Jeremy Palmerio
I6239656
PRA3021

Topics in Scientific Computing Week 1

Introduction

This practical is not my first encounter with matlab, fortunately, I have endured the learning curve first through the linear algebra course, although not in depth. Later on I used matlab for visualizing results in Physics Lab and further explored it in an external DACS course Mathematical Modeling (KEN2431) and lastly in the Differential Equations course (MAT2007). Over the past year I have slowly progressed and gained insight into the language of matlab and its useful applications as a vector oriented language with a plethora of built in functions. This report will focus on my use of Matlab to create a sampling function, dft function, and fft function which will be utilized to analyze Beethoven and my own voice.

Sample function

The goal of the sampling function is to output two vectors, x and t which represent the corresponding signal and time vectors. To do this, though, three inputs are required: the function we want to sample, the sample rate (Hz) and the sample duration (s). The principle is easy: create a time vector (t) of evenly spaced points from 0 to the sample duration and then apply the function to the vector t, yielding the vector x. Additionally, I have created two versions of the sampling function; one that simply returns the two vectors to be plotted, and another that also trims both vectors to be a power of 2 long. (this will be useful for Myfft function) There were no issues while creating this function.

DFT function

This function was a bit trickier to write as the math behind the dft was new to me. My goal here was to translate the math from the slides into matlab code. The goal of this function is to take a sampled signal and output matrices of values a and b which correspond to the Discrete Fourier Coefficients. To do this a loop is used to calculate each individual value of a and b for each index of the loop. I am uncertain as to what the coefficients a, b intuitively mean, but I would assume they quantify the amount that a specific frequency is present in the sample. While writing this, I encountered many errors to do with multiplication of arrays which were not the same length. Overall, individually checking the logic and the length of all the vectors finally resulted in a working function.

Jeremy Palmerio
I6239656
PRA3021

FFT function

Writing the fft function was enjoyable as I got to learn about the Cooley-Tukey-Algorithm behind which has always been of interest to me as it can dramatically reduce the computational power required for spectrum analysis compared to the DFT method by a factor of n/logn. The hardest part of this function was wrapping my head around the recursion. Even though I believed to have successfully completed the recursive function, it would not behave properly. It wasn't until the new slide was added to the pptx that I managed to fix the problem. To this point I am unaware what the problem was, but I suppose it had something to do with how I was recursively inputting the output of the previous recursion. Another issue to deal with is the fact that this algorithm only handles signals that can be split into two repeatedly , which means the length of the signal must be a power of 2 long. (This is where it's useful to have the sampling that cuts the signal at the closest, smaller power of 2.)Overall, the math behind this function is quite understandable but translating the recursion steps into matlab proved to be more difficult than anticipated. I failed to investigate the Danielson-Lanczos lemma as it was beyond the scope of this practical and I had already invested far too much time in this project than should be expected, but I will hopefully be able to explore it in a future eventuality. To quantify the success of my fft function I performed a quick tolerance test by plotting the difference of my fft and the matlab built-in fft of the same sample, which yielded a graph of the order of 10^-12 which seemed reasonable for me. For a more detailed description of how the fft works, see comments on the code.

Beethoven sample

The Beethoven sample was my favorite part of this project as it involved a lot of thinking and researching to achieve a useful goal. The idea of this code was to analyze an audio sample using my freshly built fft function and then to play around with the recording by creating a cutoff frequency after which all frequencies are removed. Then by performing the inverse fft on that spectrum, I could recover the signal devoid of any frequencies above the cutoff. There are some subtle complications that I had to deal with, notably the fact that I had to handle both single sided and double ffts. When performing the inverse fft, since I am only feeding it the single sided fft, the output signal is only half the original amplitude. I could have dealt with this problem more at the source, but I kept running into errors so I simply patched it by doubling the signal after performing the inverse fft. Furthermore, there were many issues about the different lengths of time vectors and signal vectors, which were all dealt with appropriately. Another issue which I haven't fully resolved but seems to be stable in the current state, is to do with my fft function. For some reason, at some points in writing this code the signal would not get filtered if i used my fft but it would if i simply used the built in function. I believe this has something to do with the dimensions that are outputted by my fft function (which is a row vector) and the built in function (which is a column vector). However, re-writing the code and making sure all dimensions match across the code and I obtained a cohesive code. One other interesting part was cutting off the frequency after 4000Hz. To do this, after performing the fft i created a variable which

represented the index corresponding to 4000Hz and equated all the fft matrix values after this index to 0. Then performing the inverse fft, I obtained the signal without the frequencies above 4000Hz. For more detailed information, see the code files. One of the major problems encountered during this project was the foundational confusion between single sided ffts and double sided fft and how to handle their complex parts. For example, originally I displayed the signal output by the ifft function as an absolute value, which only produced the positive part of the signal and resulted in very low quality audio. Then I decided to plot the real part of the signal instead and obtained a better signal which sounded a lot clearer.

Voice analysis

The analysis of my voice was very similar in method to the Beethoven analysis. Instead of filtering out specific frequencies though, I attempted to filter out wind using the fft. The audio of my voice has a substantial amount of windy noise which I also recorded without my voice. I then performed an fft of the signal of just the wind, in the hopes that I could subtract the wind spectrum from the voice + wind spectrum. It wasn't until I discovered that I had to display the signal as a real number rather than the absolute that I perceived some improvements. However the improvements are barely noticeable and I believe there must be a better way to go about this task. A major confusion for me, which I believe to be the root of my problem is that the spectrum of the wind signal has much higher amplitudes than the wind + voice signal. Overall I have not been able to remove the noise from the wind. However, using the plots, I can analyze what frequencies my voice operates in. Indeed there are two peaks of similar amplitude around 100Hz and 200Hz. Alternatively I tried using a Savitzky-Golay filter to reduce the noise to no avail. I suspect more knowledge about the filter is required.