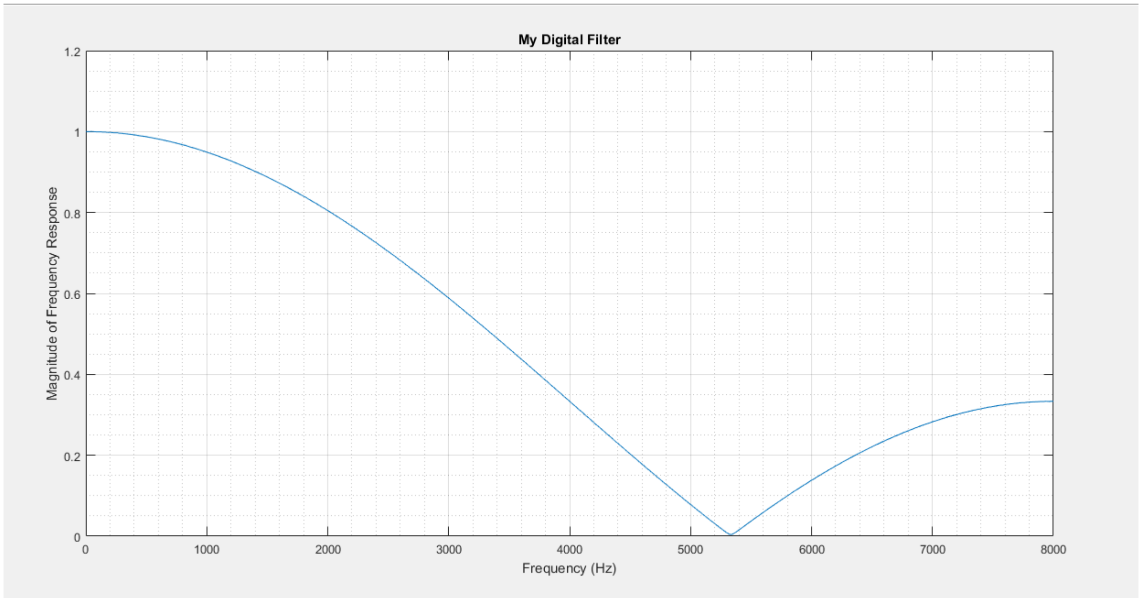
1a1) What are your bk coefficient values? (2)

1/3

1a2) Deliverable: Python code, with comments (4)

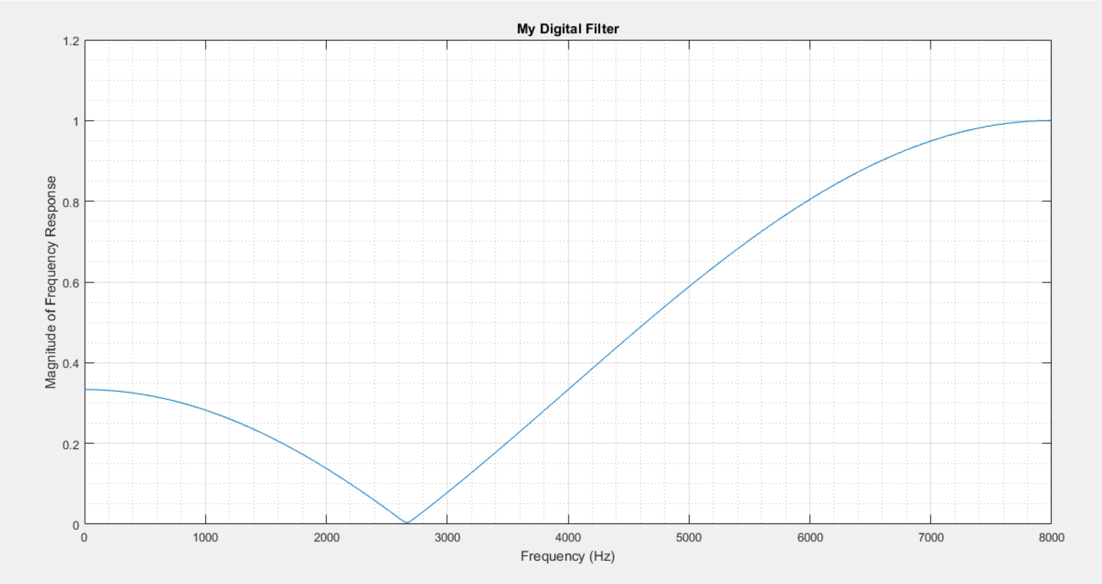
```
#!/usr/bin/python

import sys
import time

import base64
import random as random

import datetime
import time

from cpe367_wav import cpe367_wav
from my_fifo import my_fifo

############################################
############################################
# define routine for implementing a digital filter
def process_wav(fpath_wav_in,fpath_wav_out):
    """
    : this example implements a very useful system:  y[n] = x[n]
    : input and output is accomplished via WAV files
    : return: True or False
    """

    # construct objects for reading/writing WAV files
    #  assign each object a name, to facilitate status and error reporting
    wav_in = cpe367_wav('wav_in',fpath_wav_in)
    wav_out = cpe367_wav('wav_out',fpath_wav_out)

    # open wave input file
    ostat = wav_in.open_wav_in()
    if ostat == False:
        print('Cant open wav file for reading')
        return False

    # setup configuration for output WAV
    # num_channels = 2
    # sample_width_8_16_bits = 16
    # sample_rate_hz = 16000
    #
wav_out.set_wav_out_configuration(num_channels,sample_width_8_16_bits,sample_rate_hz)

    # configure wave output file, mimicking parameters of input wave (sample rate...)
    wav_out.copy_wav_out_configuration(wav_in)

    # open WAV output file
    ostat = wav_out.open_wav_out()
    if ostat == False:
        print('Cant open wav file for writing')
        return False

    # students - allocate your fifo, with an appropriate length (M)
    M = 11
    fifo = my_fifo(M)

    # students - allocate filter coefficients, length (M)
    # students - these are not the correct filter coefficients
    bk_list = 1/3#[1/3, 1/3, 1/3]

    # process entire input signal
    xin = 0
    while xin != None:

        # read next sample (assumes mono WAV file)
        #  returns None when file is exhausted
        xin = wav_in.read_wav()
        if xin == None: break

        #################################################################
        #################################################################
        # 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] = b[k] * x[n-k]
            yout += bk_list * fifo.get(k)
        # students - well done!
        #################################################################
        #################################################################

        # convert to signed int
        yout = int(round(yout))

        # output current sample
        ostat = wav_out.write_wav(yout)
        if ostat == False: break

    # close input and output files
    #  important to close output file - header is updated (with proper file size)
    wav_in.close_wav()
    wav_out.close_wav()

    return True


#############################################
#############################################
# define main program
def main():

    # check python version!
    major_version = int(sys.version[0])
    if major_version < 3:
        print('Sorry! must be run using python3.')
        print('Current version: ')
        print(sys.version)
        return False

    # grab file names
    fpath_wav_in = 'in_noise.wav'
    fpath_wav_out = 'out_noise.wav'

    #############################################
    #############################################
    # test signal history
    #  feel free to comment this out, after verifying

    # allocate history
    M = 3
    fifo = my_fifo(M)
    # add some values to history
    fifo.update(1)
    fifo.update(2)
    fifo.update(3)
    fifo.update(4)

    # print out history in order from most recent to oldest
    print('signal history - test')
    for k in range(M):
        print('hist['+str(k)+']='+str(fifo.get(k)))

    ###############################################
    ###############################################

    # let's do it!
    return process_wav(fpath_wav_in,fpath_wav_out)


###############################################
###############################################
# call main function
if __name__ == '__main__':

    main()
    quit()
```

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)



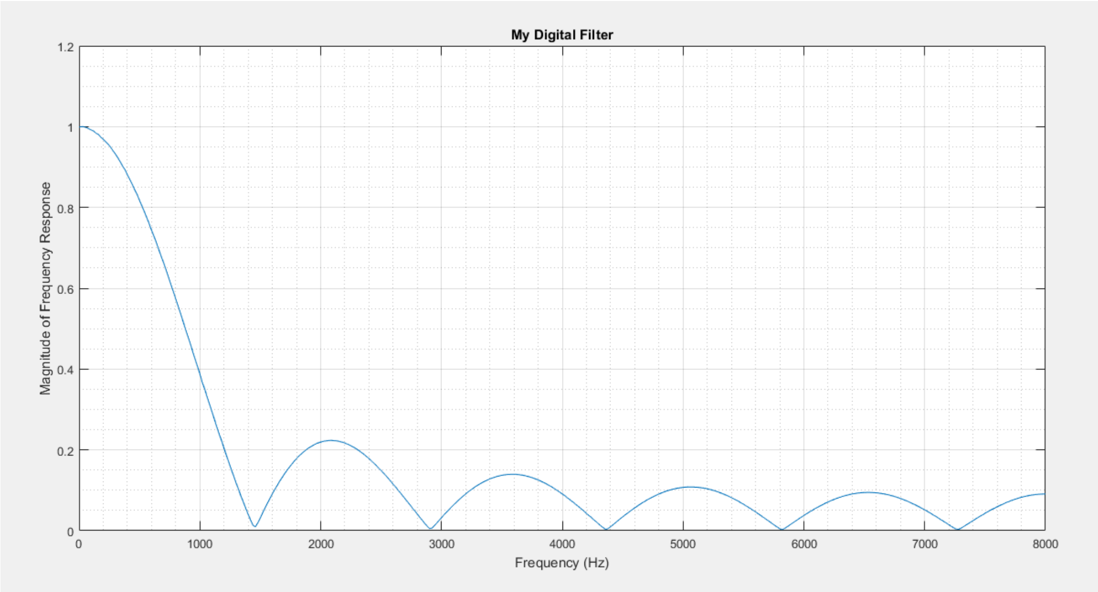
1c2) Describe the type of your modified filter (HaHa! Wow!) (2)

The modified filter is Haha.

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)

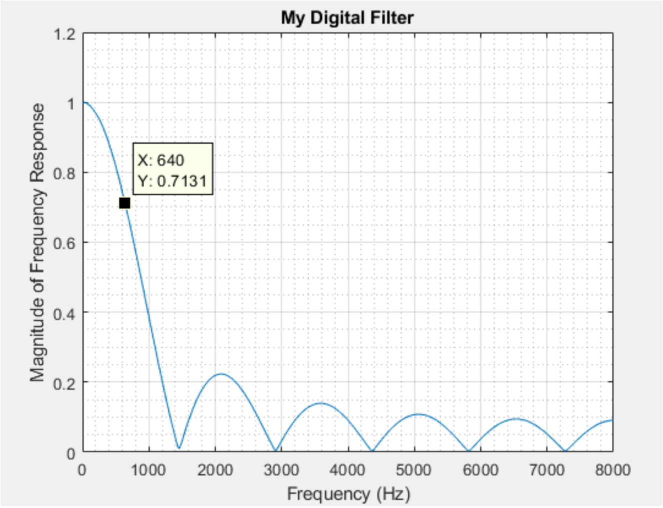
The implementation of FIFO is just like a circular list and the buffer index is maintained whenever FIFO is updated. The buffer index is increased whenever the most recent input is added in the buffer, if the buffer index increases the length of the buffer itself, it is reinitialized to 0. While the buffer index is decreased whenever a past value is accessed from the buffer, the index if goes in negative is added to the total length of the buffer itself.

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



2c1) What is the actual cutoff for your filter? in Hz (2)

From the following figure, the actual cutoff frequency of the filter is 640 Hz.



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

The actual first zero of the filter is calculated as:

$$z_l = j\frac{1}{\cos\left[(2l-1)\frac{\pi}{2N}\right]}$$

Where N is filter order determined from matlab, that is, 400. Therefore, first zero is at;

$$z_1 = j\frac{1}{\cos\left[(2-1)\frac{\pi}{2\times 400}\right]} = j$$

$$
\begin{aligned}
s - j &= 0 \\
s &= j \\
j2\pi f &= j \\
f &= \frac{1}{2\pi} Hz
\end{aligned}
$$

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

```
#!/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
		# buffer index is maintained for accessing
		# history and updating recent values
		# initialized with -1 so that when upadation
		# increments by 1 it starts first value at 0 index
		self.buff_index = -1
		self.buff = []
		for k in range(buff_len): self.buff.append(0)

		# initialize more stuff, if 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
		"""
		# updation requires increment of index by 1
```

```
    self.buff_index = self.buff_index + 1
    # if index exceeds the buffer length, it is re-initialized to 0
    if self.buff_index == self.buff_len:
        self.buff_index = 0
    self.buff[self.buff_index] = current_in

    return True



############################################
# get value in history at a given age, specified by age_indx
#  age_indx == 0  ->  most recent
#  age_indx == 1  ->
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
    """
    # the higher the age_indx the older the values are
    index = self.buff_index - age_indx
    # index is wraped around in case index decrements below 0
    if index < 0:
        index = self.buff_len + index

    val = self.buff[index]

    return val
```