INTRODUCTION TO DISCRETE TIME SIGNALS AND SYSTEMS

This lab is meant to give you an introduction to working with discrete time signals and systems and also start the "analog" vs "digital" comparison in that you will appreciate the difference between analog and digtal signals and why we would prefer one over the other. It consists of three main assignments and is to be completed over two weeks. This lab totals to 20 points including the assignments and discussions.

LAB INSTRUCTIONS

1. You might have to spend time outside the allocated lab hours to finish the lab. In doing so, you can approach any of the course instructors be it your lab TA, the other TA(s) or the main instructor.
2. You may work in teams or groups of 1-3 members and are not allowed to collaborate with anyone outside your group except the instructors of the course. You may change group or team members for different labs but you cannot change group members for the given lab you are working on.
3. Please document your code well by using appropriate comments, variable names, spacing, indentation, etc.
4. The starter code is not binding on you. Feel free to modify it as you wish. Everything is fine so long as you're getting the right results.
5. Please upload the `.ipynb` file to canvas. One notebook per team is fine and any one team member can upload the file. The required file(s) must be uploaded by the deadline.

1 Implementing Discrete Time Filters to Filter Time-Series Data

In this section, we are going to see how digital filters are implemented on time-series data. Just like frequency domain, the time domain tells us a lot of important information about signals and their associated phenomena. But, more often than not, these signals will not be perfect for analysis as they will almost invariably include some sort of corruption in the form of noise, outliers, etc. We are going to see some ways of dealing with some commonly occurring noise and means of corruption. This is not an exhaustive list but is meant to give you a head start into how you would go about analysing time-series data.

1.1 Implementing a Running Mean Filter (2 points)

The running mean filter works wonders in filtering out high levels of random noise in your signal. It works by replacing each point of the signal by the mean of a given number of points around the point and the point itself. The exact

number of points on either side of the point in question is a parameter that is left to our discretion. Let us call this parameter `k`. To implement the mean filter, use the following sequence of instructions:

(a) You can start by defining your `k` value to be 20.

(b) Define a random noise signal to be added to the base signals.

(c) Now create two new noisy signals which are just the result of adding the same noise to the two different base signals.

(d) With the defined value of `k` apply the mean filter to both the signals. To do this, loop through the signal points starting from `k` and replace each point by the average of `k` points that come before it, `k` points that come after it and the point itself.

(e) Create two subplots, one for each signal. Each subplot must have the original signal in the first subplot, the noisy signal in the second subplot and the filtered signal overlaid on top of the noisy signal in the third subplot. Your results should look like the plot shown below. Be sure to label your plots and use a legend of the third subplot in both cases to show which one is the filtered signal and which one is the noisy signal.

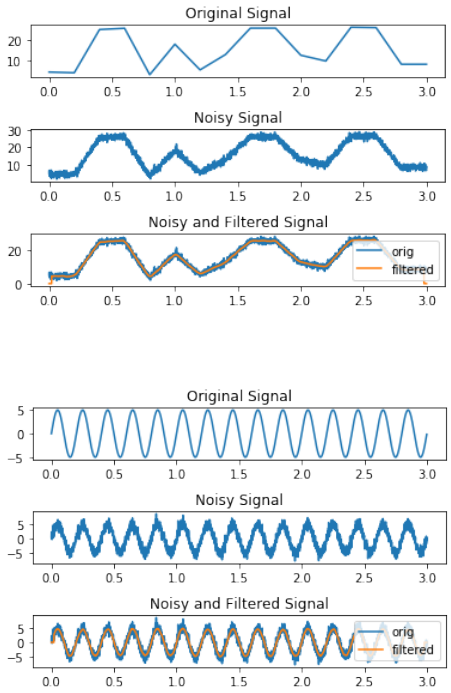

Figure 1: Expected Result for Assignment 1.1

Discussion (1 point)

- Comment on how the results and plots change when you amplify the noise more and also change the value of `k`.
- Mention and explain any ONE of many possible drawbacks of the mean filter in analysing noisy time-series?

1.2 Implementing a Median Filter to Remove Spikes (2 points)

The median filter is very good at dealing with outliers in data. To observe and appreciate this, a signal with noise in the form of sharp spikes has been provided. You will denoise the data or signal building on top of the starter code and following the sequence of instructions given:

(a) Run the code to visualize the noisy signal. The original signal is essentially the signal you see minus the spikes and this is evident from the code as well.

(b) The spikes will be removed by applying a threshold to determine which points are indeed spikes and which points are not. To figure out what this threshold is it helps to plot a histogram of the signal to see where most of the points lie. Use `plt.hist()` to plot the histogram of the signal.

(c) Now, from the histogram, choose a threshold that would successfully, differentiate the signal from the spikes.

(d) Find all the points in the signal whose value exceeds the threshold using `np.where()` and store it in the variable `suprathresh`.

(e) Now apply the median filter at only those points where the signal value exceeds the threshold (the points contained in `suprathresh`). To do this, take the median of `k` points on either side of the point of interest and the point itself. It's essentially the same as the mean filter but we are computing the median instead of the mean. (Hint: use `np.median()` to compute the median and be mindful of the bounds on the signal and the time window for the filter)

(f) Plot your result by overlaying the filtered signal on top of the noisy signal in the same plot.

(g) The three plots you will generate for this assignment should look similar to the ones below.

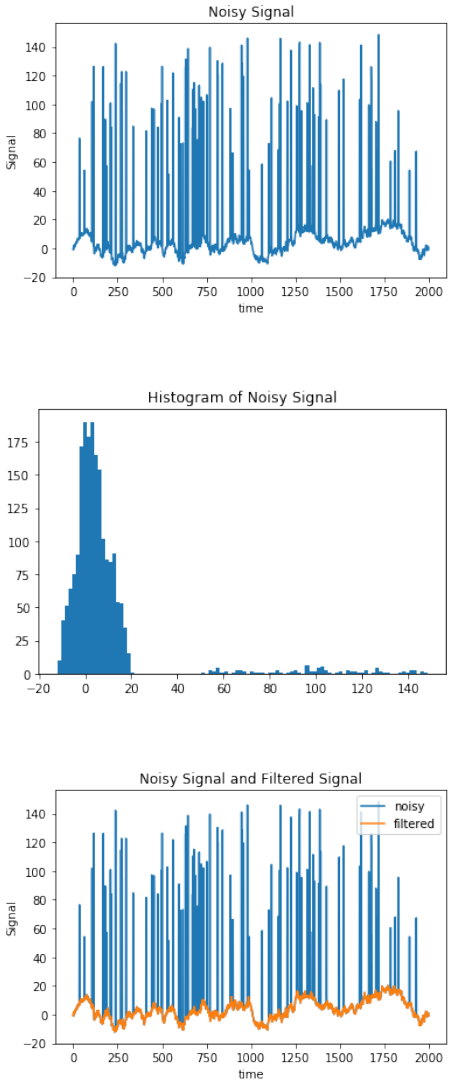

Figure 2: Expected Result for Assignment 1.2

Discussion (1 point)

- Compare the mean and median filters in terms of their uses and one advantage and disadvantage one has over the other.

1.3 Denoising an EMG signal (2 points)

Electromyography (EMG) is a measure of motor unit activity of the muscles of macro-organisms such as humans, animals, birds, etc. It can be used to analyse

movement, diagnose muscle problems and all kinds of other things. But when the EMG is extracted, it is not noise free (as most things aren't!). There are several features of the EMG one would be interested in. But we are just going to look at a time-domain feature in the abrupt changes as they signify muscle activity. So, checking when muscles are triggered could be of importance for many clinical reasons.We will see how the signal can be pre-processed to better lend itself to detection of spikes in activity using the Teager-Kaiser energy-tracking operator (TKEO). You will denoise and pre-process the EMG signal with the following sequence of instructions:

(a) The starter code will start you off with reading in the signal and extracting the variables of interest. Please build on top of that.

(b) The TKEO algorithm is given as $y_t = x_t^2 - x_{t+1}x_{t-1}$. Here, let y_t be a point in the filtered signal at index t and let x be the input EMG signal and x_{t+1} and x_{t-1} are points in the EMG signal at indices one after and one before the index t respectively. Fill in a new filtered signal whose indices are updated with TKEO algorithm described.

(c) Notice that the original EMG signal and the filtered signal obtained as a result of TKEO are not on the same scale or units. EMG is measured in μV or mV and the TKEO filtered signal is measured in energy. This is common, and you must be aware that this can happen. There are several ways to deal with this and one such way, which will be implemented in the lab, is converting the signals to a corresponding Z-score signal. Recall from your statistics class that the Z-score is essentially how many standard deviations away from the mean the current value in question is. So, to implement this:

- Find the mean of all the points of the signal to the left of the 0s time point as that is the base we are comparing against.
- Find the standard deviation of the same slice of the signal from the previous part.
- Subtract the mean calculated from the signal and divide by the standard deviation. Do this for both the original and filtered signal (obtained from TKEO in part b). This should give you two new signals, one Z-score signal corresponding to the original EMG signal and the other corresponding to the filtered signal from part b.

(d) Plot your results showing the filtered signal overlaid on top of the original signal for both methods. Use separate plots, one for the TKEO method and the other one for the Z-score scaled result.

When plotting the TKEO results make sure you normalize the signals being plotted by dividing the signal by it's maximum value so that the range of values in both signals are capped at 1 and can be compared properly.

When calculating and plotting the Z-score, the original signal refers to the original signal also converted to the corresponding Z-score signal.

(e) The expected result should look something like what is shown in the figure below. Even if does not exactly match the result shown below, it must be clear that the spike is very distinct and is the only sign of activity with all else being near zero.

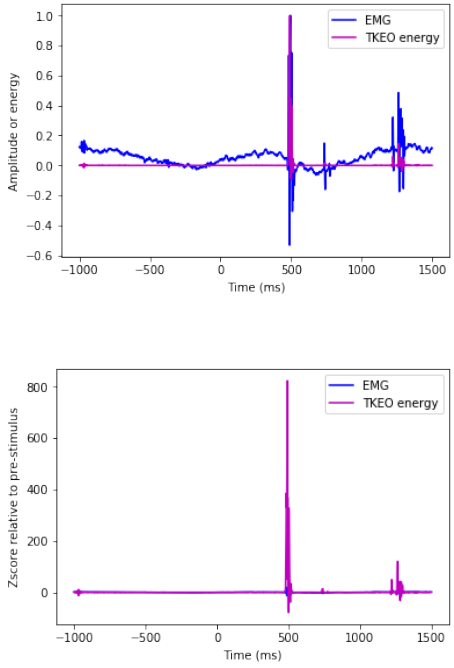

Figure 3: Expected Result for Assignment 1.3

Discussion (1 point)

- How would the other two filters implemented, i.e, the running mean and median filters fare against the TKEO method in analysing the EMG signal in this fashion?
- If you had to use a running mean filter or a median filter to analyse the EMG signal to detect muscle activity, which one would you prefer and why?

2 Convolution (3 points)

In this section we will revisit convolution, which you learned about in EE 235, but this time there's more of an emphasis on it being a convolution sum rather

than the convolution integral. You will implement your own convolution sum with a basic signal and a basic kernel. Both of these basic signals will be defined by in-house functions that you will write by not using any extra libraries like `numpy` or `scipy`, etc. To build on top of the starter code and finish this section follow the sequence of steps given:

- Start by filling in the functions for the basic signals,i.e, the ramp and the step signals.
- Now create a 4 second long ramp and a 4 second long step function and store it as directed.
- Fill in the for loop that is missing the main steps that define the convolution sum,i.e, taking a portion of the signal and multiplying it with the kernel and summing the resulting products of points.
- Verify the visualization with the result below.

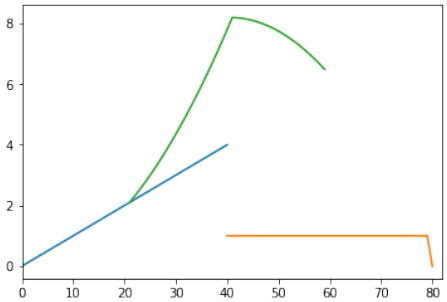

Figure 4: Expected Result for Assignment 2

Discussion (2 points)

- If you do not normalize your computed convolution sum by dividing by the sampling rate, the magnitude is actually incorrect and does not represent the convolution result. Why is dividing by sampling rate or multiplying by sampling time important? (Hint: you have done this in EE 235, check back with the relevant labs and you should find your answer)
- How is the convolution sum different from the convolution integral?

3 Analog Transmission vs Digital Transmission (4 points)

In this section, we will explore the potential advantages of digital transmission over analog transmission. We will consider the case of transmission over a long (e.g. transoceanic) cable in which several repeaters are used to compensate for the attenuation introduced by the transmission.

Remember that if each cable segment introduces an attenuation of $1/G$, we can recover the original amplitude by boosting the signal with a repeater with gain G. However, if the signal has accumulated additive noise, the noise will be amplified as well so that, after N repeaters, the noise will have been amplified N times:

$$\hat{x}_N(t) = x(t) + NG\sigma(t)$$

Here, $\hat{x}_N(t)$ is the resulting signal after all the transmission is complete, $x(t)$ is the original signal that was to be transmitted and $\sigma(t)$ is some random noise introduced into the signal during the process of transmission.

If we use a digital signal, on the other hand, we can threshold the signal after each repeater and virtually eliminate the noise at each stage, so that even after several repeaters the transmission is still noise-free.

The following are the steps that will take you through the starter code and finishing the section:

(a) The first cell in the section introduces you to how one can play audio in the notebook and also gives you a signal to work with for the assignment. Execute the cell and make sure that the audio is playing properly and that it's waveform is displayed.

(b) The next cell creates two versions of the audio signal, an "analog" version and a "digital" version. Obviously the analog version is just a simulation, since we're using a digital computer; we will assume that, by using floating point values, we're in fact close enough to infinite precision. In the digital version of the signal, on the other hand, the audio samples will only take integer values between -100 and +100 (i.e. we will use approximately 8 bits per audio sample).

(c) Remember that there is no free lunch and digitization implies a loss of quality; this initial loss (that we can minimize by using more bits per sample) is the price to pay for digital transmission. Compute and plot the error signal.

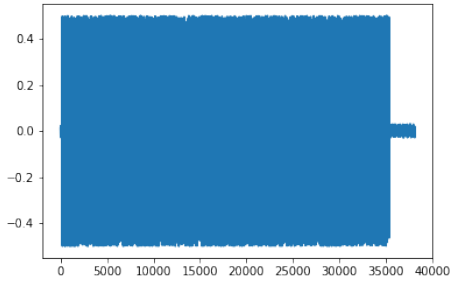

Figure 5: Expected Result for Assignment 3 part c)

(d) Now define a function that calculates the Signal-to-Noise ratio (SNR) of a given pair of signals with one being noisy and the other being the original signal. The function prototype has been laid out for you. Continue to fill it in and calculate the SNR of the above signal by assuming that the "digital" signal is the noisy signal that has been corrupted by the process of digitization and that the "analog" signal is the original signal and print out the SNR. You should get SNR = 17.124344 dB or something close.

(e) Now define a function that represents the net effect of transmitting audio over a cable segment terminated by a repeater. For this, translate the following effects into python code in the function prototype given for the repeater:

- the signal is attenuated
- the signal is accumulates additive noise as it propagates through the cable
- the signal is amplified to the original amplitude by the repeater

(f) We can use the repeater for both analog and digital signals. Transmission of the analog signal is simply a sequence of repeaters. For digital signals, however, we can rectify the signal after each repeater, because we know that values should only be integer-valued. To implement analog and digital transmission through the sequence of repeaters given:

- Fill in the analog_tx and digital_tx by passing the signal through the repeater num_repeaters times by calling the function for the repeater.
- But with digital transmission be sure to make sure the signal is still integer valued by rounding off values.

(g) With the given parameters, find the final signal that is received after transmission and store the "analog" and "digital" versions in the given variables and print the SNR values for both, the analog signal and the digital signal. You should see that the SNR for the digital transmission remains almost the same and the SNR for the analog transmission is defitely worse than that of digital.

Discussion (2 points)

- Mention and explain any ONE advantage that analog signals have over digital signals (It could be a use case, some characteristic property, etc).
- in part d) of this section you found the SNR of the signal which is a measure of the consequence of digitization. Play both the "analog" and "digital" audio and comment on whether you can hear the difference that corresponds to the resulting SNR value.
- In some cases digital transmission performs more poorly than analog! This can be worked around of course, but for the given signal and parameters of the transmission system, can you identify and explain why? (Hint: try increasing the amplitude of the noise in part g))