CPE 367 – Experiment 3 – v6
Implementing Digital Filters in Python – Dr F DePiero 28 points

Overview and Motivation

This experiment provides an opportunity to implement and test a digital filter. You will implement the filter in Python. It should use WAV file I/O to read an input sample by sample and produce an output that is the average of current and recent inputs. Your first version of the processing will be one that is hardcoded to produce an output that is a length 3 average of the input. Your second version will be one that handles an arbitrary length filter using a circular buffer. This second version will serve as the basis for most of your programming projects in the remainder of the course. To test your signal processing system, you will process a noise signal as input. A MatLab M-File is provided that analyzes changes between the input and output noise signals to find the actual frequency response.

Learning Objectives

- Implement a digital filter that efficiently accesses the recent history of input values
- Verify the frequency response of your filter by processing noise and then using MatLab (M-file provided)
- Knowledge of the spectral properties of a pseudo-random signal: a random magnitude that has uniform values when averaged

Prerequisite Learning Objectives

- Knowledge of filtering
- Knowledge of data structures and methods known as a queue, FIFO buffer or circular buffer (terms used synonymously)
- Python programming with WAV file I/O

Background on the Use of Noise Signals to Test Digital Filters

Noise signals are valuable for testing. This is because "ideal noise" has an average power that is uniform across all frequencies. When processing the noise, gain and attenuation are introduced in the output spectrum, revealing the filter's frequency response. MatLab computes the spectrum of the input and output and takes a ratio to yield the frequency response

$$|H(f)| = \frac{|\,Y(f)\,|}{|\,X(f)|}$$

Where the spectra Y(f) and X(f) are associated with the output and input, respectively. The reason averaging is needed is because a short noise sequence will have a random spectrum. However, the longer-term average of a noise spectrum is more accurate. It is more uniform, as shown in the figure below. MatLab uses the "Welch Periodogram" technique. (Defined by Schuster in 1898 as part of a study of meteorological phenomena. Hence, the non-EE/CPE terminology!)

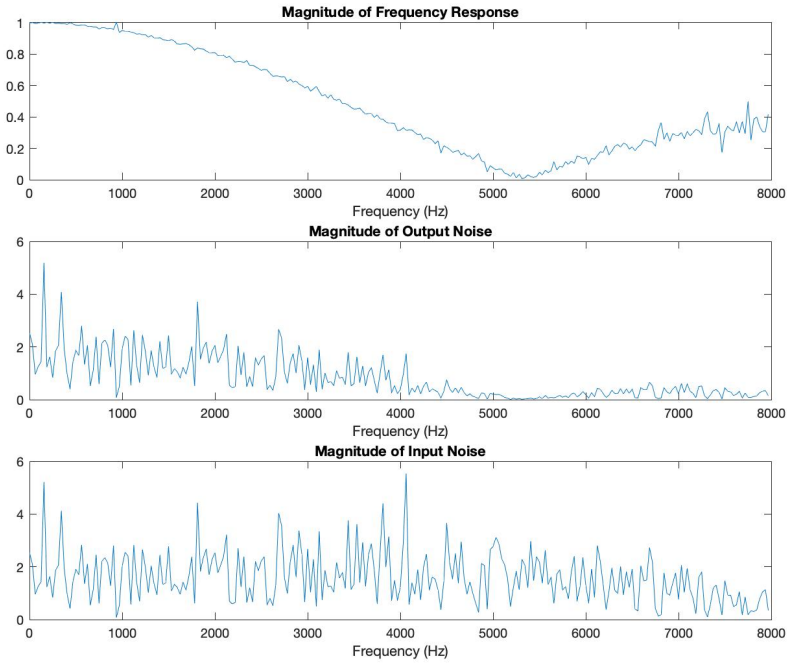

Figure 1. Frequency response of a length 3 filter, using a short noise signal. The limited duration of the noise signal results in a frequency response plot that lacks accuracy.

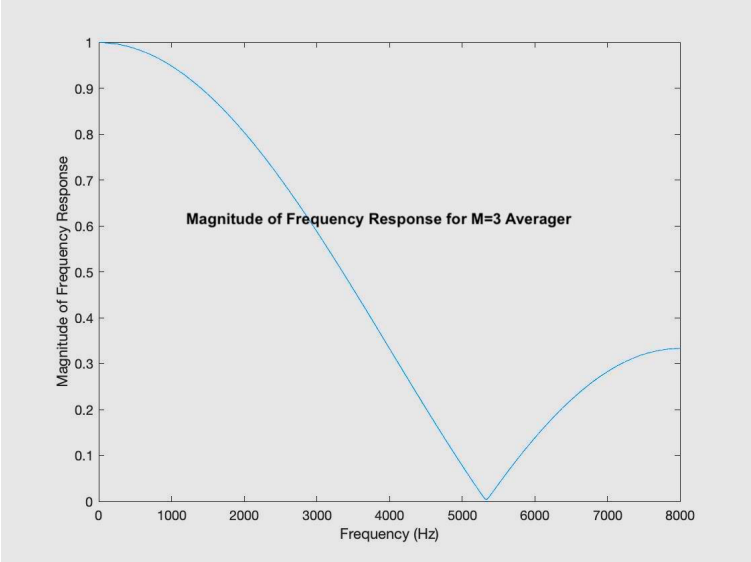

Figure 2. Frequency response of a length 3 filter, found by averaging a longer noise signal, for improved accuracy. Note the smoother curve, compared to Figure 1.

Use the M-File **fresp_system_welch.m** to compute and display the frequency response of your system. It takes two arguments that specify WAV files.

>> fresp_system_welch in_noise.wav out_noise.wav

Procedures, Questions and Deliverables

1) Implement a Length-Three Digital Filter ***Use scalar variables in this part, no FIFO yet***

Implement a digital filter that computes an output, y[n], by averaging the current and the two most recent inputs. Use scalar variables in order to implement this first version of your filter (as opposed to a FIFO). Use the in_noise.wav as an input. It has a 16kHz sample rate.

Comparing your length 3 averager to a generic FIR difference equation, we see that in this case the bk filter coefficients are all identical. Note that this is not true with most filters, but is the case for the average. Your implementation needs to accommodate arbitrary bk values.

$$y[n] = \sum_{k=0}^{M-1} b_k \, x[n-k]$$

1a1) What are your bk coefficient values? (2)
1a2) Deliverable: Python code, with comments (4)

Use the fresp_system_welch M-File to test your system. Its frequency response should be quite similar to Figure 2. Listen to the input and output WAV files to hear the difference, which should sound low pass in nature.

1b) Deliverable: Frequency response of your filter, as computed by MatLab (2)

Negate the middle bk coefficient and re-run the MatLab analysis.

1c1) Deliverable: Frequency response of your modified filter here, as computed by MatLab (2)
1c2) Describe the type of your modified filter (HaHa! Wow!) (2)

2) Implement a Length-M Digital Filter ***Implement a FIFO in this part***

Evaluating a difference equation in Python is readily accomplished via a for loop. However, accessing the recent history of the input x[n] in an efficient manner necessitates the use of a FIFO or circular buffer. This is a common implementation in higher level software programs and also at the assembly language level – on DSP processors.

Implement your length-M filter in Python. Assign bk = 1/M. Create a method that accesses the history of the input for a specific value of k, as with fifo.get(k). See the my_fifo.py file for suggestions. Also see the cpe367_filter.py file. It provides an example of WAV I/O, accessing the input and processing in a sample-by-sample fashion. When instantiated, you can fill your FIFO with zeros. The FIFO will remain full during operation. (A new sample is saved and the oldest overwritten with every update.) This simplifies implementation.

FYI, Python has a built-in queue class. The built-in classes may not be used for your fifo! Sorry –our goal is to maximize learning! Also, the above fifo.get(k) method is only a few lines long.

2a) Deliverable: Describe your method for implementing the FIFO. Do you increment or decrement the buffer index when you make space for the most recent input? Do you increment or decrement the buffer index when you access a past value in the fifo? (2)

Verify the operation of your filter by processing noise signals and using MatLab, as done previously with the M = 3 case. The filter should be lowpass and you should see zeros in the frequency response at f = fs/M, where fs is the sample rate. Store the bk filter coefficients in a list to facilitate the evaluation of your filter. **Try testing with M = 3, 11, 51.** The M = 3 case should yield results in Matlab that are identical to your prior implementation.

2b) Deliverable: Frequency response of your length-11 filter, as computed by MatLab (2)

Interpret your frequency response plot. The cutoff is defined as the frequency at which the magnitude of the frequency response is at $\frac{H_{PEAK}}{\sqrt{2}}$, where H_{PEAK} is the peak response of the filter. This should be exactly 1.0 in the case of the length 3 averager, for example. The "first zero" of a filter is the frequency at which the magnitude of the frequency response attains a value of exactly zero. The notion of the "first" is associated with the first instance of a zero output, moving out of the passband. Hence for a lowpass filter this is the first zero encountered when starting at DC and then increasing frequency. For the M=11 filter -

- **2c1) What is the actual cutoff for your filter? in Hz (2)**
- **2c2) What is the actual first zero for your filter? in Hz (2)**

- **2d) Deliverable: Your version of the my_fifo.py class, with comments (8).**

Note that the frequency domain plots presented by MatLab are generated computationally. This results in a finite spacing between samples in the frequency domain. Hence, a sample may not fall exactly at a zero. However, you can still estimate the zero locations.

FYI - Unanswered Questions Necessitating Further Techniques for Analysis and Design

- How can we design a filter to meet specified requirements such as cutoff?
  - Answer involves the Inverse-DTFT transform
- How can we design a filter to reduce stop band ripple?
  - Answer involves the Inverse-DTFT transform and convolution
- What is the theoretical reason explaining why particular changes to the bk coefficients alter the frequency response? How can these changes be designed into a filter to meet specified requirements? Answer involves properties of the DTFT transform
- How can we predict when or if zeros occur in a frequency response?
  - Answer involves the DTFT and the Z transform
- How can a spectrum be computed computationally?
  - Answer involves the DFT