ELEC4400: Filtering Using FFT and IFFT

Thomas Fuller

July 18, 2018

Introduction

When I was a graduate student, I studied how to measure the pointing direction of a camera in order to assist extraterrestrial rovers navigate on the surface of other planets. (Thats the version of the story that I tell at parties). But the real version of the story goes something like "An error of 1 degree in the pointing direction of the camera yields an error in the position estimate of hundreds of miles, and life is really sad". In that context, having a clean signal for the measurement of hte camera pointing direction is very important.

Goal

Goal today is to examine a noisy signal which I have provided on the blackboard, decompose it into component frequencies, and then filter out high frequency noise signals. We will do this through both a time domain based way, and also a frequency domain based way.

What you hand in

Hand in your code that you used, as well as

(fill it in)

Imports for this lab

```
import numpy as np

import matplotlib.pyplot as plt

from scipy.fftpack import fft, ifft

from scipy.signal import lfilter,freqz
```

FFT and IFFT

Just as in Z Transform, the FFT has an inverse function, called "inverse fast fourier transform". FFT takes a function in time $f(t)$, and returns the frequency domain $F(\omega)$. Inverse FFT takes a function $F(\omega)$, and then returns the time signal $f(t)$.

1. Define our signal by the following equation.

$$xt = \cos(2\pi 10t) + \cos(2\pi 100t) + \cos(2\pi 200t) \tag{1}$$

Make the signal 1000 data points long, and 2 seconds in duration. This is going to give you a good idea of what a noisy signal looks like in the wild.

2. Use `f,xarr = plt.subplots(3)` to set up the Plot of the noisy signal xt vs t.(refer to lab 6 for previous use of this command)

3. On the middle subplot, generate a plot of the fft amplitude spectrum. That is, a plot which has the absolute value of fftOutput of xt on the vertical axis, and the frequencies on the horizontal axis. Lab 8 has code we've previously written to do this. We should be able to clearly see spikes in the amplitude at 10Hz, 100Hz, and 200Hz.

4. To generate the lowest subplot, call `iFftOut = ifft(fftOut)`, and plot t vs iFftOut. It's very important to note that while the previous plot uses the absolute value of fftOut so we can easily see the frequency component of a signal, that the ifft function takes the actual complex numbers which are generated in fftOut in order to generate the actual inverse.

5. Refer to the figure below for the expected results. Convince yourself during this section that the FFT and IFFT are actually inverse operations that let us see the time domain or frequency domain content of a signal.

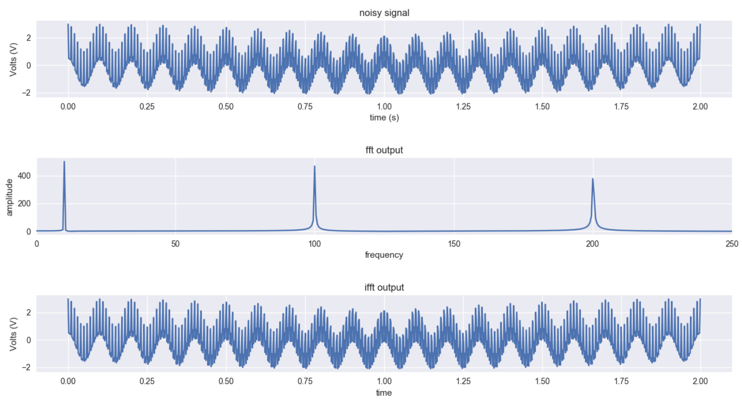

Figure 1: Expected Results of FFT and IFFT Section

Time Domain Filtering

We're going to build a moving average filter in Python. The equation below shows the difference equation for the moving average filter; it's mostly there for completeness. What needs to be known about it is that it iterates along the signal, and takes the moving average as it runs along the signal.

$$y[n] = \frac{1}{N}\sum_{n=0}^{N-1} x_n \tag{2}$$

To actually implement this we make a sequence we want to convolve with our signal in time. The below steps will discuss how that's done.

1. To make a 15 point moving average filter, we'll do `N = 15`.

2. We want to build a sequence of "1's" which is 15 elements long, and the whole sequence is divided by 15, so b = `(1/N)*np.ones(N)`.

3. To generate the filtered output, `filterOut = lfilter(b,1,xt)`. This command is convolving the sequence "b" with the data of our noisy waveform.

4. Generate a figure containing the noisy time domain signal in one subplot and the filtered output in another subplot. The expected result is shown in the figure below. **yours will not contain the impulse response that mine does, it's just to show you the two things that are being convolved together to perform the filtering operation**.

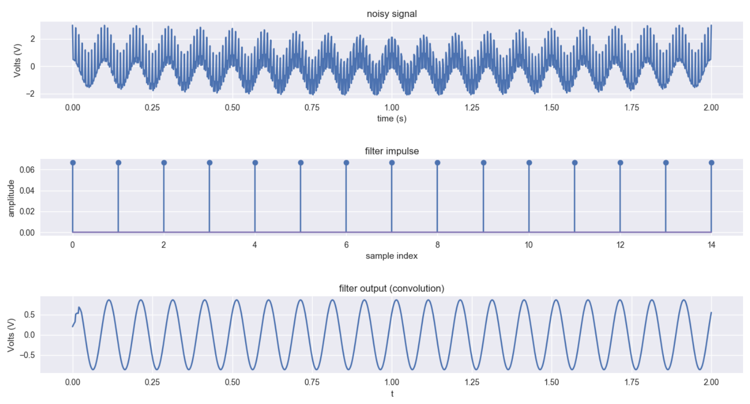

Figure 2: Expected Results from "Time Domain Filtering" Section, Except yours just has the top and bottom plots

1 Frequency Domain Filtering

A long time ago I had told you guys that convolution in the frequency domain was equivalent to multiplication in the time domain. To help prove that this is true and also perhaps to convince you of the utility of the transform methods, we'll also filter this signal in the frequency domain.

1. We need to determine the frequency response of our moving average filter. To do this, we will invoke the `freqz` command in the following way. `hz,H = freqz(b,1,worN=freqs*2*np.pi/sampleRate)`, and then do `hzTrue = hz*sampleRate/(2*np.pi)`.

2. The output H is a representation of our moving average filter called the frequency response. When we plot H vs hzTrue, we are seeing how the filter affects the output at various frequencies.

3. To perform the filtering, we multiply the frequency domain version of our signal (fftOut) by the frequency domain version of our system/filter (H). `wFilterOut = H*fftOut`.

4. Generate a figure that contains the fft amplitude spectrum of our noisy signal in the top subplot. The second subplot should contain the frequency response of the filter, `plot(hzTrue,H)`, and the third subplot will contain the result of performing the filtering in the frequency domain. `plot(freqs,wFilterOut`.

5. A plot of the expected result is shown below.

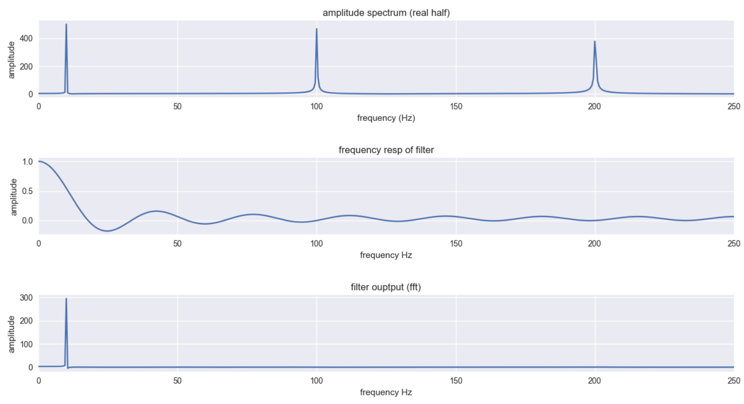

Figure 3: Expected Results from "Frequency Domain Filtering" Section, Except yours just has the top and bottom plots

The Prestige

In my humble opinion this is the most interesting part of the lab.

Final Plot Part 1

1. The final plot contains the following information. The first subplot, put the result of the time domain filtering method. The second subplot, you take the fft of that and plot the amplitude spectrum.

Final Plot Part 2

1. The third subplot of the final plot has the filtered waveform in the frequency domain in the next subplot, you take the ifft, to get the filtered signal in the time domain. Use fft and ifft to show that both approaches can filter a waveform. **The expected Result is shown below**

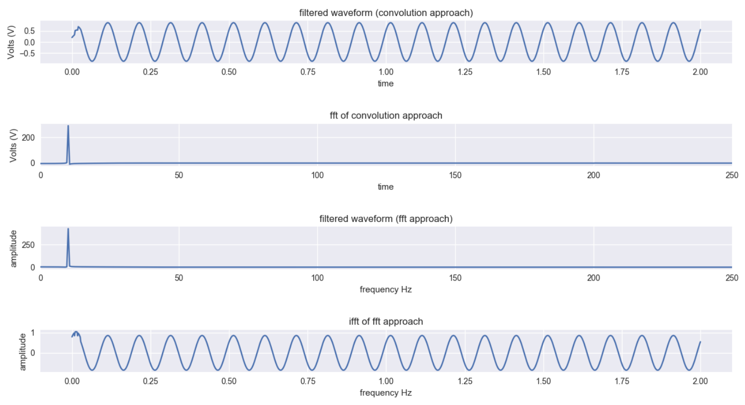

Figure 4: Final Plot expected result