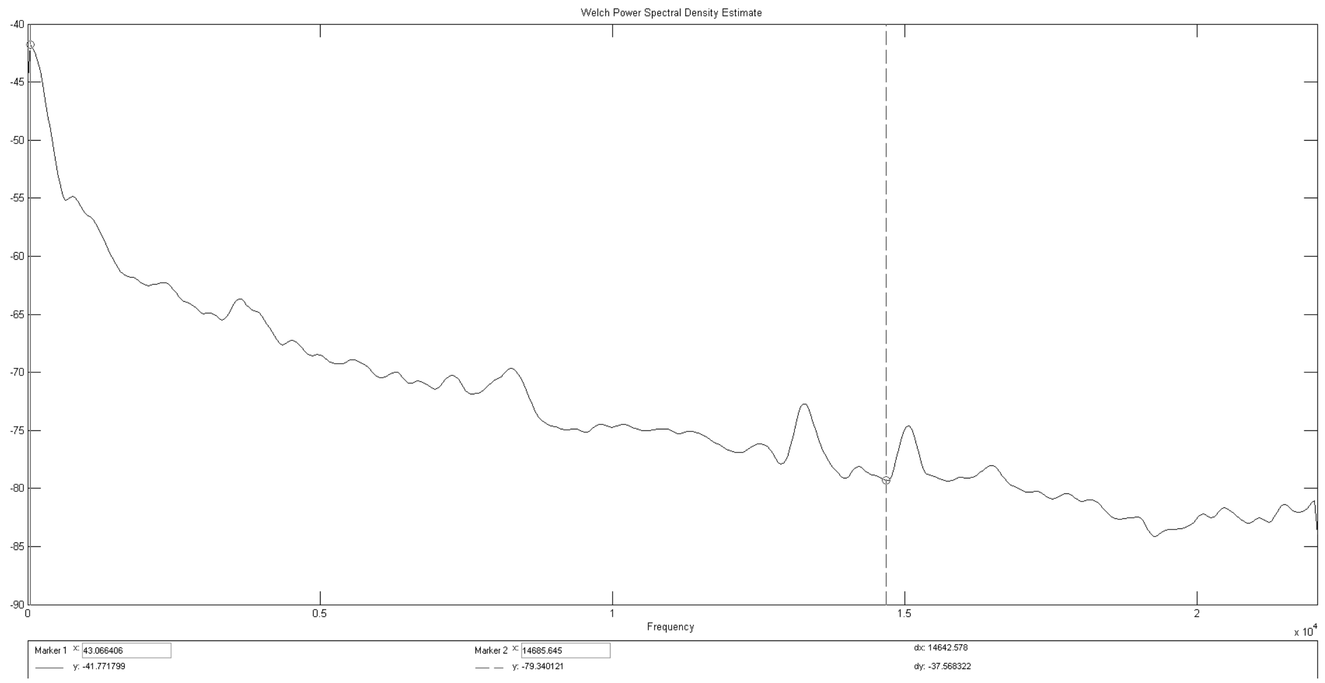


Marker 1 X: 2497.8516 y: -38.708497
Marker 2 X: 14685.645 y: -79.340134
dx: 12187.793
dy: -40.631637

Welch Spectra for noisy signal reproduced from Phase 2

In contrast, signals which lacked the 2500 Hertz wave did not produce consistent results, with the original WAV producing median dominant frequencies at 193.799 Hertz and 226.099, significantly higher than the 43.06 Hertz reported by Matlab. It is possible that Matlab is applying something other than a Fast Fourier Transform to the input signal, or using an approach for calculating the dominant frequency other than median, however the media outperforms the mean dramatically here, as the mean produced dominant frequencies of 19249.6 Hertz and 19275.3 Hertz. Since the Fast Fourier Transform is meant for periodic signals, it is likely that a different approach altogether would handle the original signals better.

Eugene Davis
CPE 381
04/21/14
Project Bonus



Welch Spectra for original signal reproduced from Phase 2

Additionally, the low-pass filtered signal was run through this program. The result here was that for the median both channels were 193.799 Hertz, and for mean both were around 18400 Hertz. This probably indicates that one channel in the original file had higher frequency components that were cut-off by the low-pass filter.

As is probably clear from the reported results, the median provided significantly better results than the mean did, even matching the Matlab produced results for the noisy WAV file.

4. Program Performance

The program yielded results in a little over 8.5 seconds for all the files, each of which were 59 seconds in length. This indicates that it is very capable of running in real-time, even on significantly slower systems.

The 8.5 second time was the run time in Linux compiled in G++ with default optimization. On a Windows XP system running in a Virtual Box virtual machine, the total execution time was almost 58 seconds, again suggesting that even on a more limited system this could be (if only just) real time.

5. Experiences and Lessons Learned

Experience

The bonus portion of this project demonstrated that an Fast Fourier Transform is very simple to apply, and when using an existing algorithm takes relatively little code to get results such as the dominant frequency from. It also demonstrated several important trade-offs which will be encountered when using an Fast Fourier Transform, which are covered in Lessons Learned.

At one point, a logic bug existed such that the windows did not overlap each other, resulting in processing time around 4 seconds. Interestingly, this did not make a significant difference in the results, perhaps because of the length of the window.

Lessons Learned

1. Median works significantly better for determining the dominant frequency of the WAV file than does mean.
2. Fast Fourier Transform has limited usefulness on signals with fewer periodic components in them (such as a recording first of voice then of music).
3. Code written for maintainability can save much time – as demonstrated by the reuse of the same basic wave_io files from Phase 1 in both Phase 2 and the Bonus.