CSE 3313 – Homework #7

Assigned: Monday, April 6, 2020
Due: Monday, April 13, 2020 at the end of class

Assignment:

1. (will not be graded) One of the properties of the DFT is *linearity*. It says that given time-domain signals $x_1[n]$ and $x_2[n]$ with spectra $X_1[k]$ and $X_2[k]$, respectively, then

$$ax_1[n] + bx_2[n] \leftrightarrow aX_1[k] + bX_2[k]$$

While I do want you to submit **hand-written** answers, it's OK to use Numpy to calculate the values. For each step, write the results.

   (a) Find the DFT of $x_1[n] = \{1, 2, 1, 2\}$ and $x_2[n] = \{3, 0, -1, -4\}$. These are $X_1[k]$ and $X_2[k]$.

   (b) Produce $x[n] = x_1[n] + x_2[n]$ and find the DFT of $x[n]$, which we will call $X[k]$.

   (c) Produce $X_1[k] + X_2[k]$ and compare it to $X[k]$.

2. (will not be graded) Another property of the DFT is that *time-domain convolution is equivalent to frequency domain multiplication*. We will use the $x_1[n]$ and $x_2[n]$ from the previous problem.

While I do want you to submit **hand-written** answers, it's OK to use Numpy to calculate the values. For each step, write the results. For those values that are not integers, write them to 2 decimal places.

   (a) Convolve $x_1[n]$ with $x_2[n]$.

   (b) Produce the element-wise product of $X_1[k]$ and $X_2[k]$. That is, produce $X[k] = \{X_1[0] * X_2[0], X_1[1] * X_2[1], \ldots\}$.

   (c) Find the inverse DFT of $X[k]$ from the previous step. *How does this compare to the results of the convolution?*

   (d) Add enough zeros to the end of signals $x_1[n]$ and $x_2[n]$ so that the length of each is the same as the result of the convolution. Let's call these zero-padded signals $xz_1[n]$ and $xz_2[n]$.

   (e) Find the DFT of $xz_1[n]$ and $xz_2[n]$, which will be called $XZ_1[k]$ and $XZ_2[k]$, respectively.

   (f) Produce the element-wise product of $XZ_1[k]$ and $XZ_2[k]$, which we will call $XZ[k]$.

   (g) Find the inverse DFT of $XZ[k]$. *How does this compare to the results of the convolution?*

Programming Applications

3. (100 points) **Application Area: DTMF Decoding**
Purpose: Learn how to produce a filter bank to detect which of several frequencies are present in a signal. Also learn to apply bandpass filters.

This assignment is based on Lab 7 (page 469) in Appendix C of [MSY98].

Touch tone phones generate dual-tone multifrequency (DTMF) signals. The numbers of a standard keypad can be seen in table 1. When a button on the keypad is pressed, a signal is sent that is the combination of the row frequency and column frequency. For example, when you press '8' a signal composed of 852 Hz and 1336 Hz frequencies is produced.

frequencies	1209 Hz	1336 Hz	1477 Hz
697 Hz	1	2	3
770 Hz	4	5	6
852 Hz	7	8	9
941 Hz	*	0	#

Table 1: DTMF frequencies

To determine which button is pressed, a **filter bank** is used. A filter bank is a set of bandpass filters, with one for each of the frequencies that you wish to detect. The idea is that the filter for frequency f_k passes the f_k component relatively unchanged while the other frequencies are attenuated. Checking the output of each filter allows us to determine which of these frequencies are present in the signal.

(a) Your source code will be named `dtmf.py`. You can find a skeleton file on the course website that shows the function to implement. Your program must receive and return the appropriate values in the correct order.

(b) On the course website is the file `tones.zip`, which contains the tones (as a set of numbers in CSV format) for several sets of data. The buttons that correspond to the frequencies present are in the filename, for example, `tones-7481414.csv`. Your function will receive the name of one of these files to process.

(c) The output will be the phone number (including the asterisk and pound sign) that the tones represent. For example, in this example your program would print

```
7481414
```

without spaces or any other special characters.

(d) The sampling rate is $f_s = 8000$ Hz. However, each tone is only generated for half a second and therefore represents 4000 samples of the overall signal. The length of the signal will be a multiple of 4000.

(e) The filter length is $L = 64$. The filter coefficients are produced using

$$h[n] = \frac{2}{L} \cos\left(\frac{2\pi f_b n}{f_s}\right), \qquad 0 \le n < L$$

where f_b is the frequency that the bandpass is designed to pass and f_s is the sampling frequency.

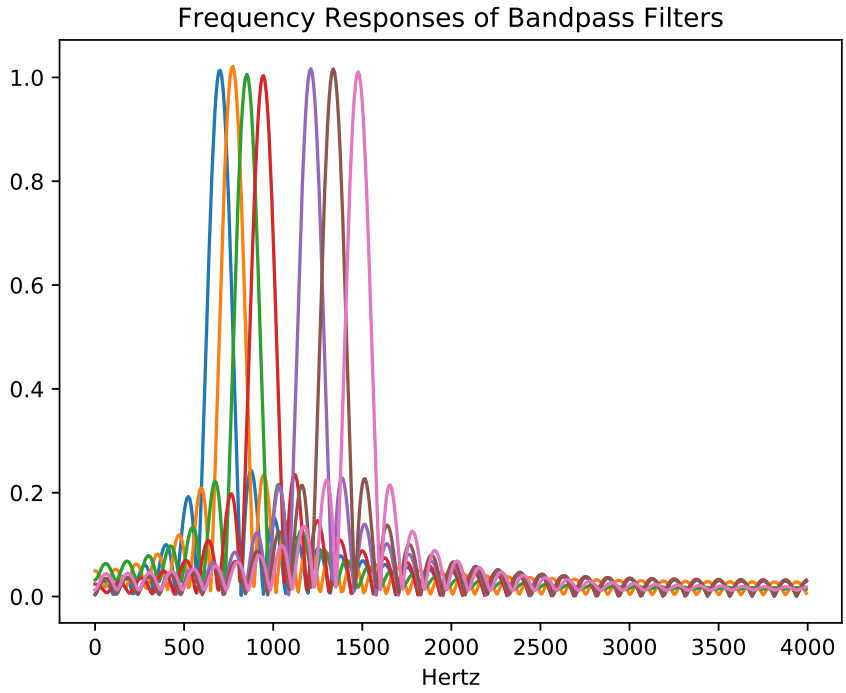

Figure 1: Frequency responses of bandpass filters for detecting touch tones

(f) To determine the frequencies present in a particular tone, determine which of the 7 filters have the highest values for

$$np.mean(y * *2)$$

where y is the output of the filter. For example, if the first 4000 values have a tone composed of a 697 Hz signal and a 1477 Hz signal, then the bandpass filters for those two frequencies should produce output signals with much higher mean values than the other filters. Do not just look at the data and determine a cut-off value. I should be able to scale the values and have your program still work.

(g) Don't hard-code your logic to this particular data.

Your program should produce the following output:

(a) The phone number corresponding to the tones, e.g., `12*34##56`. This will be printed in "main" by returning the phone number as a string.

(b) A plot in which you plot the frequency response for the filter coefficients of the bandpass filters for detecting each of the seven frequencies used to produce the touch tones. This will be a single plot with all of the frequency responses. When doing this, make the x-axis show the correct frequencies in Hertz. Figure 1 shows an example. The colors shown were the default values.

(c) A spectrogram of the signal. To get the values to plot, use the following:

```
from scipy.signal import spectrogram

f, t, Sxx = spectrogram(x, fs)
```

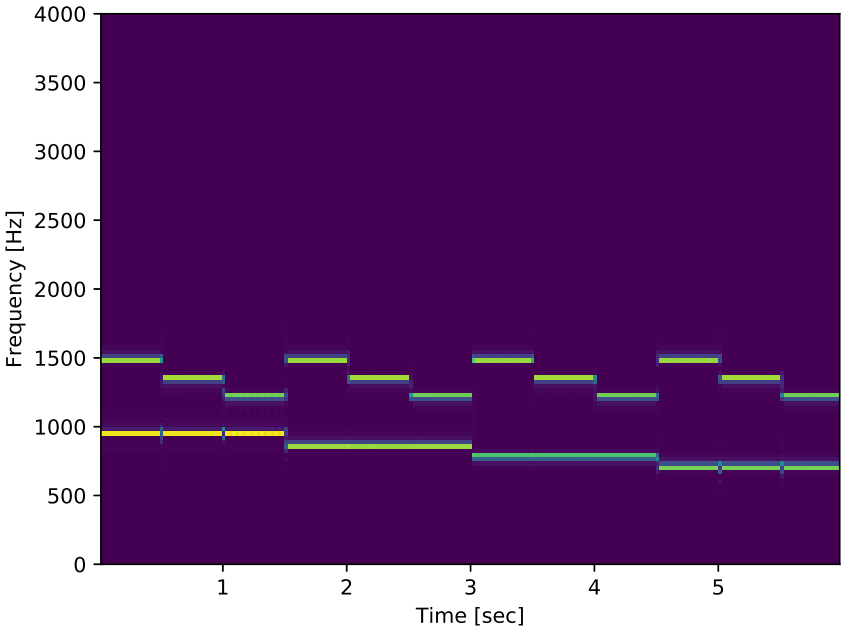

Figure 2: Spectrogram for a specific set of tones

which is based on the example at
https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.spectrogram.html
Figure 2 shows the spectrogram for *a specific set of tones*, so your spectrogram may be different.

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 last name, the last 4 digits of your student ID, and the specific homework, with a hyphen between your ID and the homework number. For example, if John Smith has a student ID of 1000123456 and is submitting hw02, his directory would be named **smith3456-hw02**. Use all lowercase and zero-pad the homework number to make it two digits.

   If you have a hyphenated last name or a two-part last name (e.g., Price-Jones or Price Jones), let's discuss what you should do.

b) Place your .py files in this directory.

c) 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 `smith3456-hw02.zip`.

d) Upload the zip'd file to Canvas.

References

[MSY98] James H. McClellan, Ronald W. Schafer, and Mark A. Yoder. *DSP First: A Multimedia Approach*. Prentice Hall, Upper Saddle River, New Jersey, 1998. This is the book that I used the first time I taught Signal Processing and therefore much of what I know came from it. The authors believe this could be the first course taken by electrical engineering students. I don't agree, but they do assume less at the beginning than traditional texts which means they explain more than traditional texts. Like all academic textbooks for DSP, it's heavy on the math.