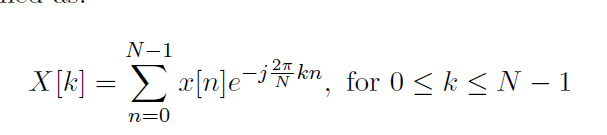
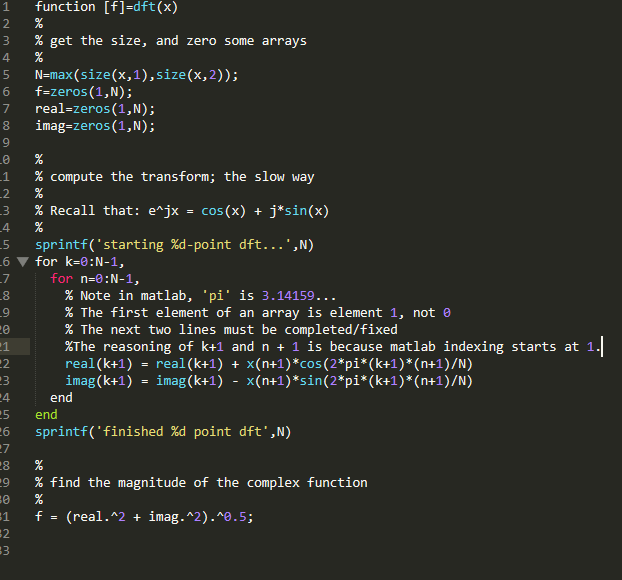
Frequency Analysis Portion:

N-Point DFT defined as: 

In speech, we only care about the magnitude of the spectrum.

There are a couple of lines marked by “???”, replace these lines to calculate the magnitude.

So the finished code-snippet looks like the following:



When running the script, both figures are equivalent, verifying that our FFT implementation is correct!

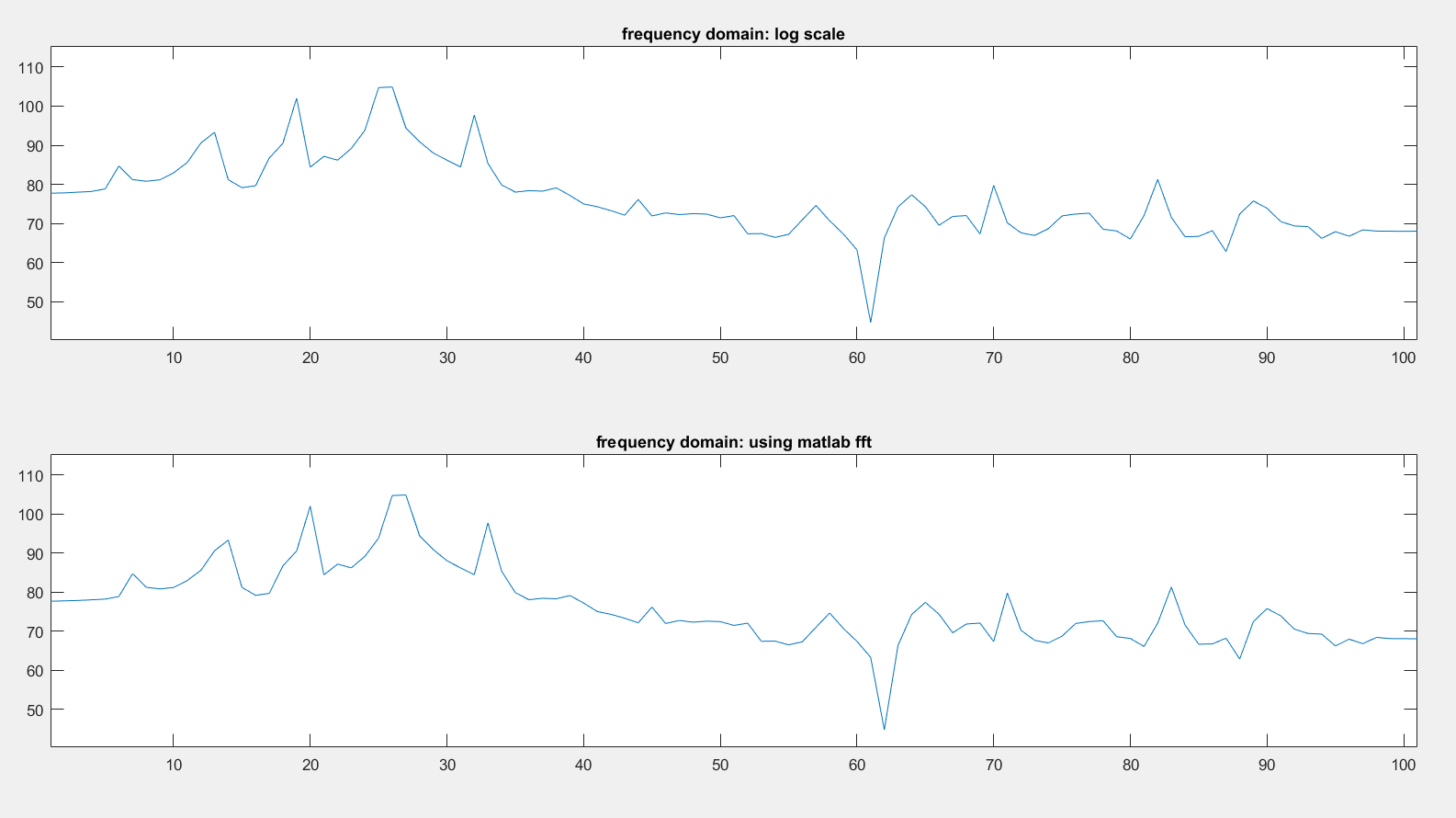


Figure 1: Verification of our DFT implementation. Both graphs are equivalent, so we wrote our script properly.

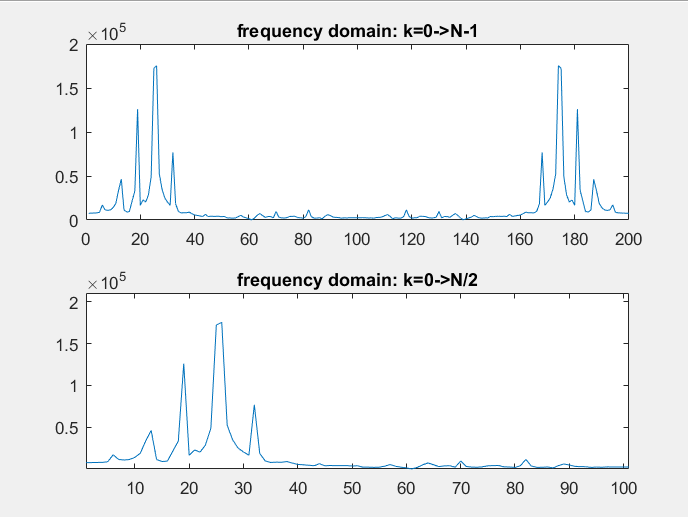


Figure 2: Output of second graph from *two.m*.

Executing one.m produced the following graphs:

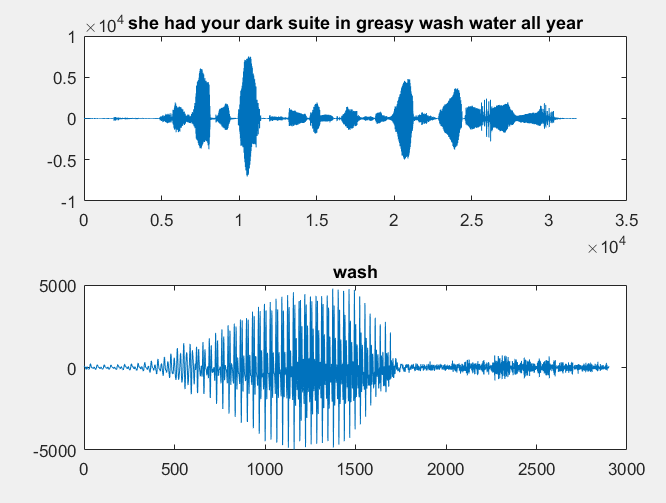


Figure 3: The amplitude spectrum of the sentence “She had your dark suit in greasy wash water all year” and “wash”

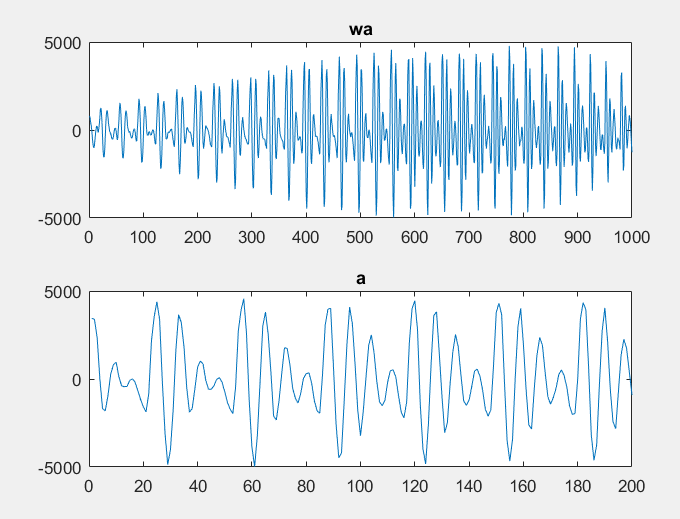


Figure 4: The sounds “wa” and “a” being voiced.

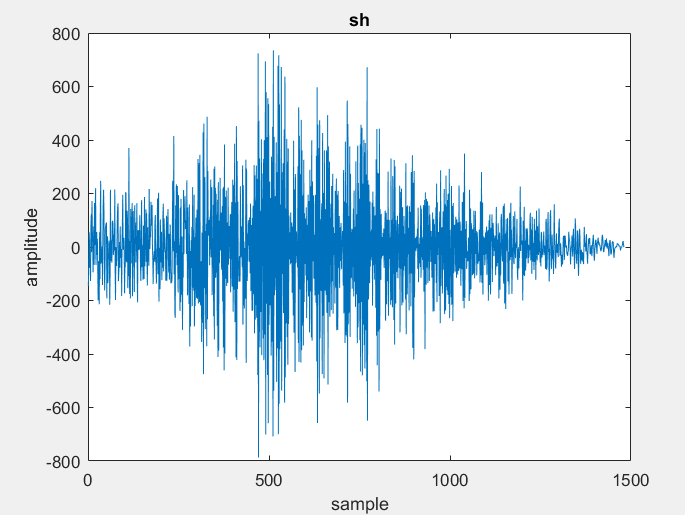


Figure 5: The /sh/ sound being voiced.

1. From figure 4 and figure 5, the phoneme’s /a/ and /sh/ have been plotted.
2. The added lines were the following: (See Code Above)
   1. %The reasoning of k+1 and n + 1 is because matlab indexing starts at 1.
   2. real(k+1) = real(k+1) + x(n+1)\*cos(2\*pi\*(k+1)\*(n+1)/N);
   3. imag(k+1) = imag(k+1) - x(n+1)\*sin(2\*pi\*(k+1)\*(n+1)/N);
3. The magnitude spectrum can be observed as part of Figure 2. It can be observed that the magnitude spectrum has symmetry across F = 3.14, since we model voice as being a real signal. (Input signal is real). Since we can assume the signal is real, then it follows the property that . The above graph is plotted as magnitude, however, so keep in mind that in reality this ends up being |X(w)| =
4. As can be seen in the second graph of figure 4, N is equal to 200. The resulting spacing, then, can be calculated as the following:
5. It’s known from 113 that the algorithm complexity for the FFT is Nlog2(N). If the DFT method is N^2, then we can express this as a ratio:.
   1. For 256 points, this becomes: when plugging in N = 256

Part B:

1. With increased oversampling, we can observe a decrease in the spacing between frequencies.
2. If you zero-pad for infinity, then theoretically, you would be able to have a FFT with 0 spacing between frequency bins.

Part C: We are seeing the difference between a linear convolution and a circular convolution (conv vs cconv)

Recording the output of the commands in the project specification, we see a huge difference between the circular and linear convolution:

(Where x previously defined as: x = [4, -1, 2])

conv(x,x)

ans = 16 -8 17 -4 4

cconv(x,x,3)

ans = 12 -4 17

Evidently, using N = 3, we get a time-aliased version of our signal. The minimum size N of the circular convolution to produce the same result as the linear convolution is N = 5.

Specifically, for the circular convolution to be equivalent, you must pad x with zeros to length at least N + L – 1. N and L are supposed to be the size of the first and second vectors of a convolution, but in our example, N = L.

cconv(x,x,5)

ans = 16.0000 -8.0000 17.0000 -4.0000 4.0000

Calculating the convolution in the frequency domain, we get the following:

>> y=ifft(fft(x).\*fft(x))

y = 12 -4 17

This evidently isn’t the same as the vector we expect, [16, -8, 17, -4, 4]. This is because this is time-aliased for the same reason above: the x-vector isn’t zero-padded.

Zero-padding the x-vector as so: x0 = [4, -1, 2, 0, 0] produces the desired result of:

y=ifft(fft(x0).\*fft(x0))

y =16.0000 -8.0000 17.0000 -4.0000 4.0000

Which is the expected result when the circular convolution is calculated properly with zero-padding or the correct N-value, and the expected result of the linear convolution! However, this can’t be done without zero-padding in the frequency spectrum.