August Byrne

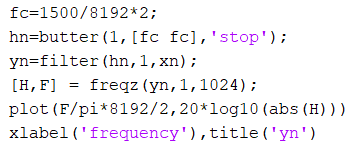
EECE 433

5/22/2021

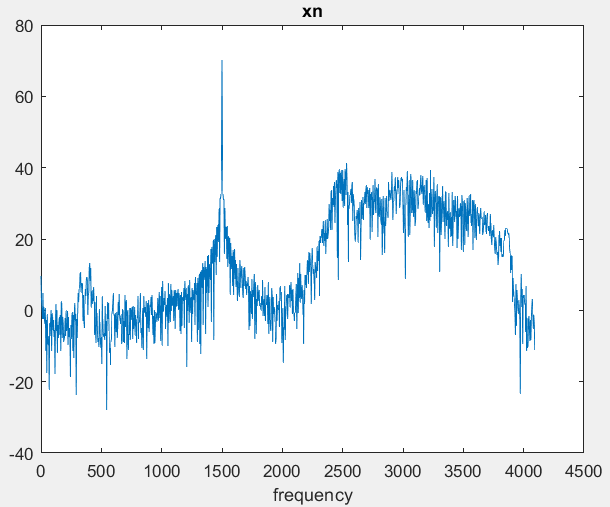
Lab 5

Part 1:

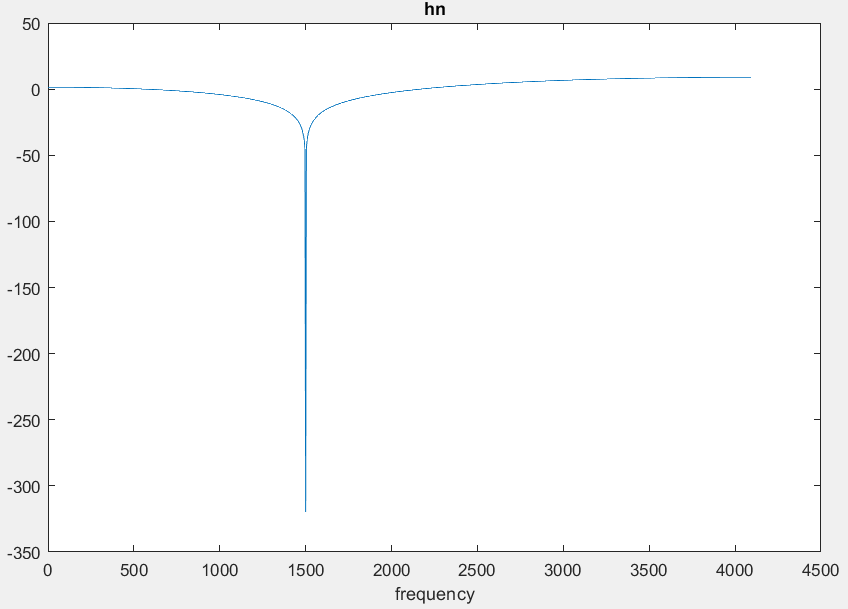
For this part of the lab, I used a 1st order Butterworth band stop filter which I designed in MATLAB using the butter function. I verified that the filter was working as intended using this matlab code



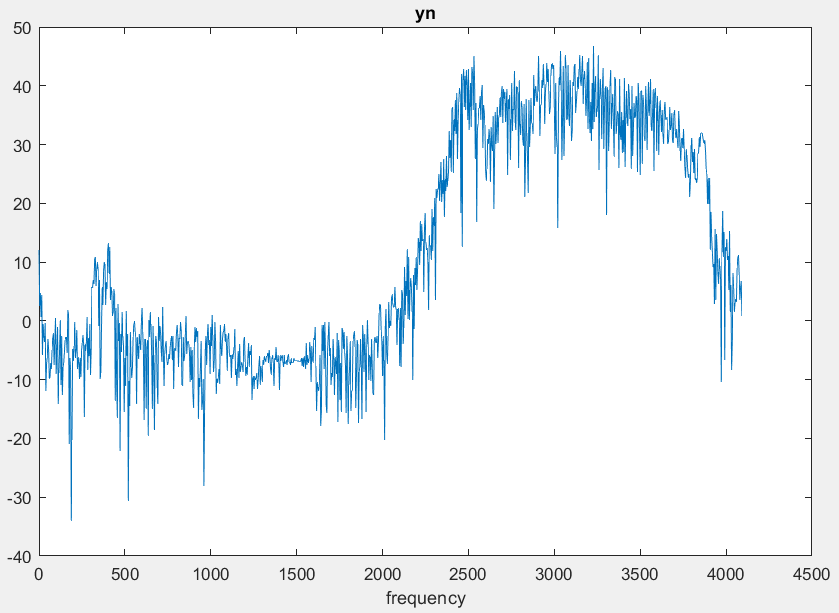
Through this code, I was able to verify that my filter was attenuating at the correct frequency, and the graphs below also helped to show that the filter was working correctly.



Xn (chirping with 1.5kHz noise)



Hn (1st order Butterworth band stop filter designed in MATLAB using the butter function)

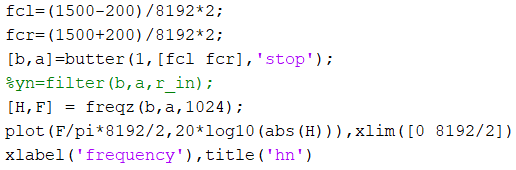


Yn (xn filtered with hn to remove the 1.5kHz noise)

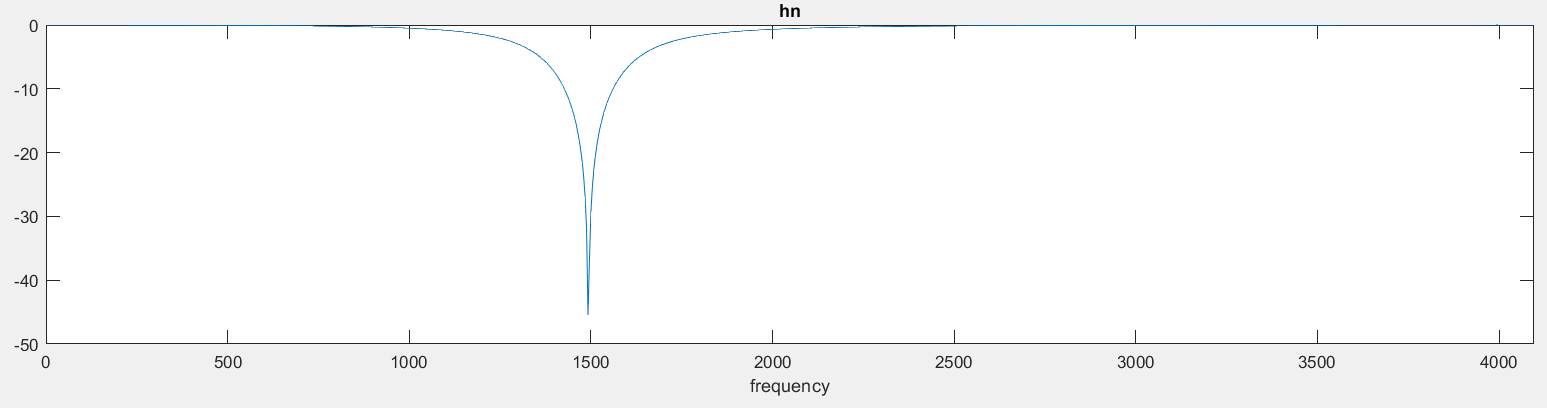
Part 2:

1. arm\_biquad\_cascade\_df1\_q31
2. arm\_biquad\_cascade\_df1\_f32
3. We would need arm\_biquad\_cascade\_df1\_init\_q31 and arm\_biquad\_cascade\_df1\_init\_f32 respectively. These are needed so that the system knows where the biquad has been instantiated, what the filter coefficients are, how many stages the biquad has, where it should store states (a pointer to the state buffer), and how much to shift the accumulator after completing calculations. Once the initialization function is given this data, it will then perform two main operations. It will set the values of the internal structure fields, and it will zero out the values in the state buffer.
4. You will need an instance of the biquad cascade structure (arm\_biquad\_casd\_df1\_inst), a state buffer, the number of stages, a post shift value, and an array of the coefficients.

Part 3:

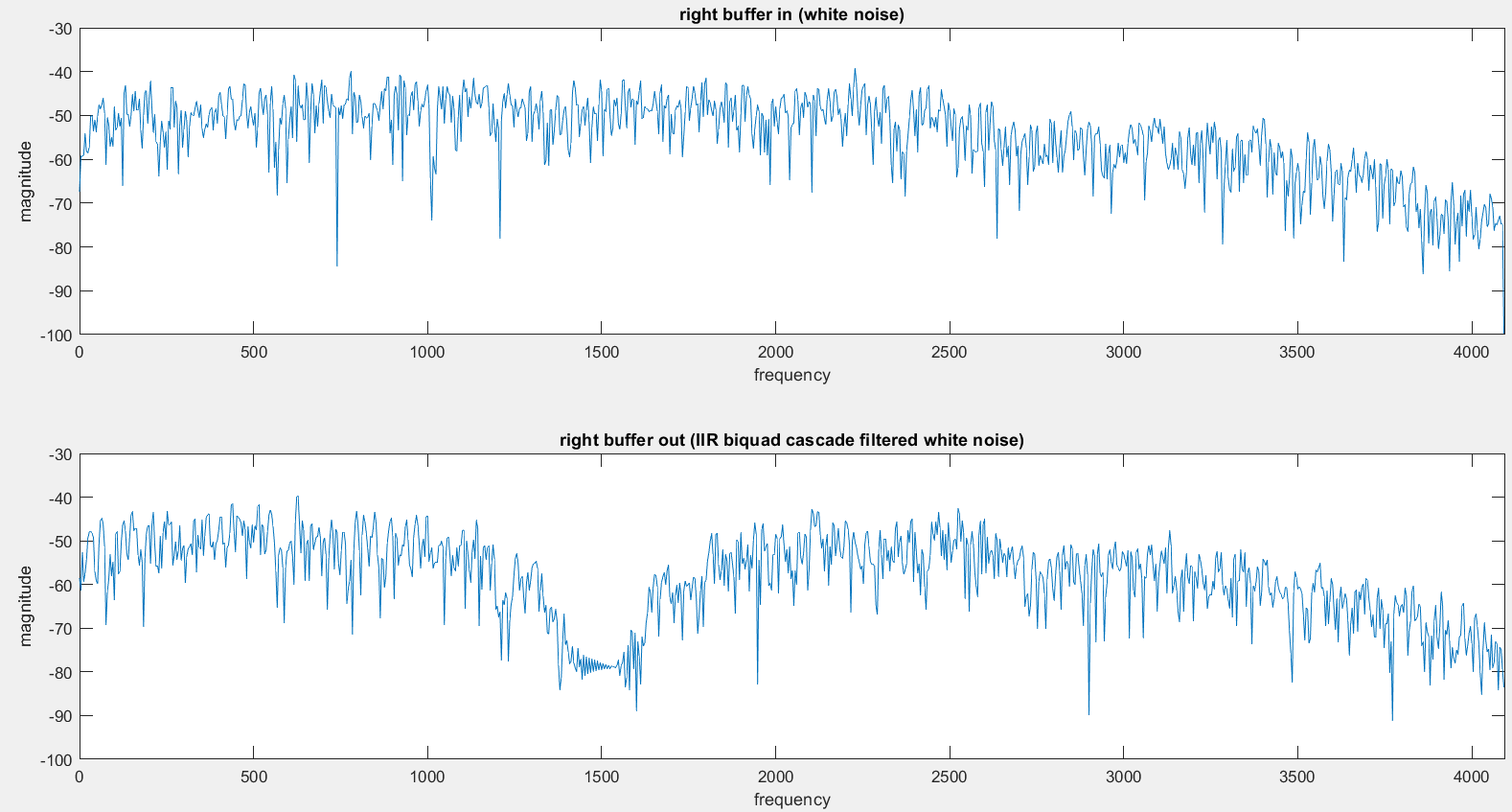
For this part of the lab, I generated my coefficients in MATLAB through the following code

This code uses butter to generate a first order band stop filter, with cutoffs at normalized values of 1.3kHz 1.7kHz. I check that the filter looks correct with freqz and plot, shown below



I then put it into MCU by putting the ‘b’ coefficients into an array first, and then, after removing the first ‘a’ value and negating all of the rest, putting the ‘a’ coefficients in the array as well. To add more stages, I just take that whole block, and copy-paste it into the array after the previous stage.

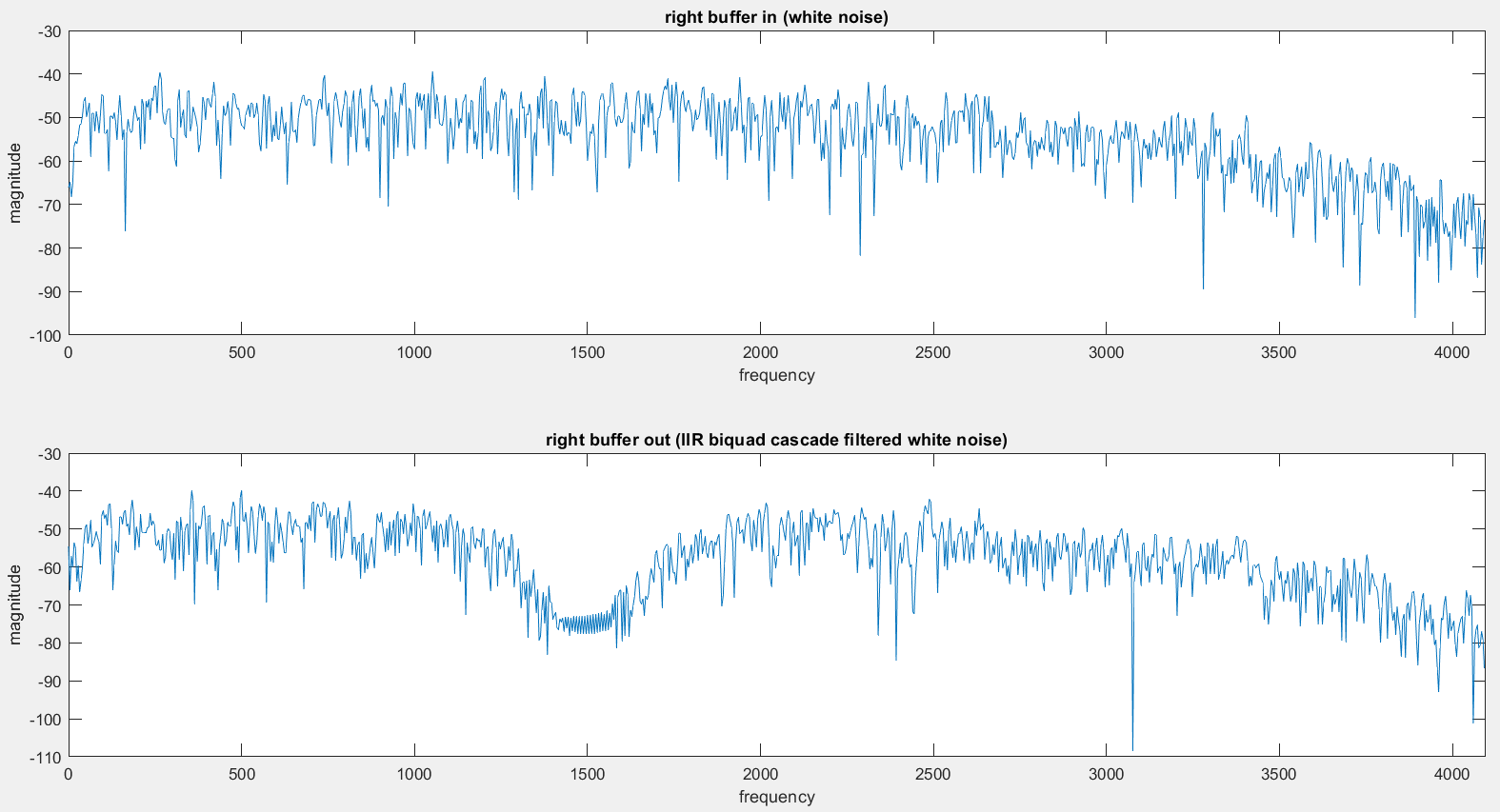
I verified that my filter was working by choosing white noise as my input signal, and then gathering the data of both the input and output buffers and comparing their graphs.



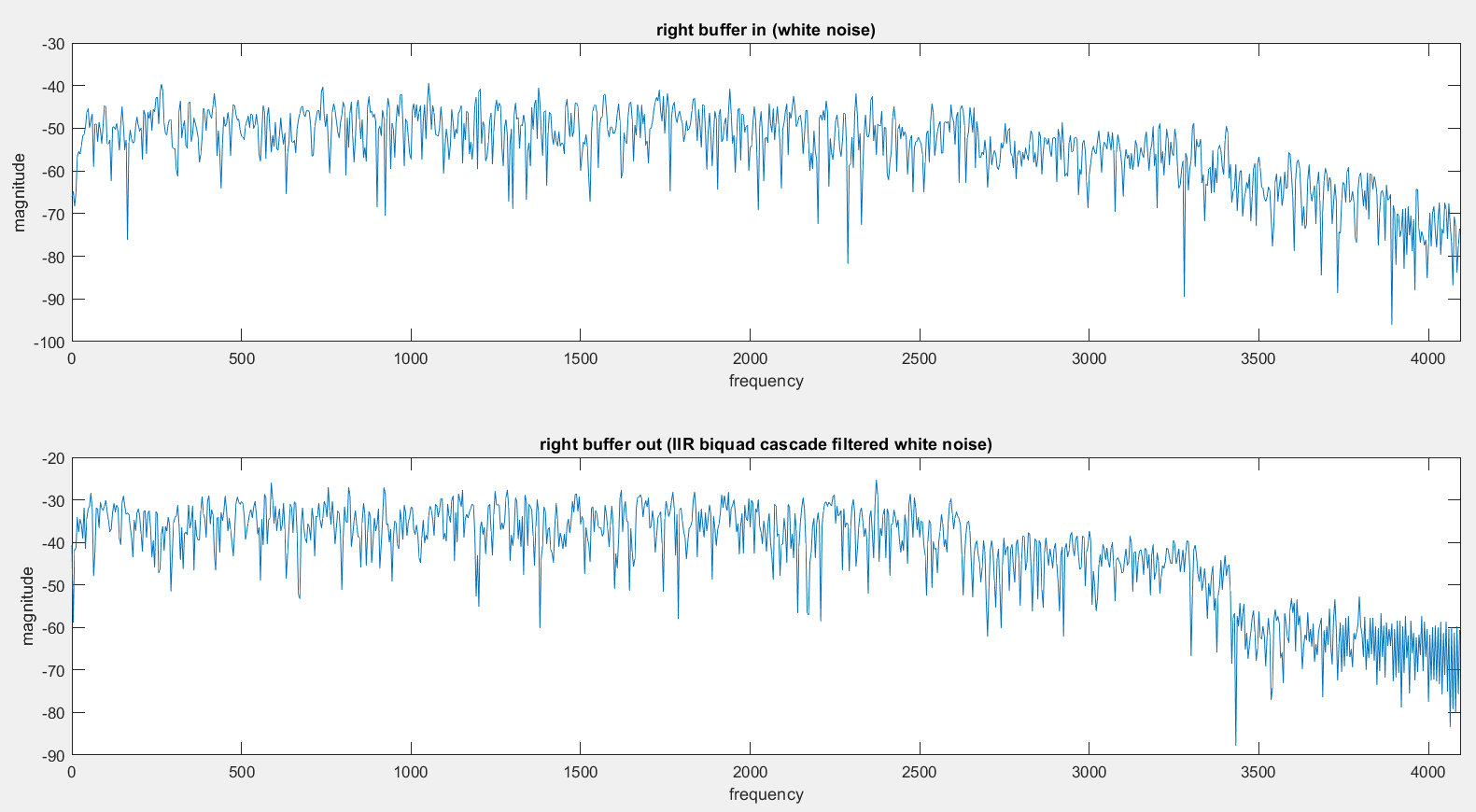
As we can see from the graphs above, the biquad cascade band stop filter is successful in stopping frequencies closely centered around 1.5kHz.

Part 4:

With the smallest input signal my phone can produce, and with filter coefficients that were between -1 to 1, my filter in Q31 format worked correctly



When I increased the input signal and tested it again, the filter continued to work correctly. I did not want to increase the input further than half of my phones output, so to get the bad filtering values from overflow, I scaled my filter coefficients up by 8. This worked, and I got the below graphs



From here, in order to get my filter to work again, I just had to scale this down. One way was to use the postScale of the biquad init function. I tried to use this and tried manually scaling down the scaled coefficients, but somehow the scaling function that I had used to scale it up seemed to make it hard to scale down to the correct values for me.