Weldon School of Biomedical Engineering

Biomedical Signal Processing
BME 595 - Fall 2021
Problem Set 1

This problem set accounts for 7% of your final grade. Please submit your solutions as a single PDF report containing typed explanations and plots (Note that it is possible to print off Jupyter Notebooks as PDFs). Keep the text as brief as possible. Each problem lists the plots and results that need to be included in your solutions report. Please label the axes of all plots with appropriate units. You need not include any code in your PDFs unless explicitly requested, but you are welcome to do so.

1. This problem is an exercise in checking that you are able to scale your time and frequency axes correctly when using discrete-time data. Attached is a data file called `mysterysignal.mat` that contains a signal called `x` which has sinusoids of five different frequencies added together. The signal also has a small amount of noise to make it hard to guess the frequencies by just looking at the signal in time domain. The data is sampled at 22050 Hz. You can assume that the sampling rate was high enough for the signal at hand to avoid aliasing. With this signal `x`, use MATLAB or Numpy/Scipy's `fft(.)` function to calculate the frequency-domain representation of the signal. Let's call the result `Xf`.

   (a) Create a frequency vector `f` with the frequency axis in Hz. This frequency vector should be the same length as `Xf`. Now plot the magnitude spectrum of the signal using `plot(f, abs(Xf))`[1]. You should see five distinct peaks corresponding to the five sinusoids. Two of the peaks are close to each other, so you may have to zoom in to see all of them. Include this plot in your report.

   (b) Read off the frequencies of each of the five sinusoids in Hz. Zoom in as needed to see the where the peaks are. All the sinusoids should have frequencies less than 1000 Hz. Use this fact to debug any mistakes you might make in generating the frequency vector. Include the results in your report.

   (c) Read off the approximate relative amplitudes of the first four sinusoids (lowest four frequencies) relative to amplitude of the fifth (highest frequency) sinusoid. Include the results in your report.

   (d) Now let's turn our attention to the phase spectrum. In a new figure, plot the phase spectrum of the signal using `plot(f, np.angle(Xf))` (if you are using Numpy `import`ed as `np`). Because most frequencies only contain noise and the sinusoids are only at five of the frequencies, the phase spectrum will look messy and not interpretable. One way to get around this to threshold the phase spectrum based on the magnitude spectrum. That is, set the phase to zero for any frequency where the magnitude is less than 50% of the smallest sinusoid, by doing something like:
   ```
   phi = np.angle(Xf)
   phi[np.abs(Xf) < threshold] = 0
   ```

   Plot the thresholded phase spectrum using open circles. This would work well because there are only five points you care about. From this, read off the approximate phase of each sinusoid in the mix (in radians). Include the plot, and the phase values in your report.

[1]Plotting functions are available for Python via *matplotlib*

2. This problem deals with time-frequency uncertainty. We will revisit the conceptual ideas behind this notion when we talk about spectrum estimation later in the course. We will use the same signal `x` that is in `mysterysignal.mat` from the previous problem to do the following:

   (a) How long in seconds is the signal `x`? Create a time vector `t` of the same length as `x`. Remember that the sampling rate is 22050 Hz.

   (b) Use the time vector to extract just the first 200 milliseconds of the data. Let's call this `x200`.

   (c) Recalculate the spectrum of the signal using `x200`. Let's call this `Xf2`. Make sure that `Xf2` has the same length as `Xf` in Problem 1. That way you can use the same frequency vector from problem 1 to plot results. This can be achieved by providing a second argument to the `fft` function to specify the number of frequency points.

       This simulates a situation where you happen to have a only short measurement when you are trying to estimate a spectrum. Plot the magnitude of `Xf2` against `f`, zoom in the same way to see the peaks clearly, and include this plot in your report.

   (d) How many peaks do you see now? Comment on the qualitative difference between the plot of `Xf2` here, and the plot of `Xf` from Problem 1. Noting that the difference between the two is the length of the time signal (longer in Problem 1, and 200 ms here), comment on how the time time length of a signal may be related to the frequency resolution of your spectrum estimate.

   (e) Extracting a 200 ms window of a longer signal can be thought of as multiplying the longer signal with a rectangular box that is 200 ms long (i.e., the box is a signal that has a value of 1 for the first 200 ms, and then zero for the rest of the time). Can you use this fact to explain the difference between `Xf2` and `Xf`? Briefly comment.

3. Here, we will use the room reverberation example discussed in class to explore how LTI systems can be characterized by just their responses to impulse inputs. Attached is a file called `stalbans.mat` that contains three variables:
   `h` - The impulse response of the Lady Chapel at St. Albans Cathedral,
   `x` - A relatively echoless recording of a random sentence spoken in a male voice, and
   `fs` - The sampling rate at which `h` and `x` were recorded.
   Once loaded, you can listen to the raw speech by saying `soundsc(x, fs)` if using MATLAB. Within Jupyter Notebooks, an audio player for your your sound waveform can be displayed by saying:

   ```
   from IPython.display import Audio
   Audio(data=x, rate=fs)
   ```

   You can also listen to the impulse response itself similarly (it is basically what a single loud clap may sound like in a large cathedral). Use the `conv` function in MATLAB or Numpy/Scipy to convolve `h` with `x` and listen to the result. Does the LTI model work? That is, does the convolution prediction of what you should hear match what you expect from your real-life experience? Briefly comment.

   Now, instead of convolution in time, apply the room filter by doing a multiplication in the frequency domain, with attention to the following:

   (a) Note that `h` and `x` are different length. First, pad the shorter signal with zeros so that the two become the same length.

   (b) Use `fft` to calculate the Fourier representation of both the speech and the impulse response. Then multiply them to apply the cathedral filter to the speech.

(c) Use `ifft` on the result of the multiplication to convert back to time domain. You might expect that the result here should match the result of the `conv` in time that you did earlier. Does it? Briefly comment. Also include the code used for this problem within your PDF report (not as a separate file).

4. Below are plots of five signals represented in the time domain (left), and the magnitude of their Fourier transforms in jumbled order (right). Absolute units on the time and frequency axes are deliberately omitted. However, the time axis and frequency axis are common across all 5 signals so that any two signals can still be compared in relative terms. Can you match which time-domain signals (A – B) go with which Fourier transforms (P – T)? Describe your reasoning for your solution briefly.

5. Here, let us explore how the random sampling that is inherent in real-word measurements affects the *estimate* of the mean from the measurements. Imagine that there is a small tumor in a patient's post-mortem brain tissue (brain has been extracted from the skull). To get a rough sense of how deep it is, you use a DIY measurement device that has a single ultrasound source, and a single detector[2]. All you can do is create a short clap sound with your ultrasound source, and wait for the reflection from the tumor to come back (much like a SONAR device that may be used under water) and then use the time it takes to come back and calculate the depth using the known speed of ultrasound in brain tissue. The measurements are noisy because there are other boundaries inside the brain besides the boundary between healthy tissue and tumor. Let's say the the noise level is such that the depth measurement has a variance of 100 mm. Let's also say that the true depth of the tumor is 47 mm.

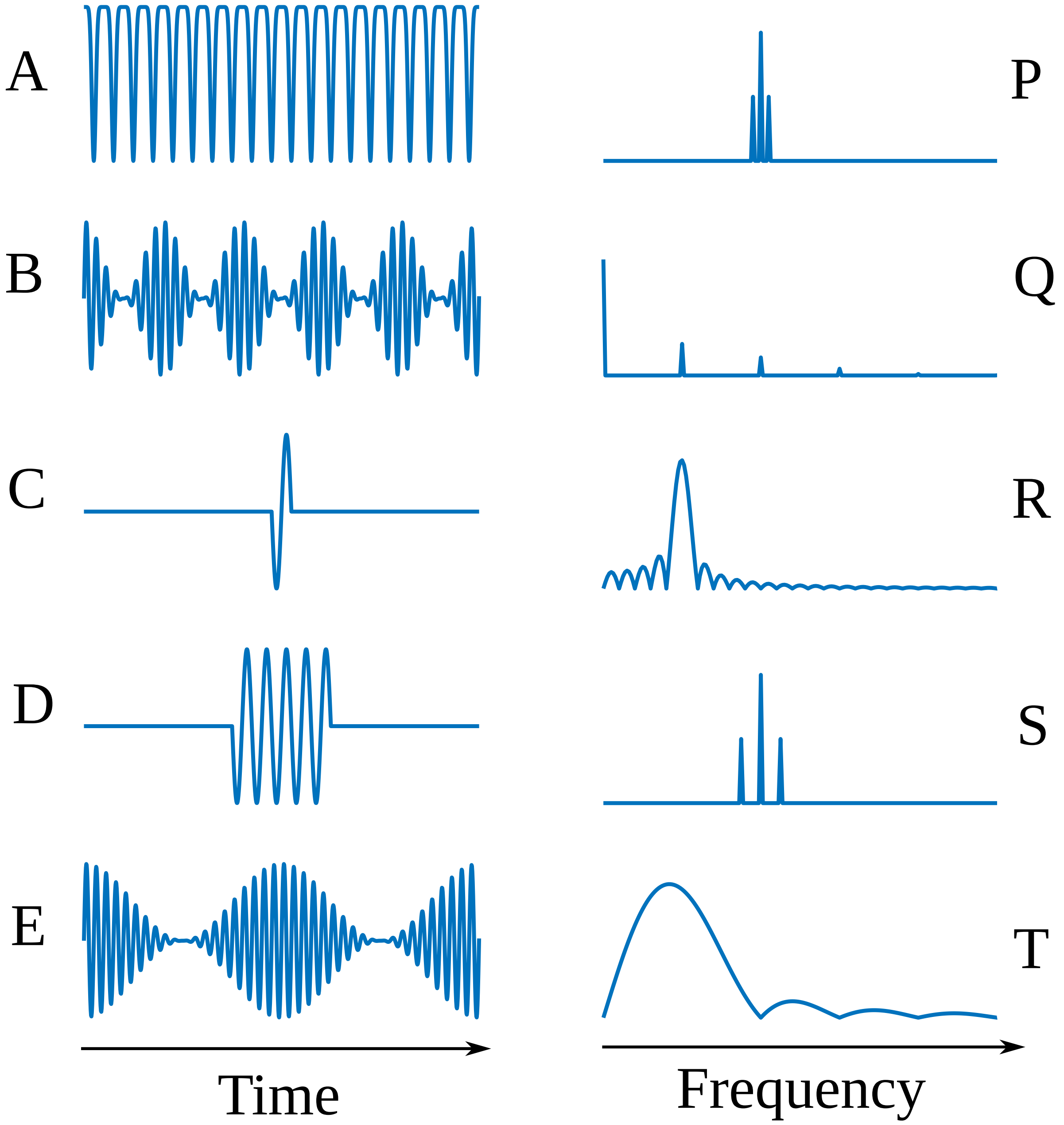
   (a) Simulate an experiment where you do `N` measurements. Assume that the measurements are normally distributed. Because the true depth is 47 mm, each measurement will be a random number whose distribution is Gaussian with a mean of 47 mm and a variance of 100 mm (standard deviation of 10 mm). Write a small script to generate `N=4` such measurements. MATLAB or Scipy's `randn` function will be useful here.

   (b) Estimate the depth by calculating the mean from your measurements. It will of course not exactly be 47 mm.

   (c) Now imagine that you repeat the whole experiment on 100 different days, with making 4 measurements on each day, and calculating an average on each day to estimate the depth of tumor. Simulate this scenario by repeating the whole experiment code 100 times. Now you should have 100 average estimates of tumor depth, one for each day. What is the variance of the estimated tumor depth across days? Is this a smaller number than the variance of a single measurement (which was 100 mm)? By what factor? Briefly comment on why you are getting that factor. How would this factor change if you did `N=25` measurements each day instead of `N=4`?

6. Now let's consider a couple of exercises in basic probability calculations in preparation for the material to be covered in the following weeks. For now, let's simply consider good old coin toss examples, but cast it in a way that will prep us for diving into statistical hypothesis testing (aka detection theory). Let's say we are given one coin and we want to know if it is fair coin, i.e., whether the probability of getting heads (H) or tails (T) is equal and $\frac{1}{2}$ each. A reasonable place to begin is toss it 5 times, and count the number of H you get.

[2]Clinically used ultrasound imaging sensors have an array of sources and detectors. The information from multiple detectors and sources is combined cleverly to produce an image. However here, you just have one source and a detector.


A
B
C
D
E
Time
P
Q
R
S
T
Frequency

(a) What is that probability that you'll get 5 H, assuming that the coin is fair? If you did get 5 H, the probability value that you calculate here is called the "P value" for the hypothesis that the coin is fair. In many fields, if this P-value is less than 0.05, the hypothesis is rejected (i.e., you would no longer consider this a fair coin).

(b) Now let's say 20 of us in this class have one coin each, and each of us does this little experiment of tossing the coin 5 times and counting H. What is the probability that at least one of us will get 5H when *all* the coins are fair? You should find that this probability is much higher than if just one of us did the experiment. This increase in probability of any event when you test for it many times is called the issue of "multiple comparisons" – more on this later.