CSE 3313 – Homework #10

Assigned: Wednesday, November 13, 2019
Due: Wednesday, November 20, 2019 at 4:00pm
(we may not have class this day; if not, submit at my office)

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.

Assignment:

1. (15 points) The default units for the Python function `scipy.signal.freqz()` are radians/sample. However, it's easier to understand the plot if the x-axis units are in Hertz.

   If we have an LTI filter with a zero at $\pi/2$ radians/sample and our sampling frequency is 512 samples/second, what is the frequency of the zero in Hertz?

2. (15 points) If our LTI filter will use a sampling rate of 3000 samples/second and we want a zero at 1250 Hz, where on the unit circle should you place the zero? Note that you are finding $\hat{\omega}$, which will be in radians/sample.

   Your answer should be exact, i.e., something like $\pi/4$ radians/sample.

3. (35 points) **Application Area: Source Separation**
   Purpose: Learn to separate linearly-combined signals.

   There are many real-world instances of signals that are combinations of other signals. In some cases, we might only be interested in one of these other signals. For example, we might have a recording of a person speaking with some music playing in the background; we might want to hear the person without the music playing. If we know something about the signals that allows us to distinguish them from each other (e.g., they are at different frequency ranges) then we may be able to use filtering to separate them. In many cases, however, we don't have this luxury and it becomes a problem of *blind source separation*.

   Blind source separation is in general a very difficult problem, but we can have some success in specific situations. One situation in which we might be successful is if the signals are mixed linearly. In this case, we can try to use *independent component analysis* (ICA).

   Want to see another cool application? Check out the use of the singular value decomposition (SVD) to remove people from a video:
   `https://heartbeat.fritz.ai/applications-of-matrix-decompositions-for-machine-learning`

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

```
unmixAudio(leftName, rightName)
```

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

Do not create any global variables.

(b) On the course website is a file, `sourceSep.tgz`, that contains

- `darinSiren0.wav` – a mixed signal
- `darinSiren1.wav` – a mixed signal

(c) We will use the FastICA library (it should be included in your Anaconda distribution). See
`https://scikit-learn.org/stable/auto_examples/decomposition/plot_ica_blind_source`
for how to use the library. Note that the example at that link does a lot more than we are doing, including producing the sample signals used in the example.

(d) As part of the process described in the link above, you will need to construct a matrix whose columns are the mixed signals. The left column should be for the signal `darinSiren0.wav`.

(e) Your output will be the two unmixed signals. These will be called `unmixed0.wav` and `unmixed1.wav`; `unmixed0.wav` will represent the left column of the matrix `S_` in the example at the link above. The amplitude of the signals in `S_` are less than they were as input, so multiply the **output** signal values by 10 before writing the output files.

(f) Also create a plot consisting of 4 subplots in the same window, one per row that show

- the time domain plot of `darinSiren0.wav`
- the time domain plot of `darinSiren1.wav`
- the time domain plot of `unmixed0.wav`
- the time domain plot of `unmixed1.wav`

(g) When uploading, DO NOT include the data.

4. (35 points) **Application Area: Frequency Analysis**
Purpose: Learn to analyze data by looking at the frequency components.

One of the most important applications of signal processing is to analyze data thought to be periodic in nature. You will analyze temperature data to better understand it.

The course website has a skeleton file called `weatherAnalysis.py` that you should complete. It calls three functions that you will write:

(a) `getData()` is given the name of a file to read and extracts the temperature from this file. The data file to read is `weather.csv`, which can be found on the course website.

You know the following and SHOULD use it in your code:

- the first row of the data file contains header information, so you should skip it

- the temperature data to use is in the 13th column

  The function returns these temperatures in a list or ndarray of floating-point numbers.

(b) `analysis()` is given the temperatures, the DFT of the temperature data, and the number of temperature values. It prints to the console the following:

   i. the sampling rate. Make sure to include the units. The most obvious options are samples per second, hours, days, or years.
   ii. the fundamental frequency of the DFT coefficients. Make sure to include the units.
   iii. the index of the DFT coefficient with the largest magnitude. Also state what this tells us about the data.
   iv. the index of the DFT coefficient with the 2nd largest magnitude. Also state what this tells us about the data.
   v. the index of the DFT coefficient with the 3rd largest magnitude. Also state what this tells us about the data.

(c) `plotTemps()` is given the temperatures, the DFT of the temperature data, and the number of temperature values. It creates the following plots, each in its own window:

   i. the first 24 hours of temperature readings. The x-axis labels should be in hours.
   ii. the first 7 days of temperature readings. The x-axis labels should be in days.
   iii. the first 365 days of temperature readings. The x-axis labels should be in days.
   iv. the magnitudes of the DFT values for the temperature data. Only display the first half of the values. The x-axis labels should be in terms of your sampling frequency (i.e., seconds, hours, days, or years).
   v. the logs of the magnitudes of the DFT values for the temperature data. Only display the first half of the values. The x-axis labels should be in terms of your sampling frequency (i.e., seconds, hours, days, or years).

Do not create any global variables.

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.