Ruiyan Guo 400256752
Xinyu Chen 400221680

Fourier Transform Take-home Exercise

For this part, the function scipy.signal.square(t, duty) is used to generate the square wave plots. In order to achieve the arbitrary duty cycles, the parameter 'duty' should be generated by the function random.uniform(0,1) which can output the float point number between 0 and 1. Since the time domain square wave with arbitrary duty cycle is plotted, the DFT function which is implemented in the previous in-lab part is taken advantage of to plot the frequency domain figure.

Filter Design Take-home Exercise

In this section, three different tones were generated, with frequencies equal to 2, 20, 40 respectively. To achieve the bandpass on this filter, inside firwin_coeff's function `signal.firwin, pass_zero='bandpass'` is added as one of the input, and the cutoff frequency for each side is set to 10 and 30. In this case, only the 20 Hz signal could pass.

Block Processing Take-home Exercise

For the filter coefficients, the build in `firwin` function is replaced by my own function `myCoeff(Fs, Fc, Ntaps)`, which is also used in the filterDesign part. The code for this function is refer to the reference code provided in the lab manual under the Digital Filter Design section.

To achieve the same functionality from `lfilter`, the build in function is replaced by my own function `mylfilter_w_block(coeff, data, size, buffer)`, with this function, block processing can be achieved. For the inputs of this function, 'coeff' is the filter coefficients derived from `myCoeff(Fs, Fc, Ntaps)`, 'data' is the piece of raw data that is required to be processed, 'size' is the block size, 'buffer' is the array storing the last few elements from each block. Inside this function, it convolve the filter coefficients with the input raw data, it is achieved by a nested for loop, where the outer loop loops through the raw data, the inner one loops through the coefficients. This function has two returns, '`buffer_out`' and '`filtered_data`'. 'buffer_out' is the buffer contains the last few elementa, 'filtered_data' stores the processed data.

To implement the `mylfilter_w_block` stated above, my own function `myBlockfilter_implementation` is writtened. It contains a while loop that break the audio data into blocks and assign them to the function `mylfilter_w_block` to process. This function returns '`filtered_data`', which contains the whole-piece processed data.

During the development process, there are challenges such as unexpected bugs and wrong output audio with cracks in it. To solve them, plotting was used, it helps to visualize the output rather than an audio only file.