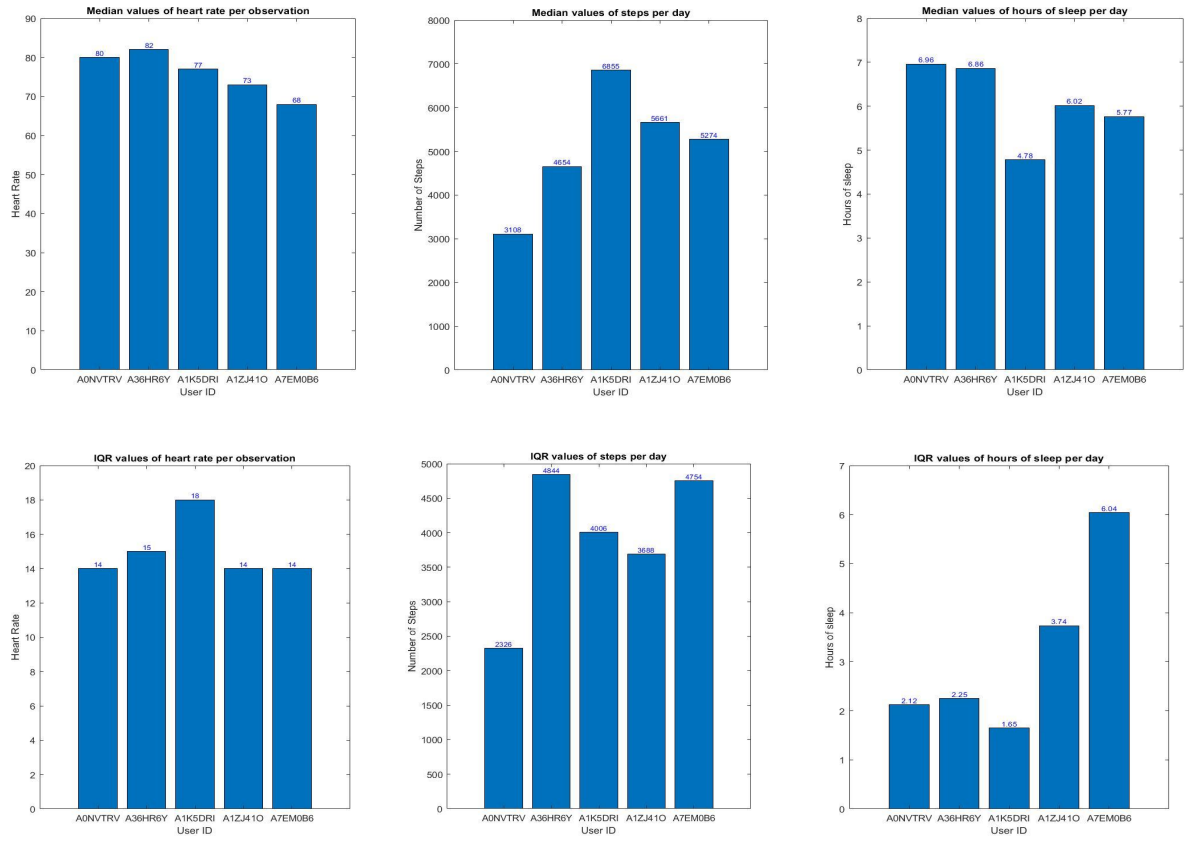
Lab 1 - Biostatistics and Signal Processing in MATLAB

Part A

As described in the abstract of "Pre-symptomatic detection of COVID-19 from smartwatch data" from Mishra et al. (2020), the objective of the paper was to determine if vital signs measured by wearable devices can be used to detect the presence of an infectious disease such as coronavirus disease 2019 (COVID-19). The findings of the studies are as follows: 81% (26) of 32 individuals infected with COVID-19 had abnormal heart rate, steps per day, and hours of sleep; 22 of those cases were detected before symptoms arose; 63% of the cases could have been detected before the presence of symptoms using smartwatch data recording irregular heart rate. This led to a conclusion that the health monitoring applications of wearable devices can possibly be used to detect respiratory infections and diseases before symptoms occur (p. 1208).

Figure 1

Central tendency (median, top row) and dispersion (IQR, bottom row) measurements of heart rate, steps, and sleep hours between five different patients



Note. The values, if too small to read, for each graph is as follows (from left to right, top to bottom): {80, 82, 77, 73, 68}, {3108, 4654, 6855, 5661, 5274}, {6.96, 6.86, 4.78, 6.02, 5.77}, {14, 15, 18, 14, 14}, {2326, 4844, 4006, 3688, 4754}, {2.12, 2.25, 1.65, 3.74, 6.04}.

The code used to determine these measurements and create the graphs were done in MATLAB as per Appendix A. The interquartile range (IQR) was used for the measure of disparity and the median was used for the measure of central tendency, both because they are robust in the presence of possible outliers.

The major trend seems to be that heart rate is a stable, consistent measure between all 5 patients (especially when observing median and IQR), however steps and sleep hours are varied drastically between patients, with fluctuations becoming even more irregular between patients (for example for sleep ours, an IQR of 6.04 was recorded for one patient and 1.65 for another, which is contributes a disparity difference of more than 4 hours). These graphs show that the disparity and central tendency of steps and sleep hours per day is more varied between patients, whereas heart rate is less varied.

The code is also used to create a correlation table between the heart rate (variable X) and number of steps (variable Y) observations. This created a 2-by-2 matrix that will display correlation between XX, XY, YX, and YY combinations. This was done 5 times for each patient in this study, however the most notable observation was that from patient A1ZJ41O, which had the highest correlation between heart rate and number of steps in both directions at 53.99%. All other patients were observed with a lower correlation, meaning that at the very most, 53.99% of the variance in the heart rate variable is also attributed to that in the number of steps and vice versa. This is not enough to correlate the two variables with strong assurance, meaning a conclusion cannot be made about the correlation between the two.

Some statistical methods used in the previous article included calculating the median and average for heart rate, as well as a total summation for sleep hours (p. 1219). In later steps outside of what was done in this report, null distributions were used to calculate P-values to determine the probability COVID-19 can be detected from smartwatch data (p. 1214), as well as boxplots to represent fluctuations in heart rate (p. 1213). These are definitely way more advanced and precise than what was achieved in MATLAB, especially since they were applied to real-time, specific data provided by patients. In comparison, the statistical measures used in this report are very fundamental and basic. These arise from various limitations that are present in this report, with the most prominent ones being less time, as well as access to less resources and more advanced knowledge in statistical analysis methods

Part B

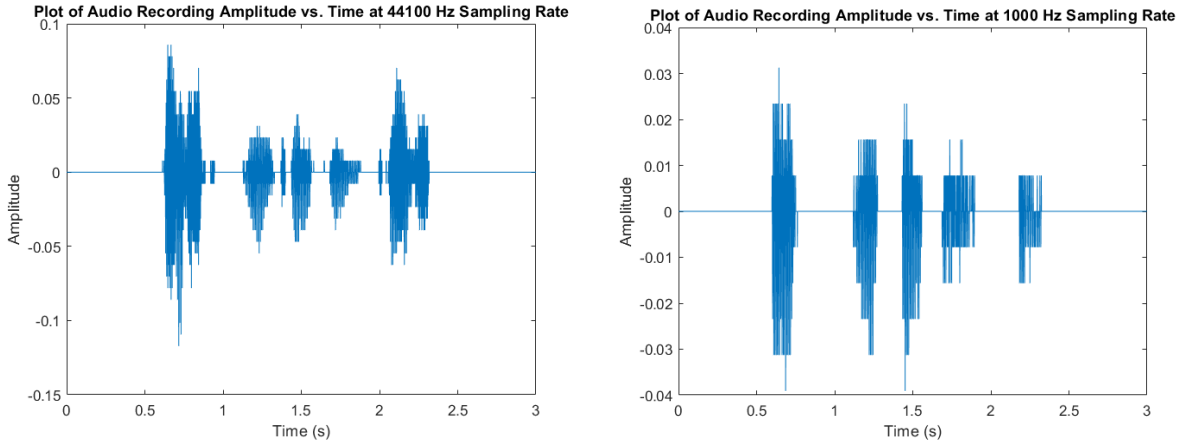
The input arguments of the audio recorder function in MATLAB are described as follows: Fs, which represents the sampling frequency in Hz (must be between 1000 Hz and 384 000 Hz as restricted by MATLAB's software); nBits, which is a number either 8 (for integers) or 16 and 24 (for floating point values) to represent the number of bits used per sample; NumChannels, which is either 1 or 2 to dictate whether the samples are recorded in one channel (mono) or two channels (stereo); ID, which denotes the ID of the device that will be used to record the audio (-1 is the default ID, otherwise the audiodevinfo function can be used to specify a device ID).

The domain of the graph at the original sampling rate (44100 Hz) was initially in $[0,14*10^4]$, based on the number of samples recorded within the 3 second window set in the code. However, after manually setting the time axis, it was changed to reflect this time window in [0,3]. While the graph seems to look continuous (connected), since the samples were recorded in discrete time, the time and the amplitude recorded are both discrete.

When comparing the sound at 1000 Hz sampling rate to that at the original 44100 Hz, it was evident that the downsampled sound was more muffled and quiet. This was due to more information being cut out and missing because of the lower sampling rate (which leads to a lower number of samples). Comparing the graphs for each scenario also reflects this relationship, since the plot at 44100 Hz seemed to display much more detailed and varied peaks, whereas the 1000 Hz plot showed more blocky and static peaks (as per Figure 2).

Figure 2

Plot audio recording at 44100 Hz sampling rate and at 1000 Hz sampling rate



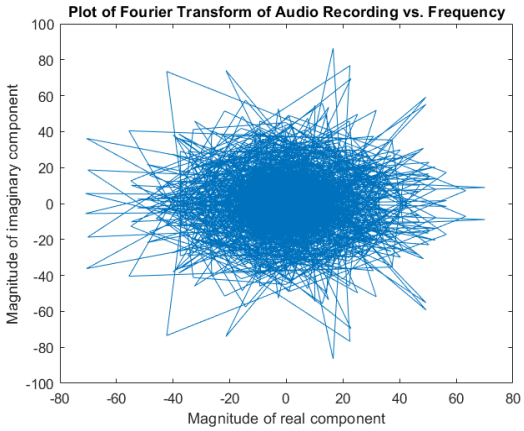
Part C

In terms of the Fast Fourier Transform (FFT) function in MATLAB, the function is used to find the frequency component of a noisy signal from its time domain. The input and output of the FFT function are discrete, much like the Discrete Fourier Transform (DFT) counterpart. This means that the frequency domain is also going to be sampled discreetly, much like the time domain when it was recorded. Looking into the function further, it becomes evident that FFT is a more efficient way of using DFT. Whereas in class, DFT is utilized by applying the definition to each individual sample linearly across the length of the signal, FFT does so by applying said definition in chunks to be more efficient.

As seen in Figure 3, what is shown in the plot is a messy convolution of the imaginary and real component of the Fourier Transform. The respective axes represent the amplitudes of the real and imaginary components, which does not provide clear information about the transformed data. This is why the graph seems to be jumbled and busy, as it is trying to label data for both the imaginary and real components at the same time.

Figure 3

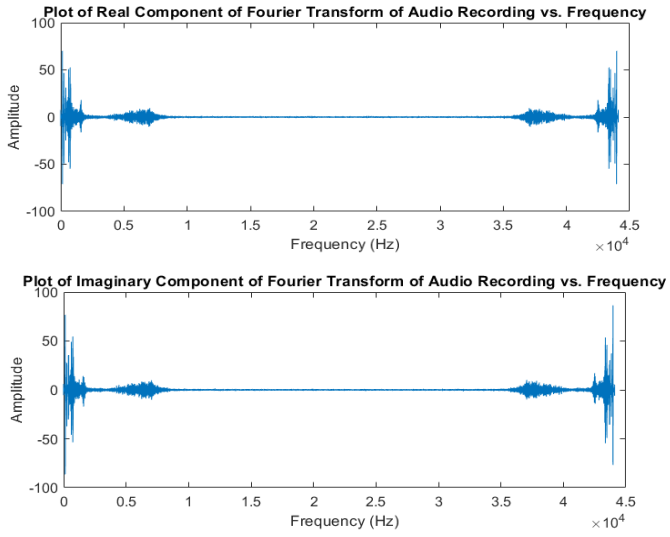
Plot of Fast Fourier Transform of the original audio recording and of only the real component



When isolating the real and imaginary, it becomes cleaner as each axis can be plotted against frequency values, showing the strength of the audio recording at respective frequencies. As seen in Figure 4, it can be seen that both components share a similar shape, but have differing values due to a change in phase. On their own, each component does not provide a specific characteristic of the samples (other than the real component providing a cosine wave and the imaginary component providing a sin wave). However, both are required to provide a full picture on the sample against the frequency domain. While plotting the imaginary component against the domain of frequency is not recommended, it was included anyway to compare the phase against the real component.

Figure 4

Plots of FFT in the real (top) and imaginary (bottom) domain against frequency



Part D

Gaussian white noise is a normal distribution of amplitudes which are added to an original signal to create random superpositions and distortions to the signal. Because the

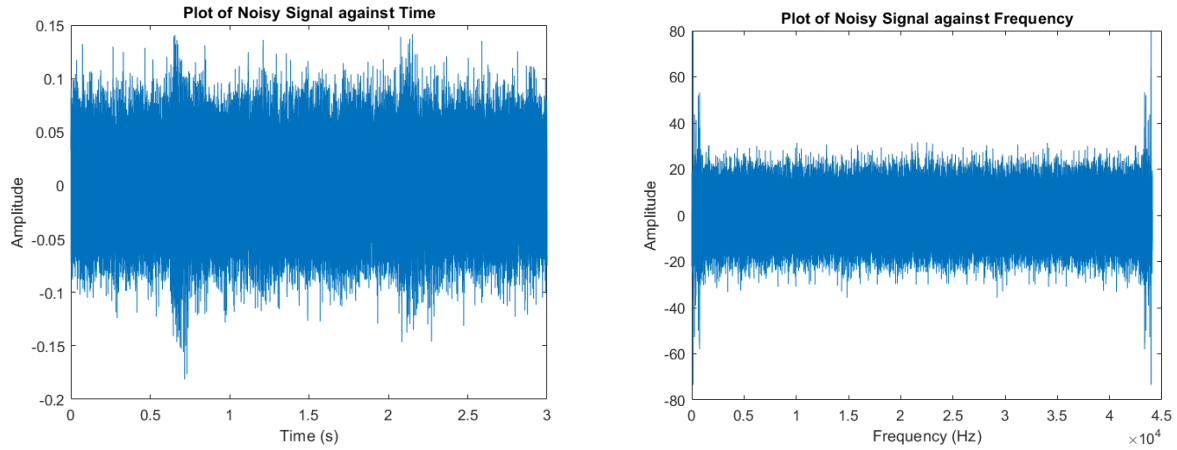
frequency follows a normal distribution, it is generally preferable to filter out the white noise using an FIR filter, usually a low-pass finite impulse response (FIR) filter. This filter passes low frequency samples, but will eliminate samples in high frequencies, which is useful since the added Gaussian white noise would create higher frequency data.

In order to create Gaussian white noise in MATLAB, the awgn function will be used. This function comes with a parameter called snr, which represents the signal-to-noise ratio of the data. The signal-to-noise ratio dictates by what proportion noise is added to the original signal. For instance, if the signal-to-noise ratio is 2, that means the amplitude of the wave is roughly twice that of the presumed noise. The larger the signal-to-noise ratio, the more noisy the sound. All code used can be found at Appendix C.

In the previous part with the recording without noise, it was observed that low frequency showed almost zero amplitude and there was lots of silence in between the talking sections. However, upon seeing the plots of the noisy signals against time and frequency, as per Figure 5, it is clear that all those silent sections have been riddled with loud, static-sounding noise. Despite there being no sound in lower frequencies beforehand, the additive of Gaussian white noise filled in sound for those lower frequencies, hence why a low pass FIR filter would be used.

Figure 5

Plots of original sound sample with added Gaussian white noise vs. time and frequency domains

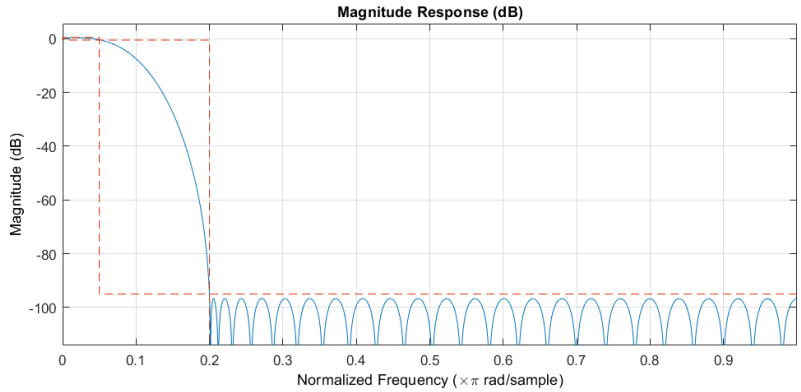


When observing the Fourier Transform plot of the non-noisy signal from Figure 4, it shows that there is noise happening from frequencies 0 to ~1500 Hz. We know that the rest of the frequencies above it should be close to, if not exactly 0 in amplitude. Using this knowledge, the passband and stopband frequency can be adjusted to this range and then fine-tuned. After testing with multiple combinations the passband frequency chosen was 545 Hz and the stopband frequency was 1200 Hz. This means that any frequencies between 545 and 1200 Hz are accepted, while anything else above is rejected. Furthermore, the sampling rate was kept the same as the original, with a passband ripple of 1 dB (meaning fluctuations in the passband permissible have to be within 1 dB) and stopband attenuation of 95 dB (which dictates the max

range between the lowest gain in the passband and the highest in the stopband). This exact model can be observed when running these parameters into the fvtool visualizer.

Figure 6

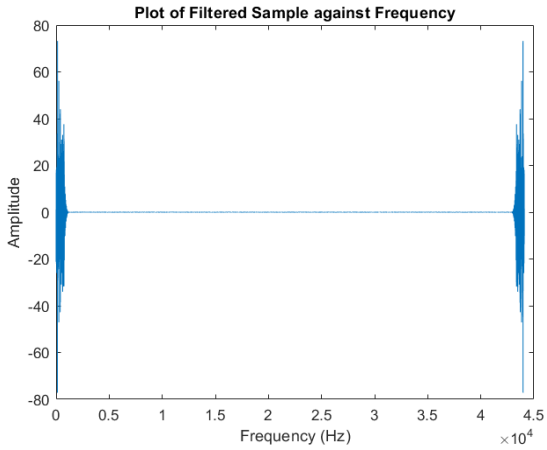
Visualization of the low pass filter using fvtool



Upon applying the filter to the signal, what is observed is that the amplitude of frequencies within the stopband frequencies (greater than 1200 Hz) have been completely decreased to 0. Listening to the filtered audio, the noise is almost completely gone but the voices are quieter and more muffled due to removing those frequencies. When comparing the plot of the filtered signal (as per Figure 7) to those of the regular and noisy signal (Figure 4 and 5 respectively), the fluctuations of noise are completely gone. Even though there was some "noise" in the regular signal, these were forced out completely in the filtered signal. Although the beginning and end sections that were within the passband seem to have been retained perfectly.

Figure 7

Plot of filtered signal against frequency



Part E

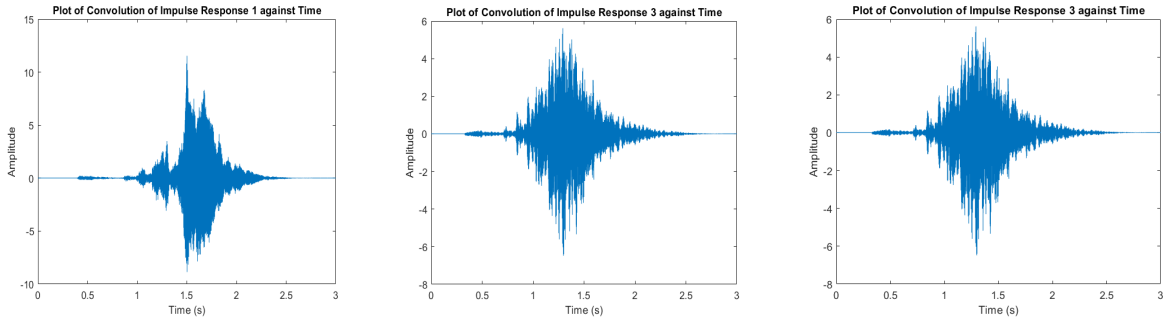
The main purpose of convolution reverb is to take an audio signal and utilize another audio signal from a different environment to make it sound like the original audio signal was recorded in that environment. This is achieved by utilizing an impulse response from that

environment, since a sharp, quick sample causes clear reverb in the signal, from which it can be processed with the audio signal in question to create convolution reverb (Lainhart, 2017). This was exemplified in three samples downloaded from *Echothief*: Square Victoria Dome, Battery Benson, and Purgatory Chasm.

Upon convolving the audio file with the three impulse responses, it becomes evident that the sound has drastically changed. While the words spoken in the audio file are still recognizable, it is set with a completely different reverb and atmosphere. Especially for the third convolution: while the original recording did not have any echo, convolution with the third impulse response created a reverberating echo after each word was said. This drastic change can also be seen when analyzing the convolved signals are plotted against time, as per Figure 8.

Figure 8

Plots of each convolution against time

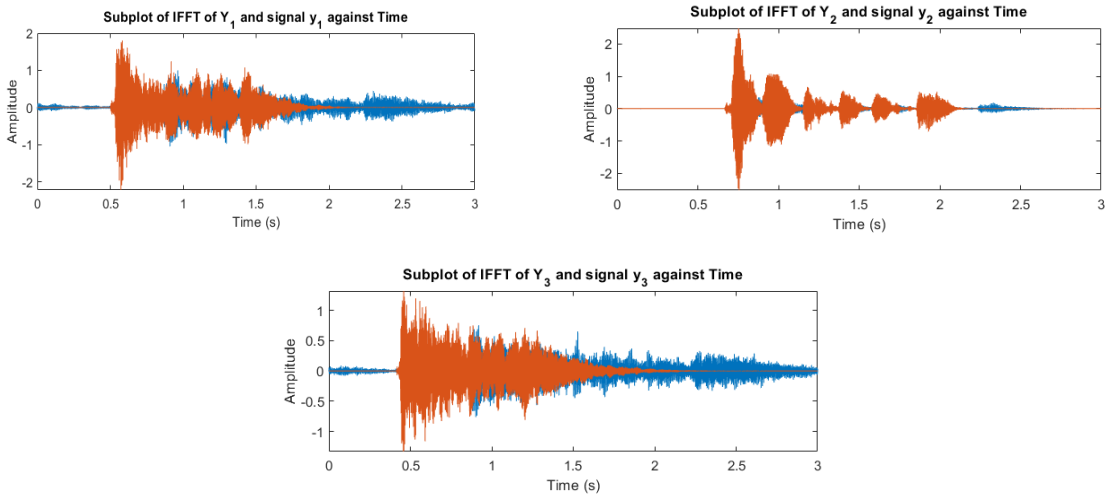


The FFT of each convolution was calculated in two ways: matrix multiplication between the FFTs of the impulse response and original signal and a straight FFT of the convolution. 6 graphs in total, 3 for each method, were created to compare the two methods (can be found at Appendix D). Upon observing the graphs, it seems as though both methods seem to product the exact same FFT graph (with minor alterations simply because the domain of the FFT of the impulse response had to be extended in order to properly perform element-wise multiplication with the FFT of the original signal). This is because convolution is distributive, meaning that an operation (like FT) can be applied both to the entire convolution or each one individually.

Figure 9 shows how the inverse FFT (IFFT) of Y_i compares to the convolution signal y_i. We can see that the two pairs in each group are very similar, but it seems as though that more information was lost in this transformation than the previous one. This could very well be because there was already a loss of information in the previous step due to the matrix multiplication, but a second transformation was done on top of that, which caused the disparity to increase. However, the general trend seems to dictate that doing the IFFT seems to accurately reverse the FFT done both individually and in convolution.

Figure 9

Subplots of the IFFT of Y_i and y_i



The phenomenon we observed through this trend is the effect of duality between the frequency domain and the time domain. Specifically, the multiplication of one domain is the same as convolution in the other domain. So in this scenario, if the audio signal is convolved with itself, this is convolution in the time domain. This would equate to multiplication in the frequency domain, meaning that the amplitude of the audio signal will be squared in the frequency domain. This would ultimately lead to a major amplification and elongation of the original sound.

Works Cited

Echothief. EchoThief. (n.d.). Retrieved January 26, 2022, from http://www.echothief.com/

Lainhart, R. (2017, September 22). *What is convolution reverb?* Ask.Audio. Retrieved January 26, 2022, from https://ask.audio/articles/what-is-convolution-reverb

Mishra, T., Wang, M., Metwally, A. A., Bogu, G. K., Brooks, A. W., Bahmani, A., Alavi, A., Celli, A., Higgs, E., Dagan-Rosenfeld, O., Fay, B., Kirkpatrick, S., Kellogg, R., Gibson, M., Wang, T., Hunting, E. M., Mamic, P., Ganz, A. B., Rolnik, B., … Snyder, M. P. (2020). Pre-symptomatic detection of covid-19 from smartwatch Data. *Nature Biomedical Engineering, 4*(12), 1208–1220. https://doi.org/10.1038/s41551-020-00640-6

Appendix A

MATLAB code for Part A assignments

The code was posted at GitHub, which is available at
https://github.com/HyprValent/bmeg_321/blob/main/lab_1/lab_1_partA.m

Appendix B

MATLAB code for Part B, C, and D assignments

The code was posted at GitHub, which is available at https://github.com/HyprValent/bmeg_321/blob/main/lab_1/partBCD.m

Appendix C

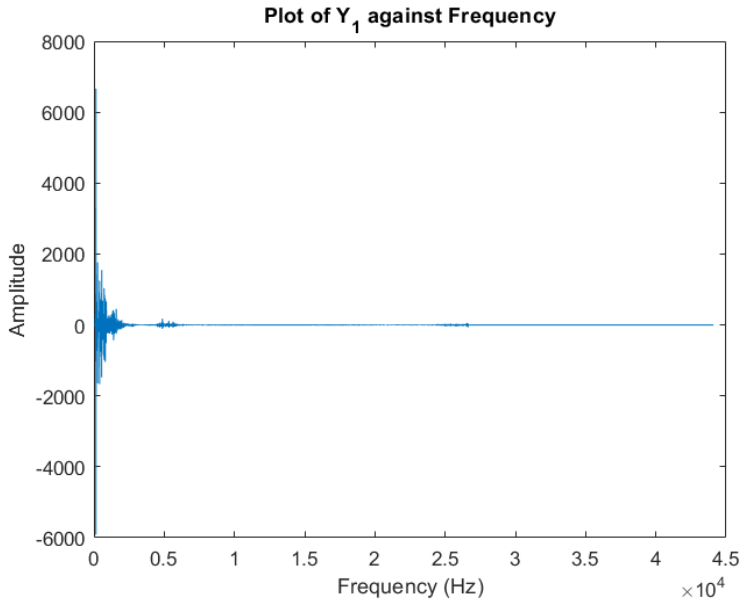
MATLAB code for Part E assignments

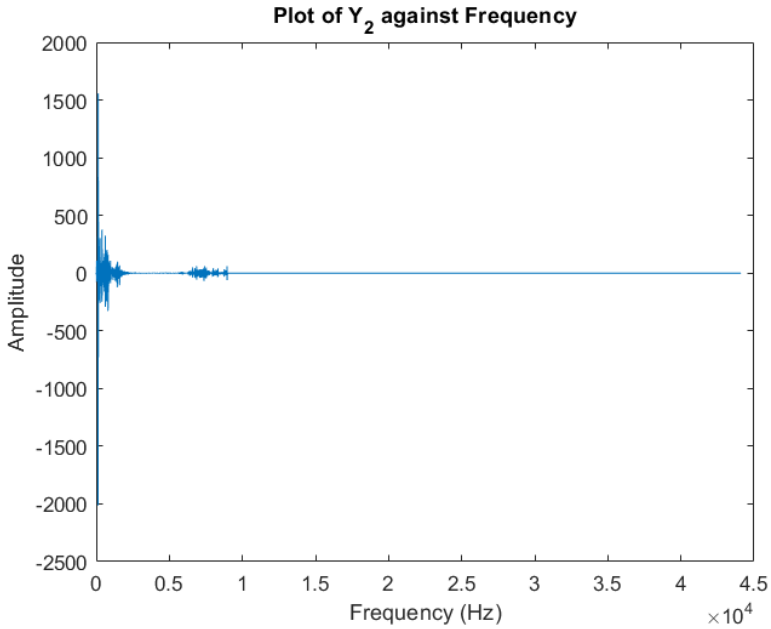
The code was posted at GitHub, which is available at
https://github.com/HyprValent/bmeg_321/blob/main/lab_1/partE.m

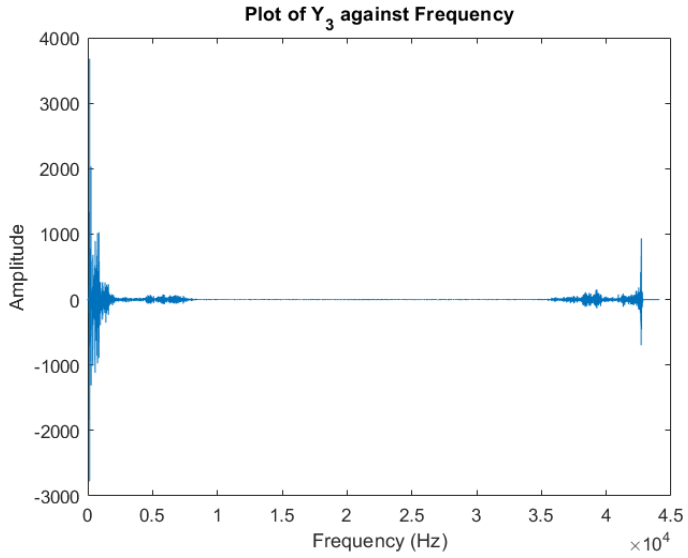
Appendix D

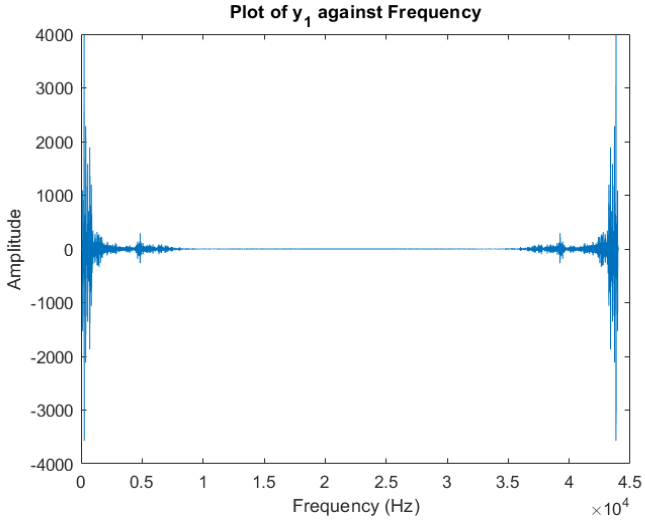
Comparison of methods to calculate FFT of convolved signal

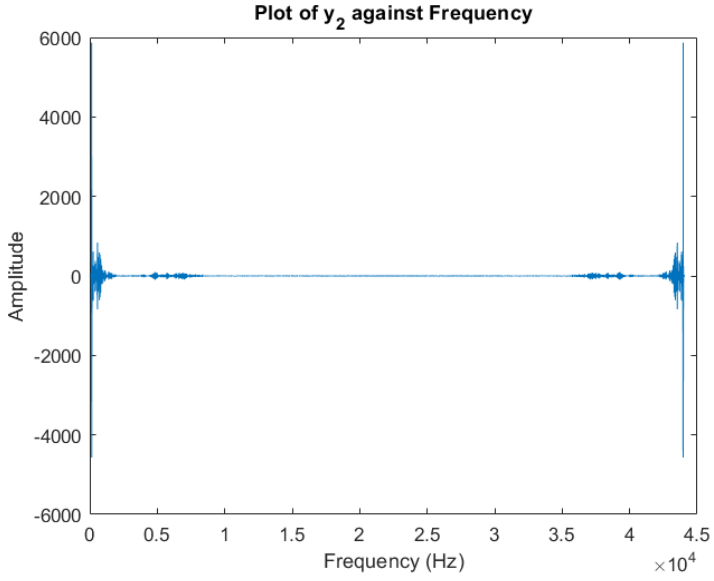
Since the task did not explicitly ask for graphs and these would aid in explaining the comparison, these have been added to the appendix instead. Capital Y (Y_1, Y_2, and Y_3) represent the FFT through matrix multiplication of the FFTs of the impulse response and the original signal. Lowercase y (y_1, y_2, and y_3) represent the FFT through utilizing the definition of FFT straight from the convolved signal.

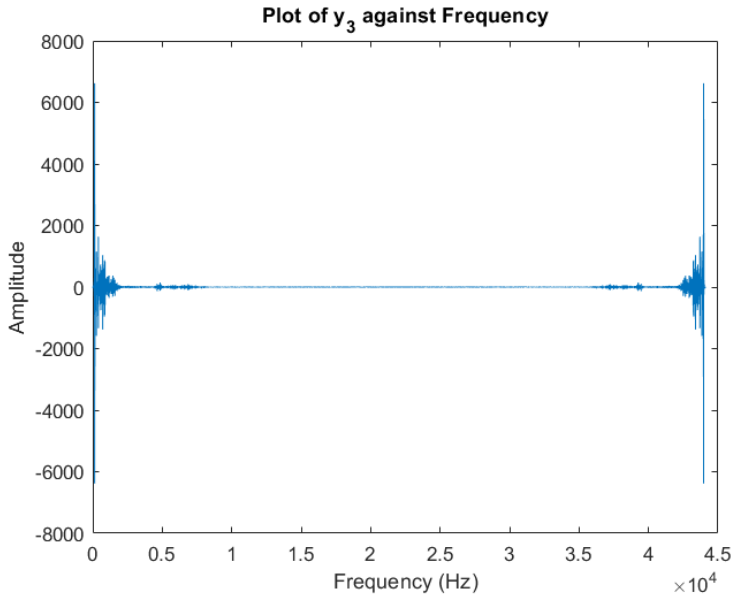





Plot of Y_3 against Frequency
Amplitude
Frequency (Hz)
$\times 10^4$


Plot of y_1 against Frequency
Amplitude
Frequency (Hz)
$\times 10^4$


Plot of y_2 against Frequency
Amplitude
Frequency (Hz)
$\times 10^4$


Plot of y3 against Frequency
8000
6000
4000
2000
0
-2000
-4000
-6000
-8000
Amplitude
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
Frequency (Hz)
×10^4