**EE430 Lab 2 Writeup**

Section I – Narrative

Problem 1)

1. The function **sampled\_spectrum** was created in MATLAB. The source code for which can be found in the third section of this document.
2. The function **reconstructed\_spectrum** was created using MATLAB, source code located in Section III – Source Code.

Problem 2)

1. A value fs = 120 was chosen as the sampling frequency, as this is twice the Nyquist limit, and would ensure the given sample frequencies would be able to be reconstructed. Plots for s(t), s[n], and a reconstructed s(t) were generated, and can be found in the second section.
2. A new sampling rate was chosen to create aliases in the signal, fs = 100. This would cause the 60 Hz. frequencies to be outside of the Nyquist limit. The same six plots as the previous section were then graphed, and can be found in Section II – Figures.
3. A lowpass filter was applied to s(t), which cut out all signals below half of the sampling frequency. The altered signal then underwent the same process as parts i & ii, revealing that the signal could be accurately reconstructed, so long as there were no frequencies present in the signal above fs/2.

Problem 3)

1. The spectrum of the given signal s(t) was calculated by hand, and is included in the fourth section of this document.
2. A sampling frequency of 1000 Hz for the given signal was not sufficiently high enough to prevent aliasing, as there are frequencies present that are larger than half of that sampling rate in magnitude.
3. Plots for this s(t) were created in the same way as problem 2, and the resulting figures are attached in section 2 of this document.
4. S(t) and the reconstructed version of s(t) were plotted on the same graph, and a stem graph showing points that are a multiple of 1/fs was overlaid upon it. For every even integer value n in (n/fs), the two graphs shared a point. This graph can be found in the second section of this document.

Problem 4)

1. Using audacity, a voice sample of a podcast introduction was used to create a monaural wav file, with a sampling rate of 44.1 KHz.
2. The audio file from the previous section was loaded into MATLAB using the **audioread** function. The operation of this function was verified by playing the audio back within MATLAB by using the **sound** function.
3. A Gaussian white noise signal was generated within MATLAB by using the **randn** function. Then, utilizing the average values of the generated audio, constant values s1 & s2 were created that allowed for the original signal to be combined with the noise. Both signals were listened to, and the resulting audio form was plotted. These graphs can be found in section 2.
4. A FIR filter of length 5 was created, with each value within being a constant ‘1’. The noisy signals were then filtered with the filter using the **ydenoised** function. This revealed that the function with a higher original SNR was improved more than the signal with a 0.8 SNR after filtration. The signal with an SNR of 2 sounded muffled, but greatly improved, while the other produced a high-pitched whine sound, which was arguably worse than it had been originally. Both signals were then plotted and are attached in section II.
5. For a length 5 filter, the signals were revealed to have SNRs equal to 18.0154 for the 0.8 signal, and 3.147 for the 2.0 signal. This seemed to imply that a higher SNR lead to a higher audio quality. To test this, the same filter was adjusted in length from 1-100, and then applied to the previous signal. The magnitude of the SNR was then plotted versus the length, which can be found in the second section. The optimal signal length, which is to say the one that resulted in the highest SNR, was shorter for the higher SNR signal, at 9, and 21 for the other signal. For both signals, the audio was improved, although the sound seemed to be more muffled on the signal that required a longer filter. This may simply be due to the averaging of more points, causing the original signal to lose amplitude overall.

Problem 5)

1. Five numbers (2, 3, 4, 1, 2) were chosen to act as an order 4 FIR. A single normalized frequency was then chosen, setting w = 0.4pi, which was then used to generate a 500 samples of a sine wave, which was then placed in an array X. The FIR filter was then convoluted with the generated sine wave, and the result was plotted. This graph can be found in the second section. It was noticed that the resultant sine wave appeared to be jagged, like a sawtooth function, and the magnitude was 2.427.
2. A **for** loop was utilized in MATLAB to then duplicate the process used in part i for many normalized values, all being an even percentage of the number pi. The value of the resulting magnitude was then saved for each generated sine wave in an array, which was used to plot the magnitudes of the generated sine waves against the value used for the frequency. This graph, as with the previous ones, can be found in section II.
3. The filtered signal formula given by this part was factored to give the requested result. This calculation can be found in the fourth section of this document. This formula was then used to plot the frequency response of the chosen filter versus the frequency. It was noted that the plot resulting was similar to the one generated in the previous section, following the same basic curve. Although, the graph in section ii had a small increase as it approached pi that the one generated for this section lacks.
4. Parts ii and iii were repeated, but in this case the FIR filter was chosen such that the coefficients added up to 0. (1, 2, -2, -1). The generated graphs from this process can be found at the second section of this document. It was observed that the signal had a frequency response that was lower in magnitude as it approached a value of 0.
5. The graph for part 3 was reproduced using the **freqz** function. This graph can be found in the second section of this document.