EE 367 Lab 3

Implementing Digital Filters in Python

Report by:
Ethan Vosburg (evosburg@calpoly.edu)
Isaac Lake (ilake@calpoly.edu)

February 13, 2024

Table of Contents

1 Implement a Length-Three Digital Filter

In part 1 of this lab we will be brute forcing a filter to work by writing a unique computation for each filter, this is to give us a better understanding of how a filter might work as well as what we should expect from a filter.

1. What are your bk coefficient values?

   For a system that averages 3 numbers we will use bk values of 1/3 and add the three most recent inputs together to get a simplified average.
2. Python code, with comments:

Listing 1: Python Digital Filter Coefficients Generator

```
################################################################
# students - allocate your fifo, with an appropriate length (M)

# This is an averaging filter that adds a 1/M (M being the size of the
# buffer we are filtering) to the bk_list

bk_list = [1/3, -1/3, 1/3]
xprev = 0
xprev2 = 0

################################################################
```

Listing 2: Python Digital Filter Processing

```
   ################################################################
   # students - go to work!

   # evaluate your difference equation
   yout = 0
   for k in range(M):

      # use your fifo to access recent inputs when evaluating your diff eq
      # y[n] = sum of b[k] * x[n-k]
      yout += bk_list[k] * fifo.get(k)

   # students - well done!
   ################################################################
```

3. Frequency response of your filter, as computed by MatLab:

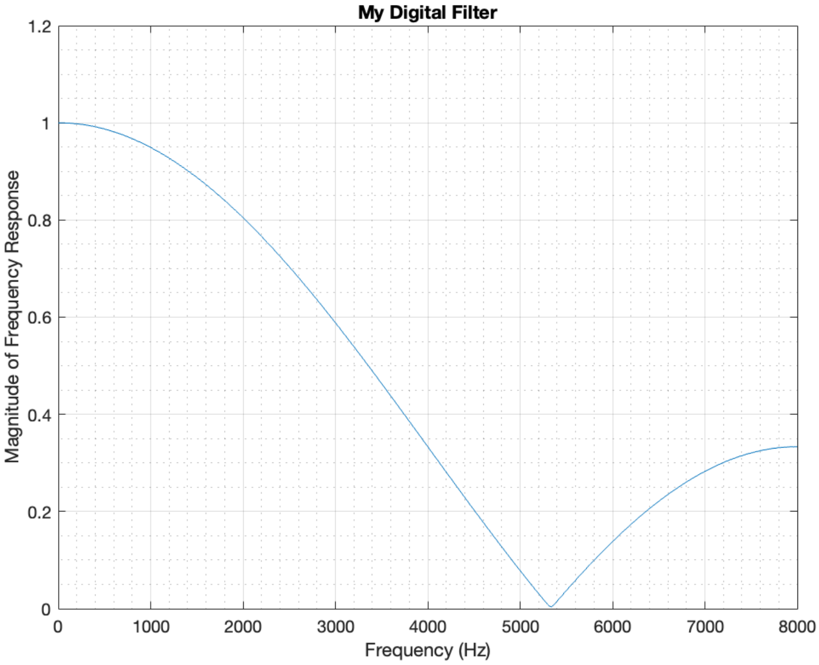

Figure 1: Averaging Filter 1

4. Now negate the middle bk coefficient and provide the frequency response as computed by MatLab:

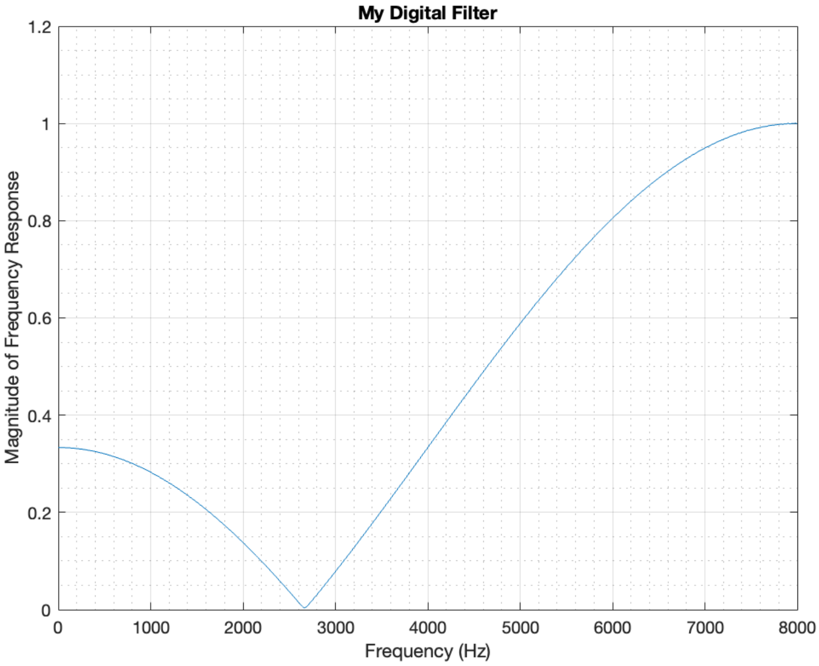

Figure 2: Averaging Filter 2

5. Describe the type of modified filter:

   While the original filter looked like a low-pass filter that only passed low frequencies, this minor change in the bk coefficient seemed to reverse it into a high-pass filter.

2 Implement an M Length Digital Filter

In part 2 of this lab we will implement a data structure that will be able to hold and implement a filter(ie a set of bk values) of variable length to work with nearly any filter we can dream of

1. 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?

   Our method of implementation was quite simple. For the getter method we simply had it return the number in the buffer at the index we were given with a single line of code. The updater method was also simple since it was already organized by first in first out all we had to do was add an extra piece of data on the end and pop the one at the front. Most of our troubles came from a lack of immediate code comprehension that comes whenever you get new code that someone else wrote.
2. Frequency response of your length-11 filter, as computed by MatLab:

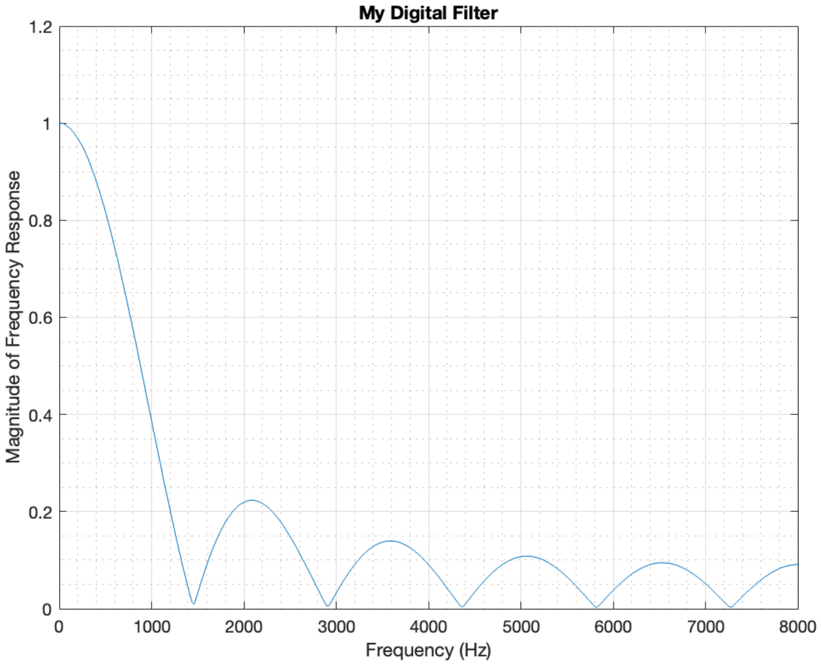

Figure 3: 11 Point Averaging Filter

3. What is the actual cutoff for your filter? in Hz

   The actual cutoff frequency is $646Hz$.

4. What is the actual first zero for your filter? in Hz

   The actual first zero is $1454Hz$.

5. Your version of the my_fifo.py class, with comments

Listing 3: Python Fifo Class

```
#!/usr/bin/env python

############################################
# this EMPTY python fifo class was written by dr fred depiero at cal poly
# distribution is unrestricted provided it is without charge and includes attribution

import sys
import json


class my_fifo:

    ############################################
    # constructor for signal history object
    def __init__(self, buff_len):

        self.buff_len = buff_len
        self.buff = []
        for k in range(buff_len):
            self.buff.append(0)

        # initialize more stuff, as needed

    ############################################
    # update history with newest input and advance head / tail
    def update(self, current_in):
        """
        :current_in: a new input value to add to recent history
        :return: T/F with any error message
        """

        # Insert the current input at the beginning
        self.buff.insert(0, current_in)

        # Pop of the last element to preserve the size of the array
        self.buff.pop()

        return True

    ############################################
    # get value from the recent history, specified by age_indx
    def get(self, age_indx):
        """
        :indx: an index in the history
                age_indx == 0    ->  most recent historical value
                age_indx == 1    ->  next most recent historical value
                age_indx == M-1  ->  oldest historical value
        :return: value stored in the list of historical values, as requested by indx
        """

        # return the value at the desired index
        return self.buff[age_indx]
```

Listing 4: Python Digital Filter Coefficients Generator

```
###############################################################
# students - allocate your fifo, with an appropriate length (M)

M = 11
fifo = my_fifo(M)
previous = list

# students - allocate filter coefficients, length (M)
# students - these are not the correct filter coefficients
bk_list = []
for i in range(M):
   bk_list.append(1/M)
xprev = 0
xprev2 = 0

###############################################################
```

Listing 5: Python Digital Filter Processing

```
   ###############################################################
   # students - go to work!


   # update history with most recent input
   fifo.update(xin)

   # evaluate your difference equation
   yout = 0
   for k in range(M):

      # use your fifo to access recent inputs when evaluating your diff eq
      # y[n] = sum of b[k] * x[n-k]
      yout += bk_list[k] * fifo.get(k)

   # evaluate difference equ, here as y[n] = x[n]
   # yout = xin

   # update history of recent inputs...
   # xprev = ...
   #temp = xprev
   #xprev = xin
   #xprev2 = temp







   # students - well done!
   ###############################################################
```