CSE 3313 – Homework #8

Assigned: Wednesday, April 10, 2019
Due: Wednesday, April 17, 2019 at the end of class

Note the following about the homework:

1. You must show your work to receive credit.
2. For the hand-written problems, submit a hard-copy.
3. For the problems requiring Python, upload your source code to Blackboard. See instructions at the end of the document.

Assignment:

Applications

1. (50 points) **Application Area: Notch Filter**
   Purpose: Learn to apply a notch filter to a signal to eliminate a specific frequency. Also gain experience with z-transforms.

   Sometimes we may wish to attenuate a range of frequencies. For this, a `bandstop` filter can be applied. When the range of frequencies is very narrow, we call this a `notch` filter [IJ02]. A common application would be to remove a 60 Hz component that is induced by electrical equipment. Another example can be found on page 4 of `https://spqr.eecs.umich.edu/papers/YanFuXu-Cuba-CSE-TR-001-18.pdf`, in which an occupancy sensor's ultrasonic sound is interfering with experiments.

   (a) (do by hand) Our notch filter has the transfer function

   $$H(z) = \frac{(1 - e^{j\hat{\omega}}z^{-1})(1 - e^{-j\hat{\omega}}z^{-1})}{(1 - 0.9372e^{j\hat{\omega}}z^{-1})(1 - 0.9372e^{-j\hat{\omega}}z^{-1})} = \frac{Y(z)}{X(z)}$$

   where $\hat{\omega}$ is the normalized radian frequency. Use the inverse z-transform to produce the output difference equation in terms of $\hat{\omega}$; you will fill in the value for $\hat{\omega}$ below.

   Do this by hand and keep the value produced in the denominator to 4 decimal places.

   (b) The course website has a skeleton file called `notchFilter.py` that you should complete. It calls a function named `applyNotch()`, which has the signature

   ```
   applyNotch(fs, dataFileName)
   ```

   You will create `applyNotch()`. While it's OK to decompose your code into multiple functions, I should be able to call `applyNotch()` with these parameters and have everything work.

   Assume that the data file whose name is passed as `dataFileName` is in the current directory.

(c) Using the difference equation for the transfer function given above, filter a 17 Hz signal from the data. To produce the normalized radian frequency, $\hat{\omega}$, use $f = 17$ Hz and $f_S =$ 500 samples/second.

(d) You will need to evaluate `y[n]` for each of the input signal values. That is, use the difference equation that you produced above. Don't use any of the NumPy or SciPy libraries that do the filtering for you.

(e) Assume that `y[n] = 0` when $n < 0$. Assume that `x[n] = 0` when $n < 0$ or $N - 1 < n$ where N is the number of elements in `x[n]`.

(f) Because we are using an IIR filter, in theory we could keep producing values forever. If the number of input values is N, produce $N + 100$ output values.

(g) Create 3 plots, each in its own figure window (don't use subplots).

   i. the original signal `x` with the **x-axis** limited to [-25, 625]. The y-axis should be whatever it defaults to.

   ii. the filtered signal `y` with the **y-axis** limited to [-2.25, 2.25]. The x-axis should be whatever it defaults to.

   iii. a signal produced by combining a 10 Hz signal and a 33 Hz signal with the **x-axis** limited to [-25, 625]. The y-axis should be whatever it defaults to.

   The limits on the plots will make it easier to compare them (pay attention to which axis you are limiting in each case). Note that the filtered signal will be a phase-shifted version of the 10 Hz + 33 Hz signal (this is a characteristic of IIR filters).

(h) Include titles on your plots indicating what is being plotted

2. (50 points) **Application Area: Shelving Filter to Change Gain of Low Frequencies**
Purpose: Learn to amplify or attenuate a particular range of frequencies.

Radios often have dials that allow the bass (low) or treble (high) frequencies to be changed as a group. These can be implemented by a type of IIR filter called a *shelving filter*.

(a) The course website has a skeleton file called `shelvingFilter.py` that you should complete. It calls a function named `applyShelvingFilter()`, which has the signature

```
applyShelvingFilter(inName, outName, gain, cutoff)
```

You will create `applyShelvingFilter()`. While it's OK to decompose your code into multiple functions, I should be able to call `applyShelvingFilter()` with these parameters and have everything work. These parameters are:

- `inName`: a string representing the name of an audio file in the WAVE format to read and process
- `outName`: a string representing the name of an audio file that you will write in the WAVE format after applying the filter
- `gain`: an integer that represents the gain g of the filter. This is how much to increase or decrease the relevant frequencies and therefore could be negative.
- `cutoff`: an integer that represents the cut-off frequency f_c of the filter. The modified magnitudes will be for frequencies in the range of 0 Hz to the cut-off frequency.

Assume that the audio file to read (`inName`) and the audio file to write to (`outName`) will both be in the current directory.

(b) On Blackboard is a file, `P_9_1.wav`, that you can use; it's from [SB96]. You don't necessarily have to use this file, but your program should read a file in the WAVE format using the sound library we have used for previous assignments.

(c) Also on Blackboard is a copy of the relevant chapter of [LDKN00], which describes the process that your shelving filter will apply.

(d) Your program should also do the following:

  i. Plot the following side-by-side using subplots:

    A. the magnitudes of the fft of the original signal with the x-axis in Hertz. This x-axis labels should be their actual frequencies.

    B. the magnitudes of the fft of the filtered signal with the x-axis in Hertz. This x-axis labels should be their actual frequencies.

  ii. If the fft of the original signal has N values, plot only the first $N/4$ values of both the original and filtered signals.

  iii. The y-axis of both plots should be from 0 to the maximum magnitude of the two signals, plus 100. For example, if the maximum magnitude for the original signal is 2000 and the maximum magnitude for the filtered signal is 1500, the y-axis for BOTH plots should be from 0 to 2100.

(e) Save the processed signal as `outName` using the sound library we have been using in our assignments.

(f) On your own play around with positive and negative values for the gain and different cut-off frequencies to see how they affect the sound. Also listen to the original signal to see what is happening.

General requirements about the Python problems:

a) As a comment in your source code, include your name.

b) The Python program should do the work. Don't perform the calculations and then hard-code the values in the code or look at the data and hard-code to this data unless instructed to do so.

c) The program should not prompt the user for values, read from files unless instructed to do so, or print things not specified to be printed in the requirements.

To submit the Python portion, do the following:

a) Create a directory using your net ID in lowercase characters plus the specific homework. For example, if your net ID is `abc1234` and the homework is `hw04`, then the directory should be named `abc1234-hw04` (zero-pad the number if necessary to make it two digits).

b) Place your .py files in this directory.

c) Do not submit the data files unless instructed to do so.

d) Zip the directory, not just the files within the directory. You must use the zip format and the name of the file (using the example above) will be `abc1234-hw04.zip`.

e) Upload the zip'd file to Blackboard.

References

[IJ02] Emmanuel C. Ifeachor and Barrie W. Jarvis. *Digital Signal Processing: A Practical Approach*. Prentice Hall, Upper Saddle River, New Jersey, 2^{nd} edition, 2002.

[LDKN00] John Lane, Jayant Datta, Brent Karley, and Jay Norwood. *DSP Cookbook*. Prompt Publications, 2000.

[SB96] Virginia Stonick and Kevin Bradley. *Labs for Signals and Systems Using MATLAB*. PWS Publishing Company, 1996.