COMPENG 4TL4 - Lab 4 Report: Discrete Fourier Transform (DFT) and an Application

Student Information

Student Name	Student Number
Kevin Le	400385350
Jonathan D'Souza	400304744

Lab Date: Nov. 10, 2025

Due Date: Nov. 21, 2025

Experiments

1. Introduction to the DFT

1(b) Plot of the DFTs of 5 Rectangular Signals of Different Lengths *N*

The following plots are created by computing the DFT of 5 rectangular signals of width 16 but with varying length *N* from zero-padding.

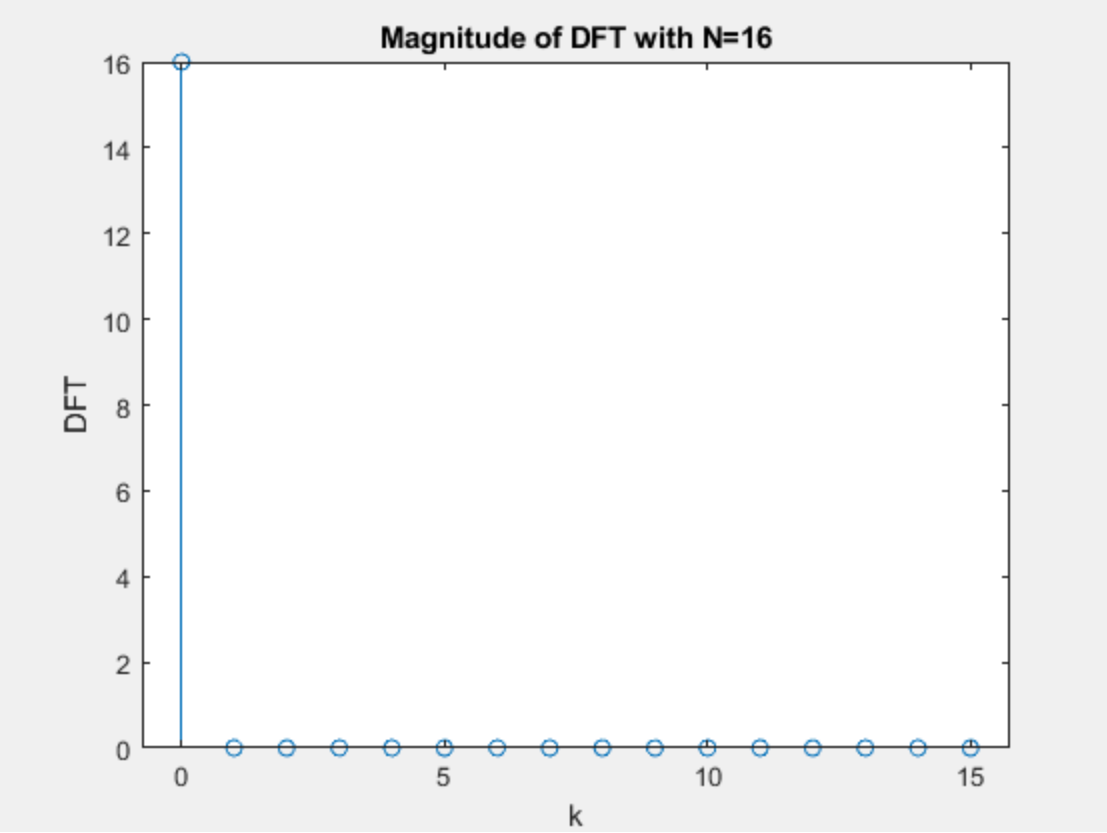

Figure 1: Magnitude of the DFT of rectangular signal with *N*=16

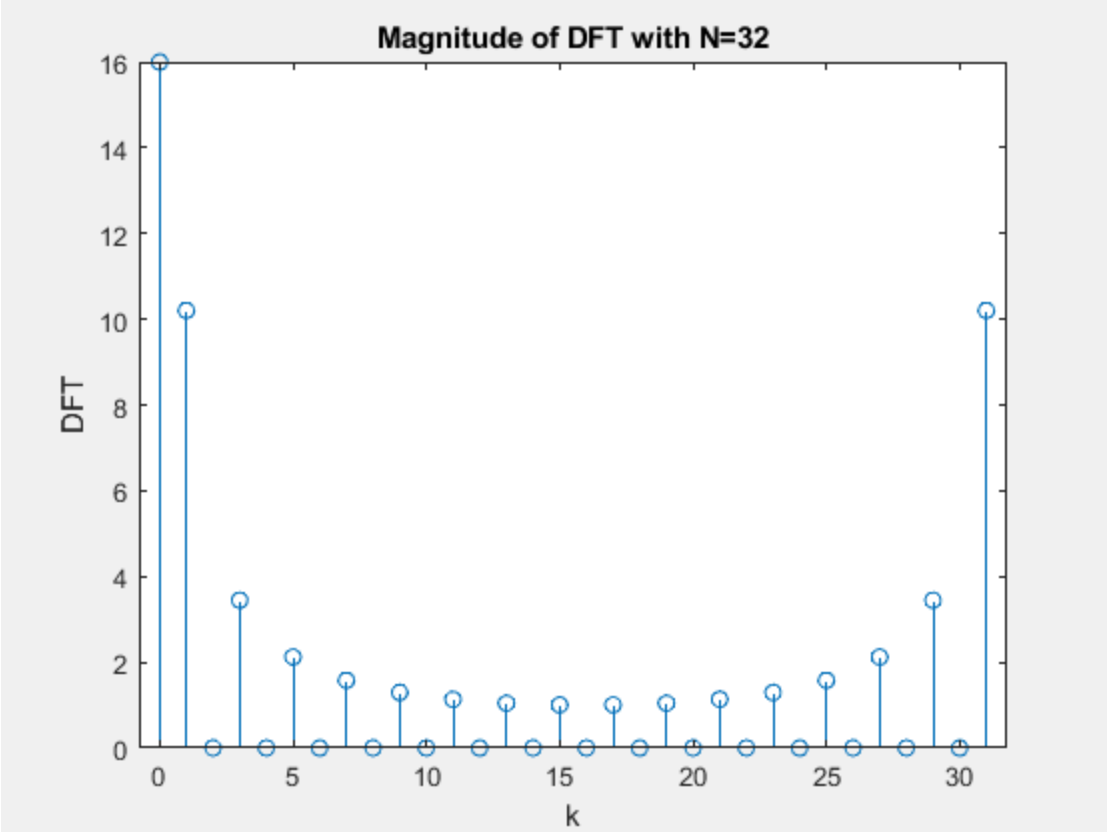

Figure 2: Magnitude of the DFT of rectangular signal with *N*=32

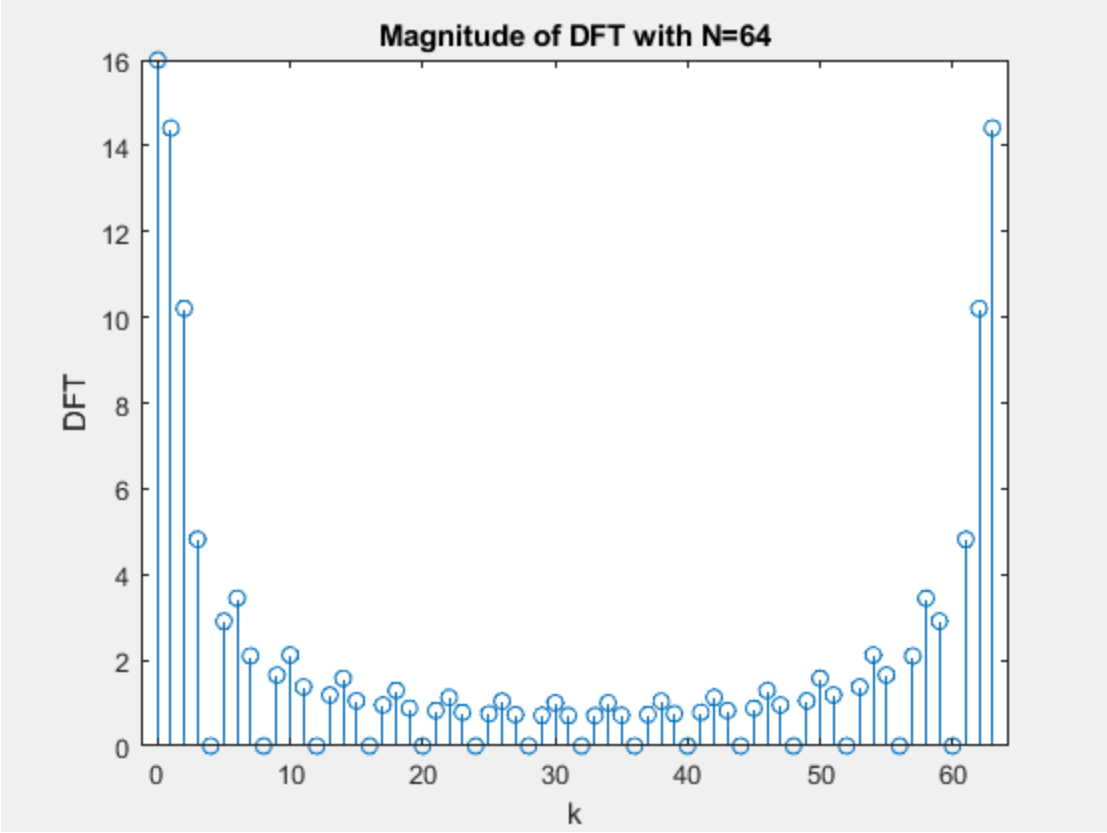

Figure 3: Magnitude of the DFT of rectangular signal with *N*=64

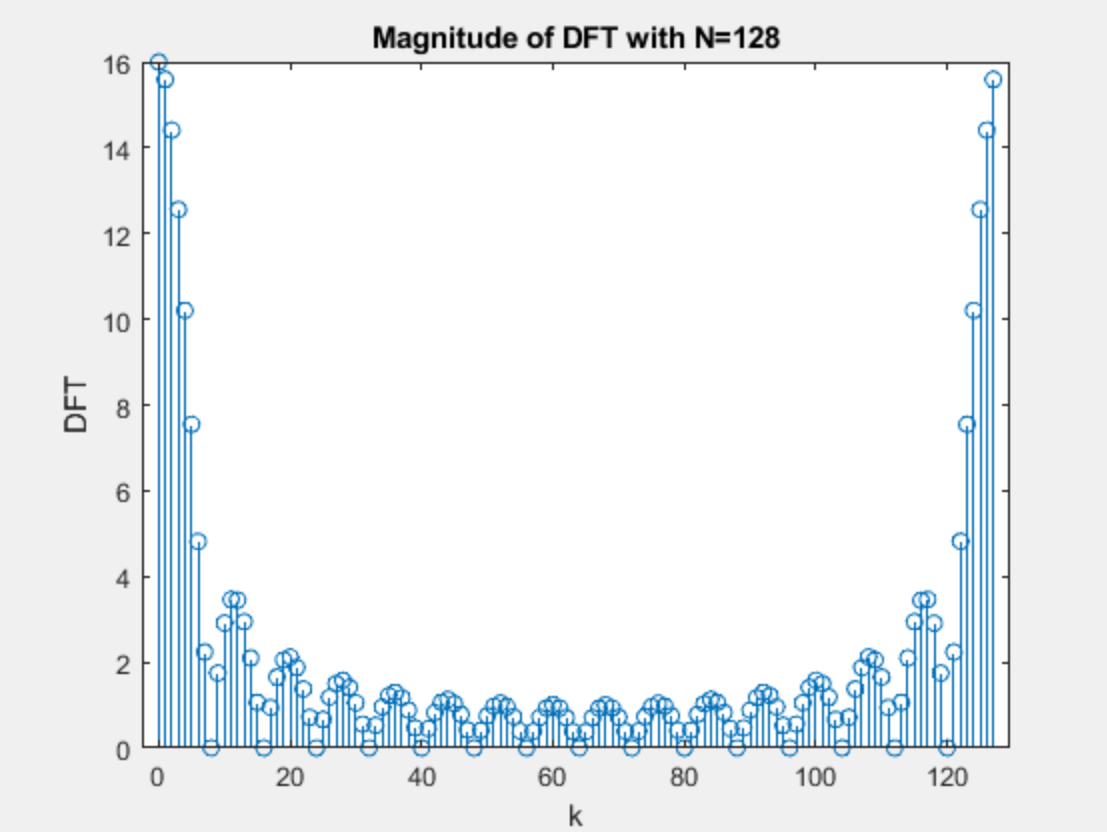

Figure 4: Magnitude of the DFT of rectangular signal with *N*=128

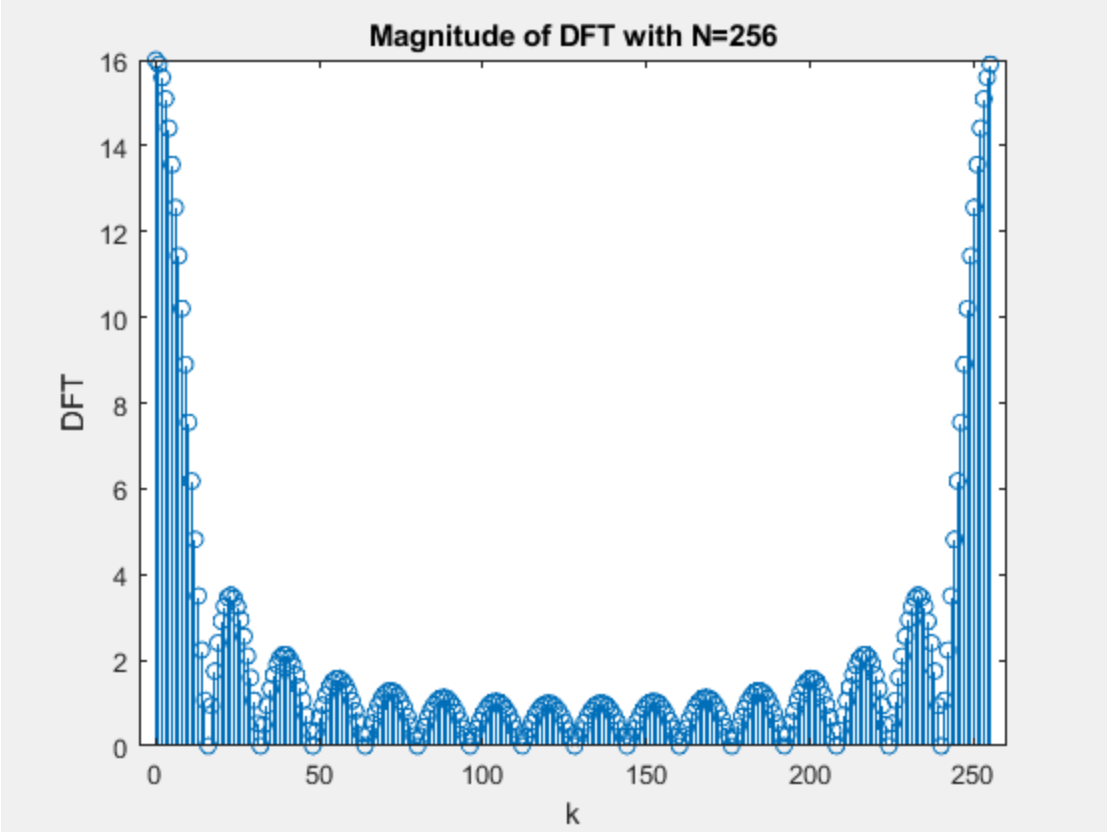

Figure 5: Magnitude of the DFT of rectangular signal with *N*=256

From figures 1-5, the frequency resolution can be observed improving as *N* increases from zero-padding. Everytime *N* doubles, it seems like the DFT plot receives an added data point between every existing point. It appears as if the frequency domain magnitude plot is being sampled with a higher sampling rate as *N* increases.

1(d) DTFT Input for DFT Equivalence

The following plots computes the frequency domain representation of the same rectangular signal of width 16, but with the DTFT function instead of the DFT function.

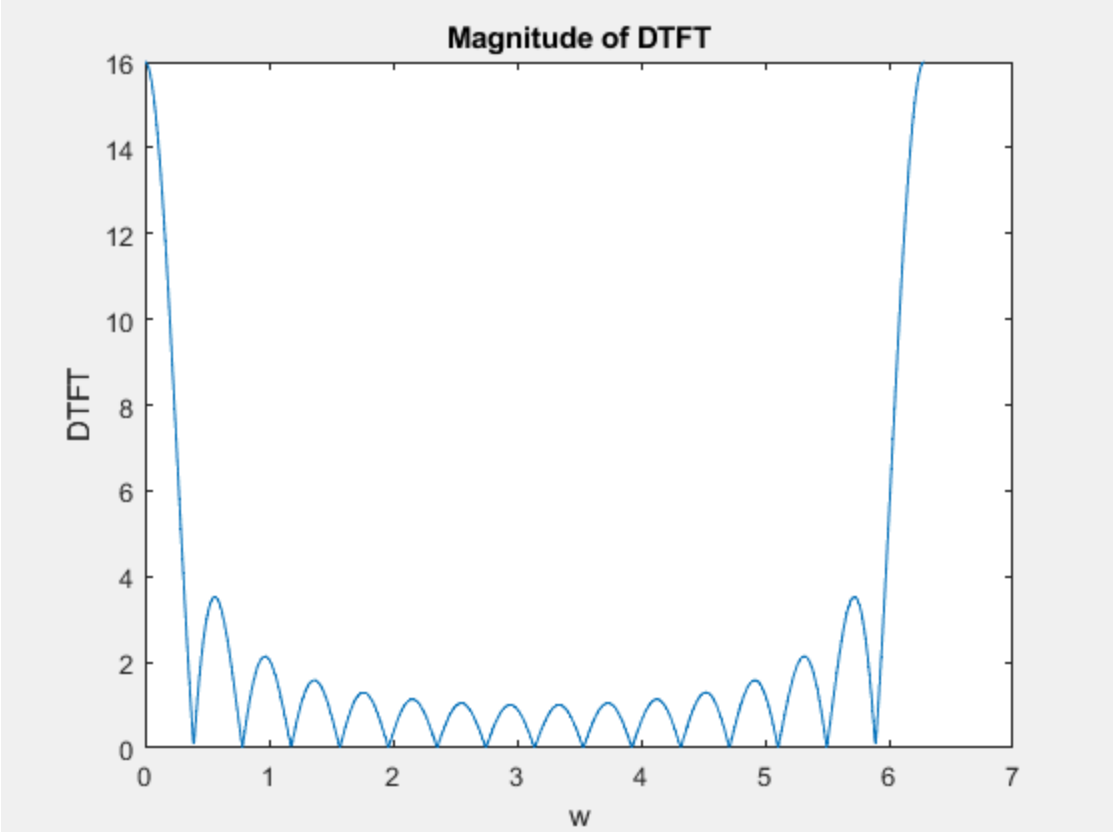

Figure 6: Magnitude of the DTFT of the same rectangular signal with a dense *w* vector from 0 to 2*pi

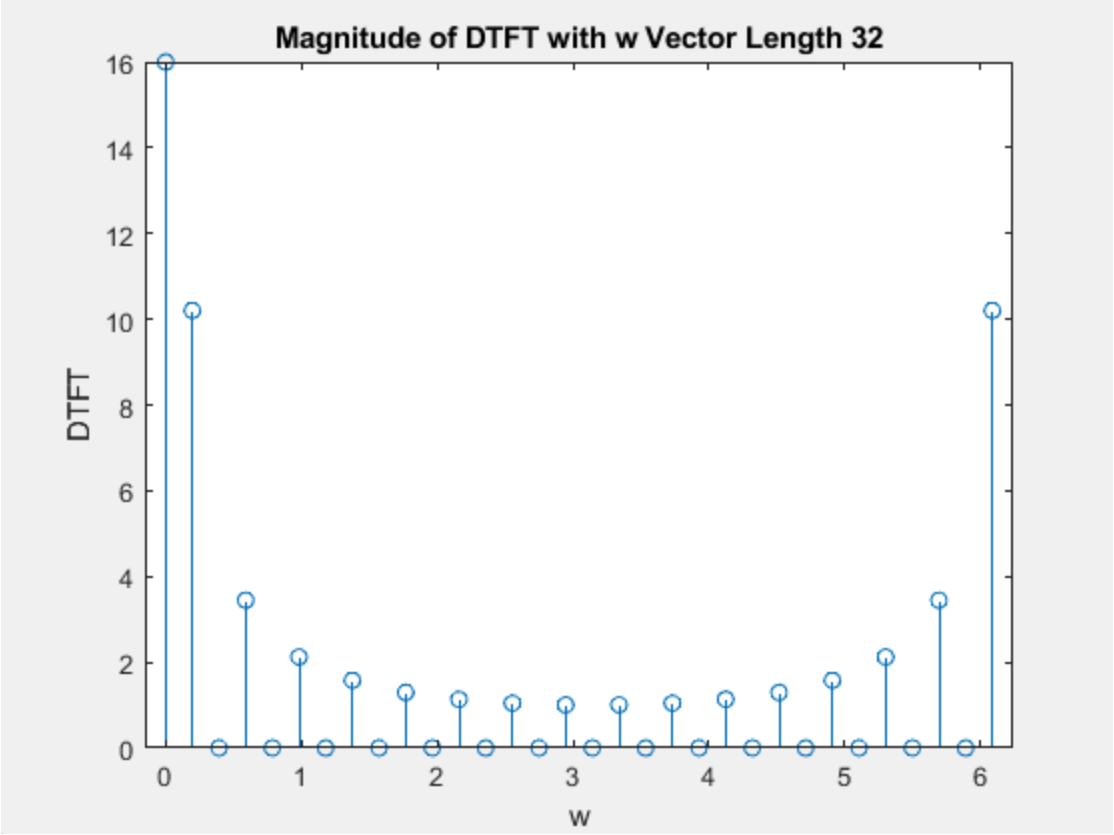

Figure 7: Magnitude of the DTFT of the same rectangular signal with a coarse *w* vector of length 32, and plotted with stem()

Figure 6 plots the magnitude of the DTFT using the plot() function where the *w* is a dense vector of length 1000 to approximate the continuous nature in the frequency domain when computing the DTFT.
Figure 7 plots the magnitude of the DTFT using the stem() function where the *w* vector has length 32 and in the range 0 to 2*pi*31/32. This plot is identical to Figure 2, where the DFT is computed on the input signal zero-padded to *N*=32.

The DFT and DTFT functions are able to return the same output in this case because the DTFT function implementation isn't a true DTFT where the frequency domain output is continuous. However, it allows the vector *w* to be created freely such that it can be dense to approximate a continuous signal or coarse to which the DTFT will resemble the DFT. When the DTFT is computed with a vector *w* of size equal to *N* in the range 0 to 2*pi*(*N*-1)/*N*, it is equivalent to the DFT where the input is zero-padded to size *N*. This is because when the DFT's *k* vector equals [0, 1, ... *N*-1], the phase shift terms are equivalent to [0, 2*pi*1/*N*, ..., 2*pi*(*N*-1)/*N*] and this is exactly equal to the vector *w*.

1(e) Computing DFT with fft() and Comparison of Fourier Transform Methods

The following plots are created by computing the DFT of the same 5 rectangular signals, however, the DFT is computed using MATLAB's built-in fft() function.

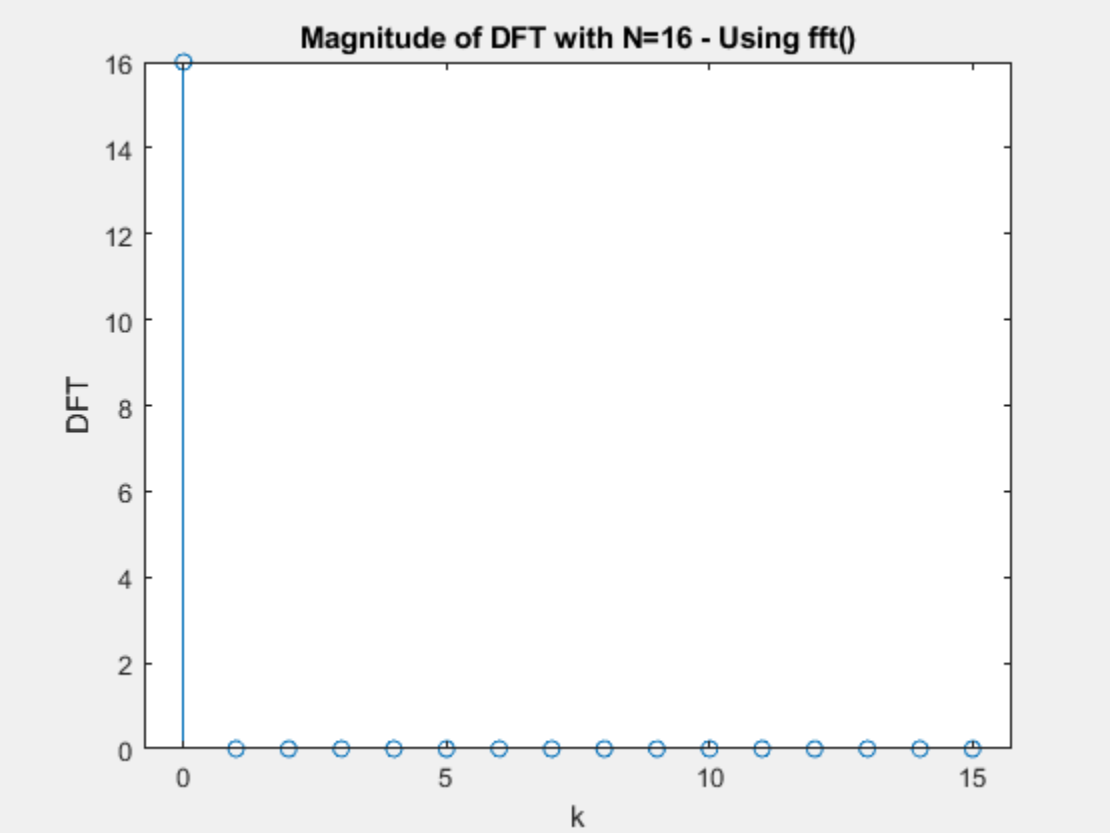

Figure 8: Magnitude of the DFT of the same rectangular signal with *N*=16 and computed using the fft() function

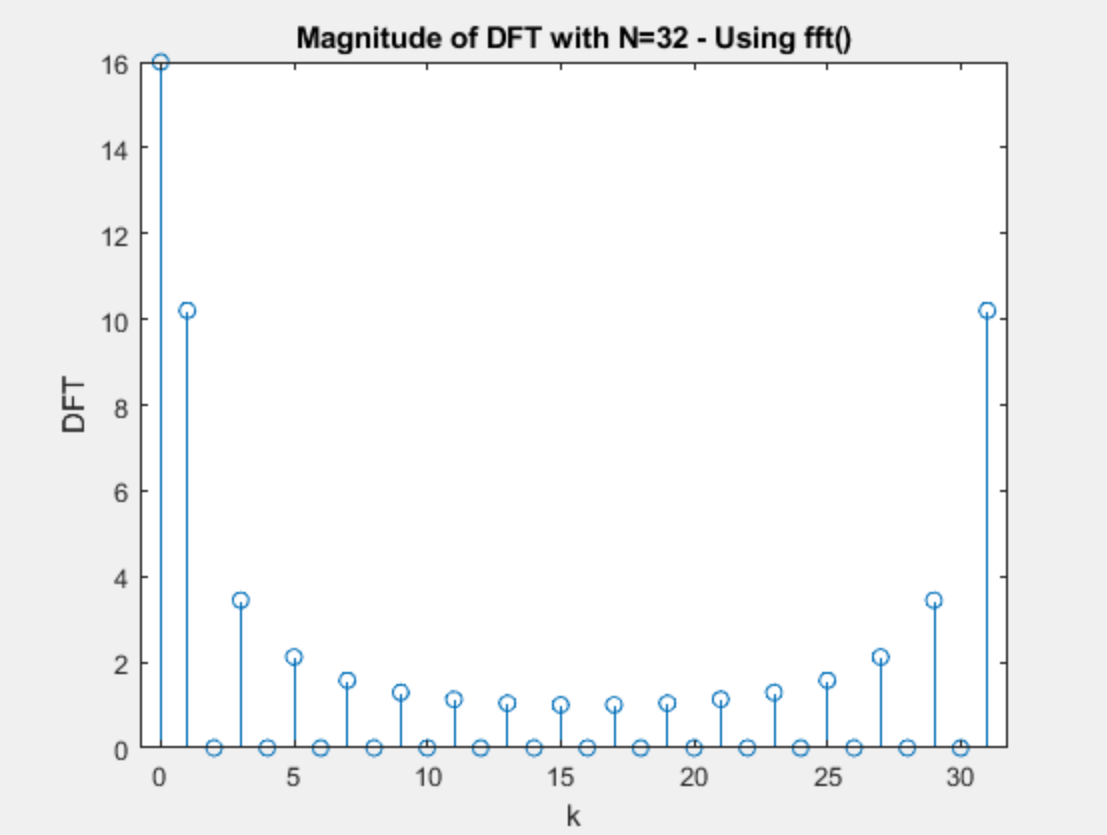

Figure 9: Magnitude of the DFT of the same rectangular signal with *N*=32 and computed using the fft() function

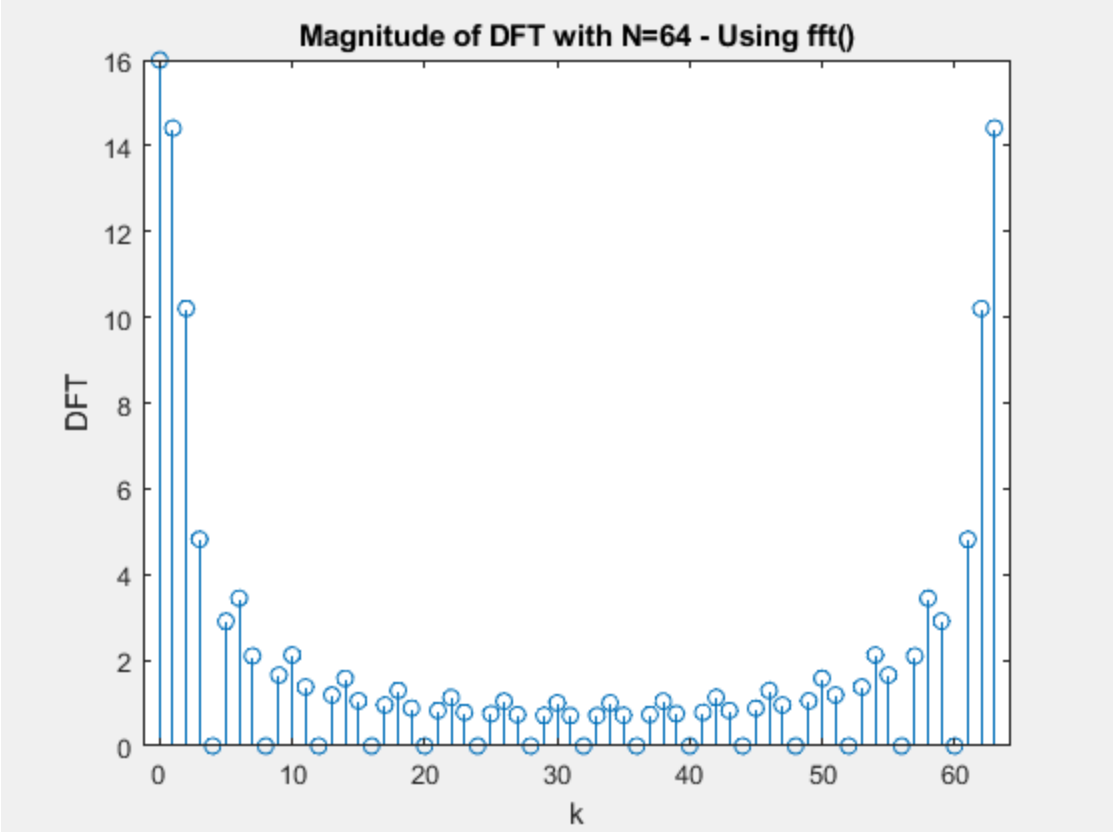

Figure 10: Magnitude of the DFT of the same rectangular signal with *N*=64 and computed using the fft() function

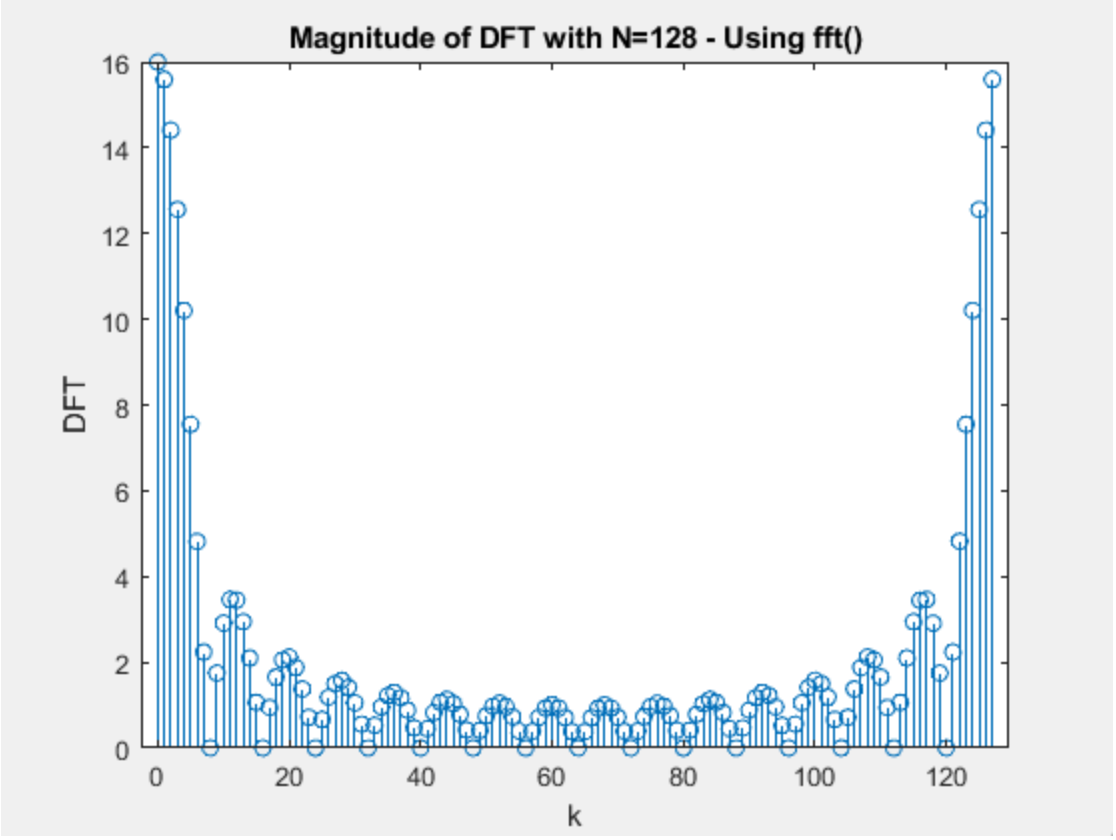

Figure 11: Magnitude of the DFT of the same rectangular signal with *N*=128 and computed using the fft() function

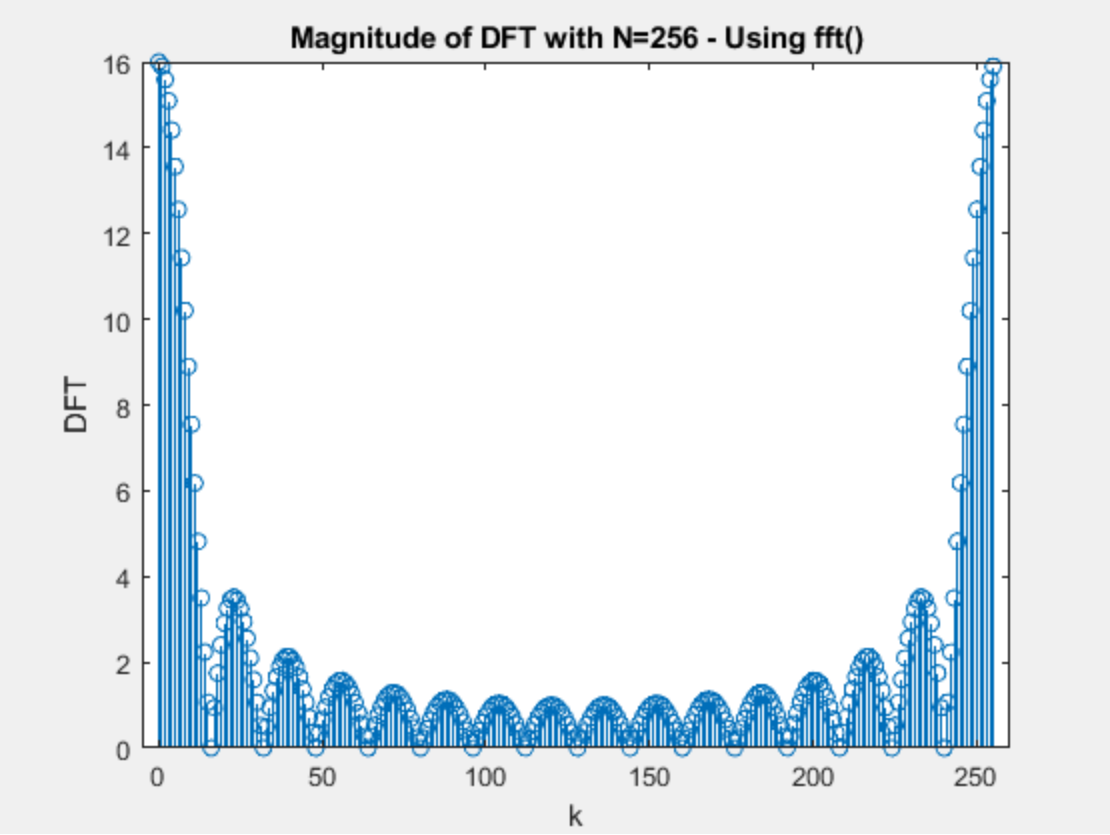

Figure 12: Magnitude of the DFT of the same rectangular signal with *N*=256 and computed using the fft() function

Figures 8-12 plot the magnitude of the DFT of a rectangular signal of width 16 and zero-padded to lengths 16, 32, 64, 128 and 256. The DFT for these plots were computed using the built-in fft() function rather than the custom DFT implementation. These figures can be observed to be identical to Figures 1-5 where the DFT of the same input signals are computed with the custom DFT implementation.

All three Fourier transform methods implemented so far are able to produce the same DFT results if the DTFT implementation has its *w* vector created in a specific way. However, the DTFT implementation is capable of changing the *w* vector such that the resulting frequency domain representation has a higher or lower frequency resolution. The DFT implementation and fft() can also achieve the same results but by increasing or decreasing the amount of zero-padding to the input signal. Although not tested in this section, the fft() is expected to have a lower runtime for increasing *N* due to the implementation having a more efficient algorithm to compute the DFT. The time complexity of the fft() implementation is O(*N* log *N*) while the time complexity of the regular DFT implementation is O(*N*^2).

2. DFT Resolution

2(a) Loading the aaa.wav file

```
Fs =

        8000
```

Figure 13: MATLAB command window output showing the sampling frequency

2(b)Plotting the Time-Domain Waveform

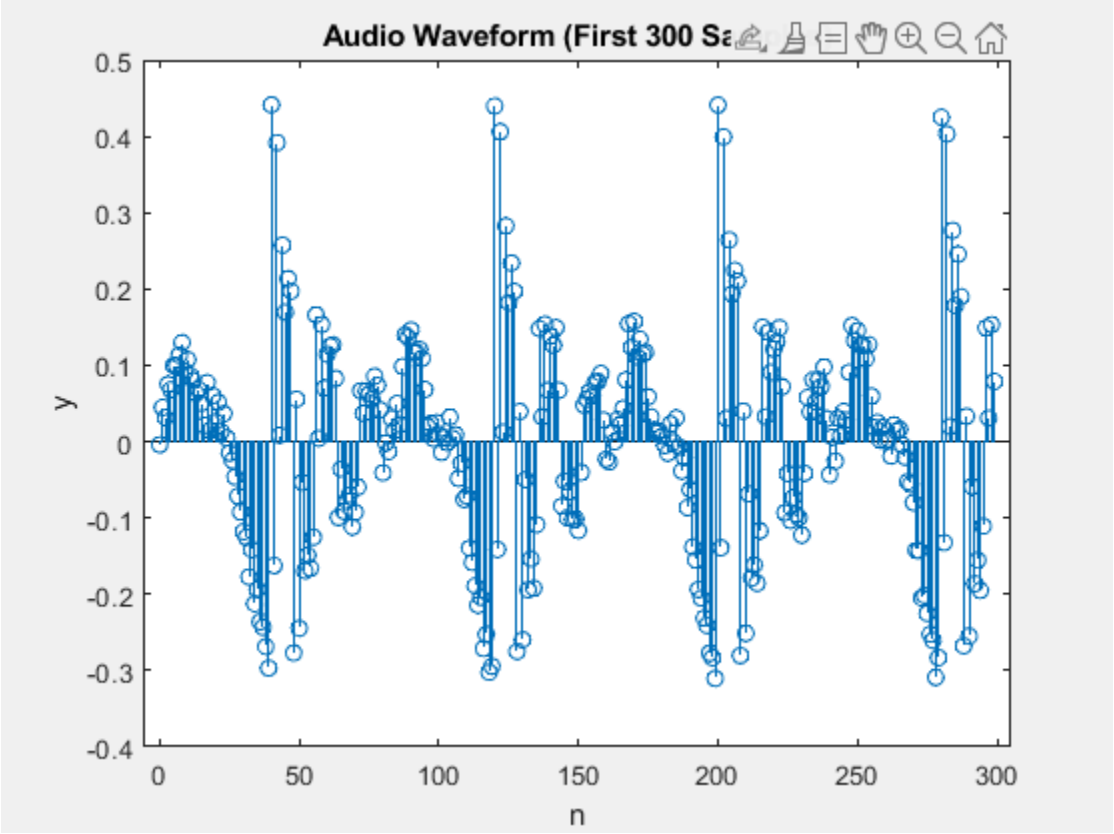

Figure 14: Time-domain waveform of the first 300 samples of the aaa.wav audio signal

Figure 14 plots the first 300 samples in the time-domain signal loaded from the aaa.wav audio file. The waveform can be seen to be periodic, with a period of 80 samples, which is equivalent to 0.01 seconds.

2(c) Magnitude of the DFT of One vs. Two Natural Periods

The following plots are created by computing the DFT of one natural period and two natural periods.

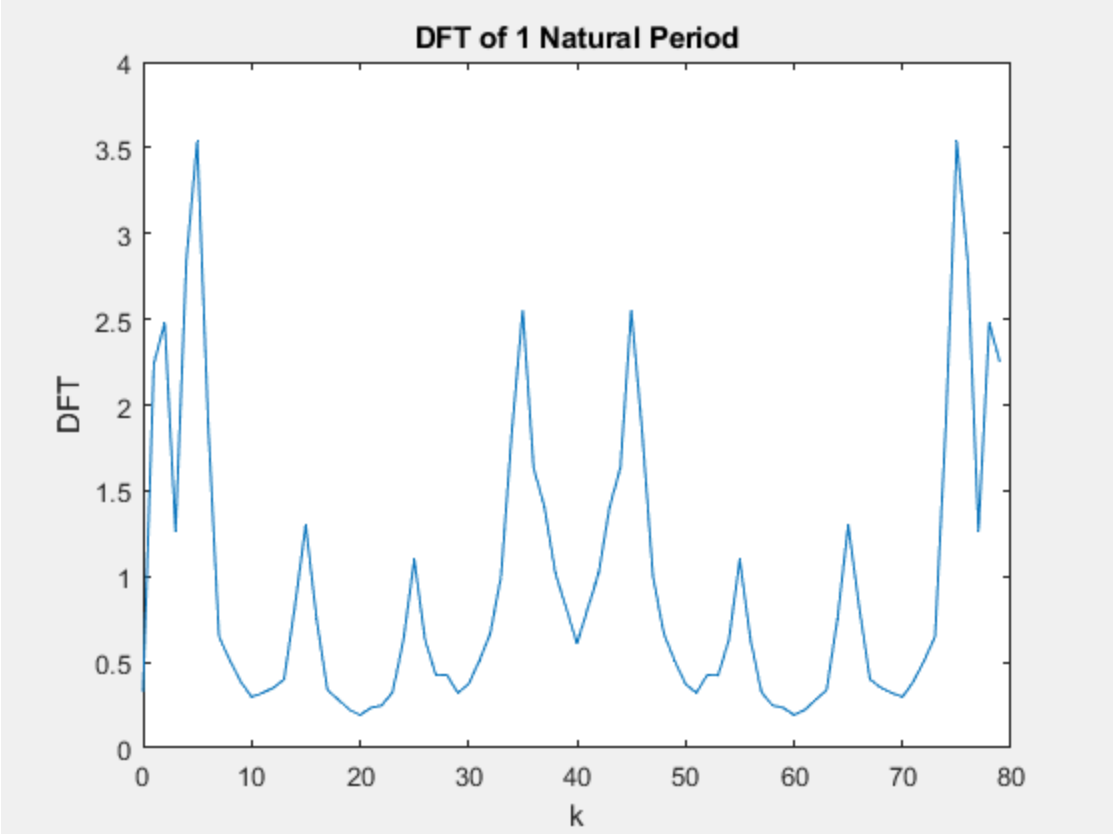

Figure 15: Magnitude of the DFT of one natural period of the aaa.wav audio signal

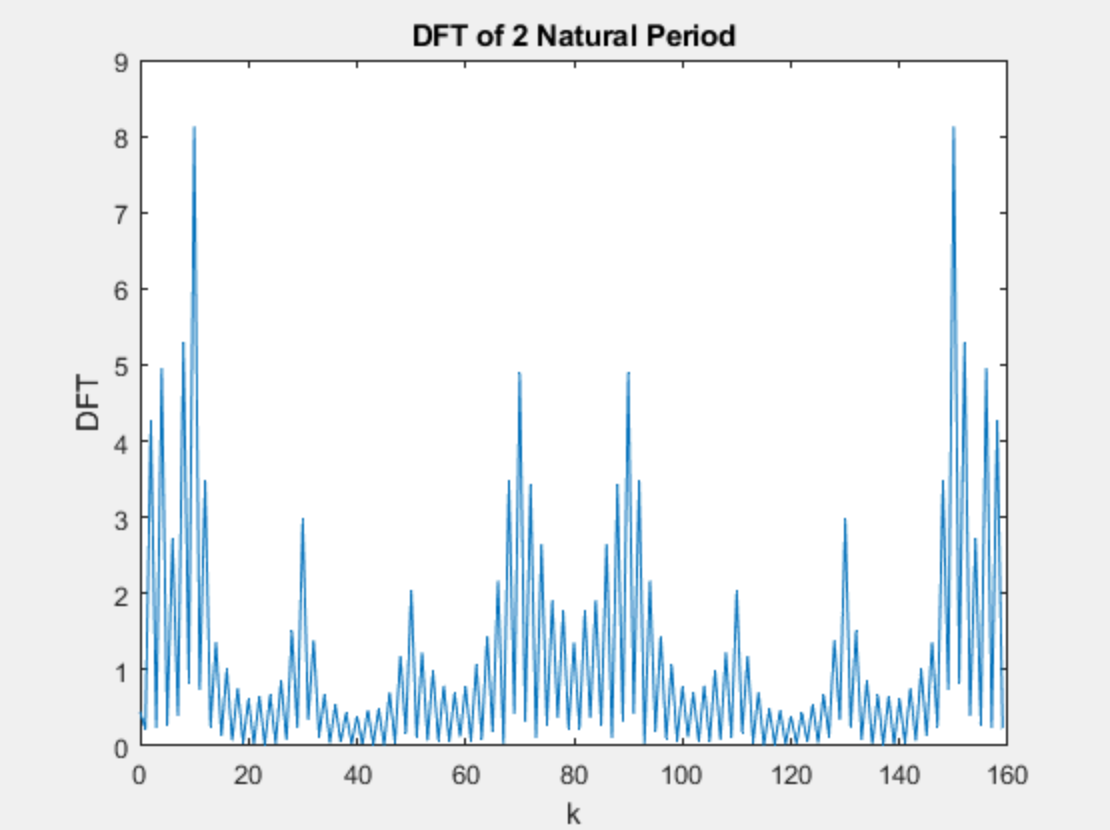

Figure 16: Magnitude of the DFT of two natural periods of the aaa.wav audio signal

Figure 15 plots the DFT of one natural period of the provided audio signal and Figure 16 plots the DFT of two natural periods of the provided audio signal. The DFT of two natural periods shows a signal with various peaks and contains many oscillations throughout the signal. The DFT of one natural period also captures these various peaks, however, it seems to only capture the envelope of the two-period DFT signal.

2(d) Effect of Zero-Padding on Frequency Sampling Resolution

The following plots are created by computing the DFT of the same input signals as Figures 15 and 16, but with added zero-padding to 1024 points.

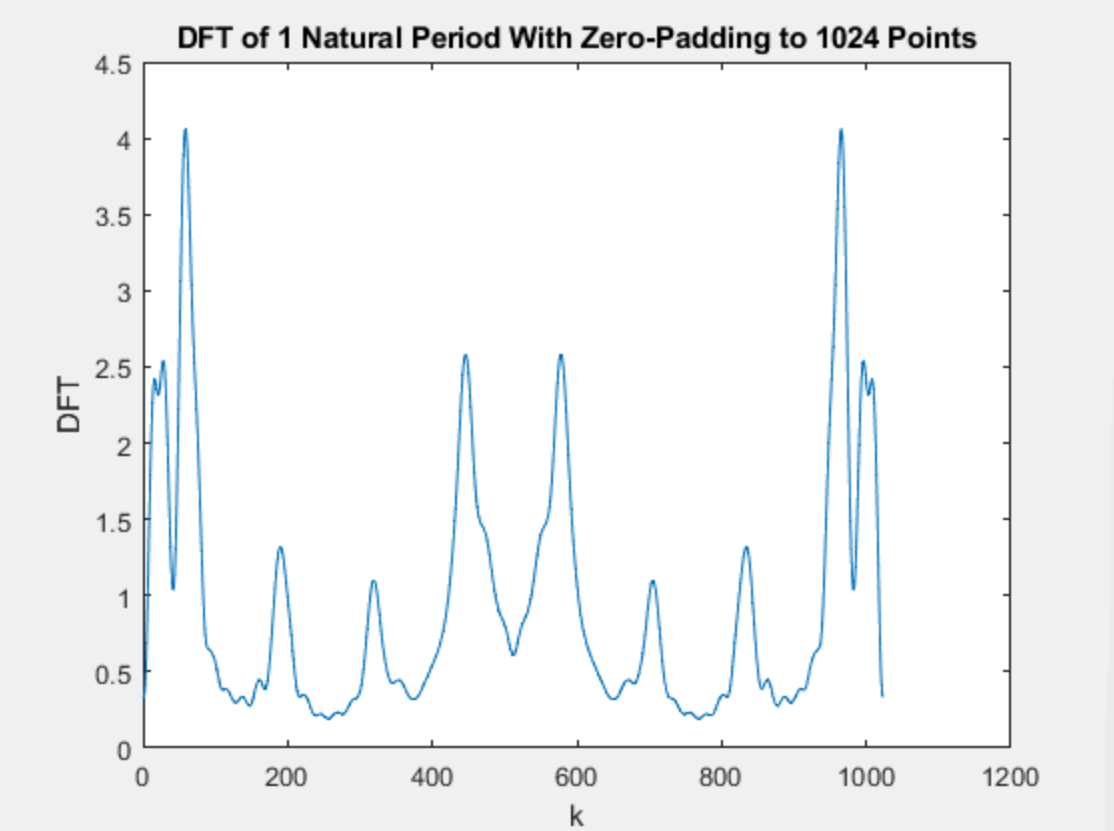

Figure 17: Magnitude of the DFT of the one natural-period signal zero-padded to 1024

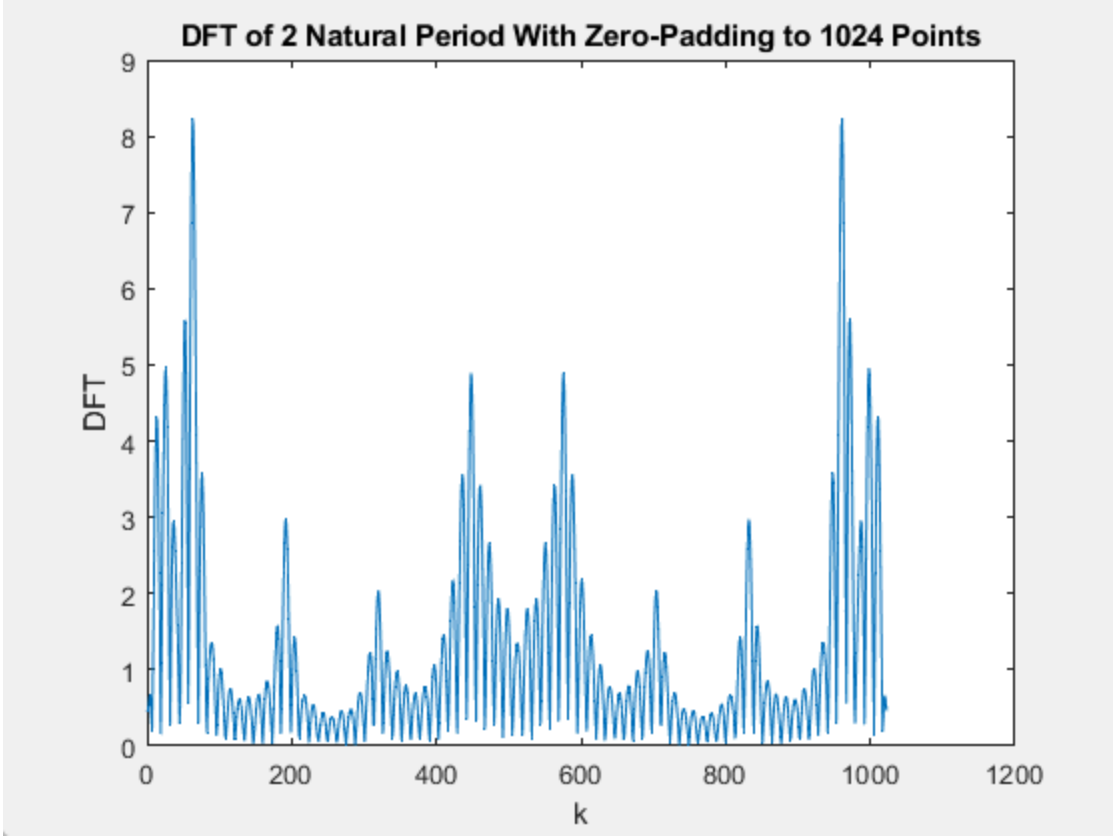

Figure 18: Magnitude of the DFT of the two natural-period signal zero-padded to 1024

Figures 17 and 18 plot the DFT of the same time-domain signal as Figures 15 and 16, however, the input signals have added zero-padding to N=1024.

We expect zero-padding to improve the frequency sampling resolution of the DFT signals, and this can be observed subtlety as the sharp spikes in Figures 15 and 16 now look more rounded and smoothened out in Figures 17 and 18, and Figure 17 sees a little more detail at the convexes of the plot. However, section 1 above saw the DFT changing dramatically with increasing N but the change in this section is almost negligible in comparison. This is because increasing N with zero-padding will only make the DFT closer to the DTFT but N without zero-padding is already enough to approximate the continuous DTFT waveform, so increasing N will only change the DFT minimally.

2(e) Effect of Windowing (Increasing M) on Spectral Resolution

The following plots are created by computing the DFT of the input audio signal windowed at 1-5 times the natural period.

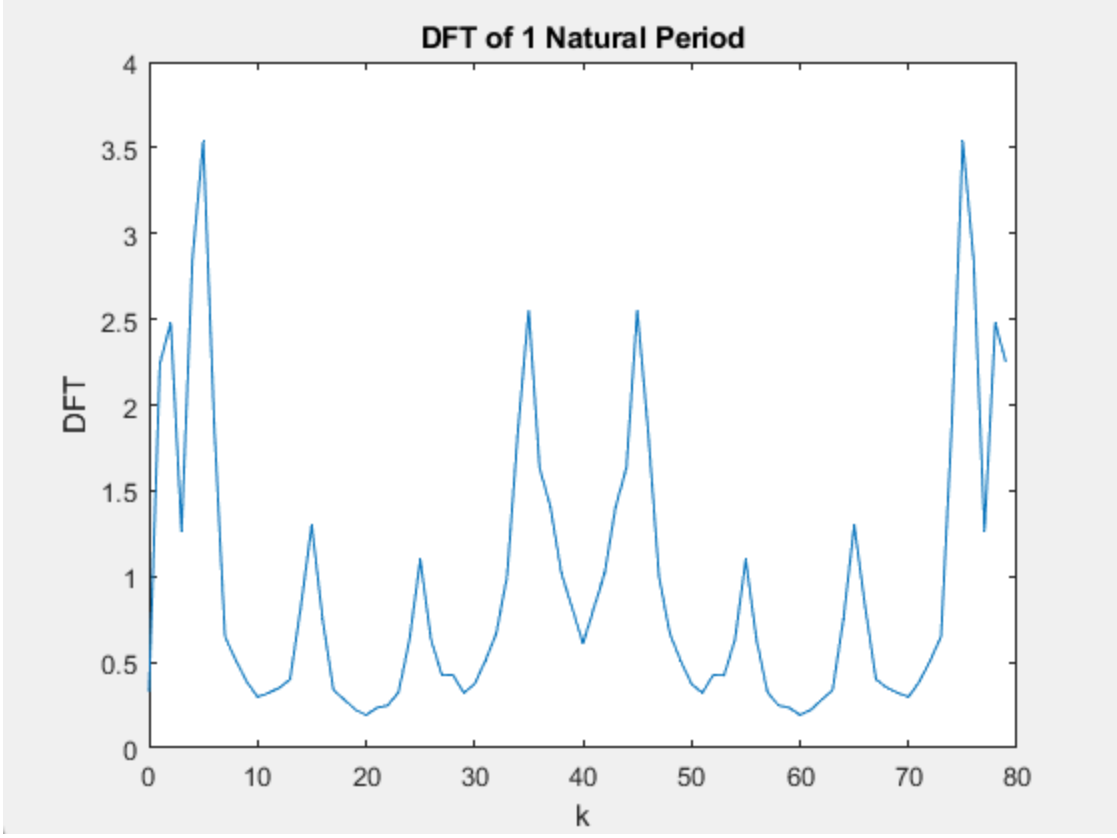

Figure 19: Magnitude of the DFT of one natural period of the audio signal

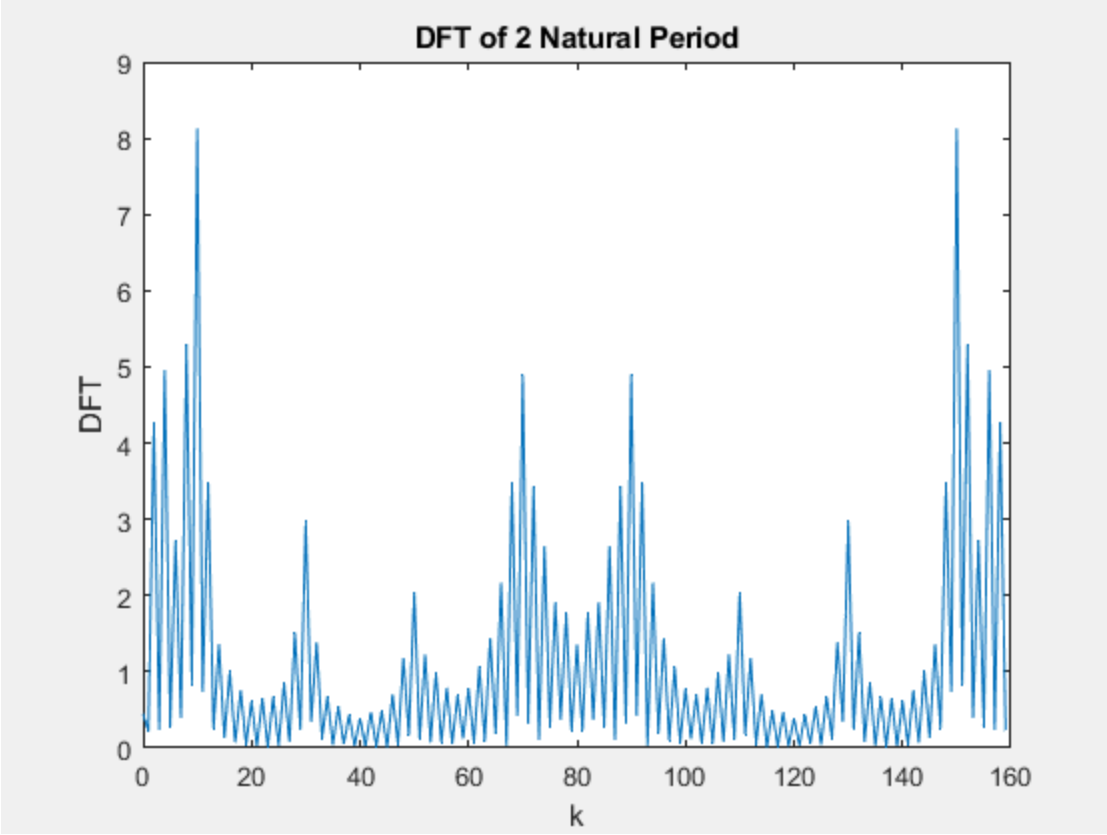

Figure 20: Magnitude of the DFT of two natural period of the audio signal

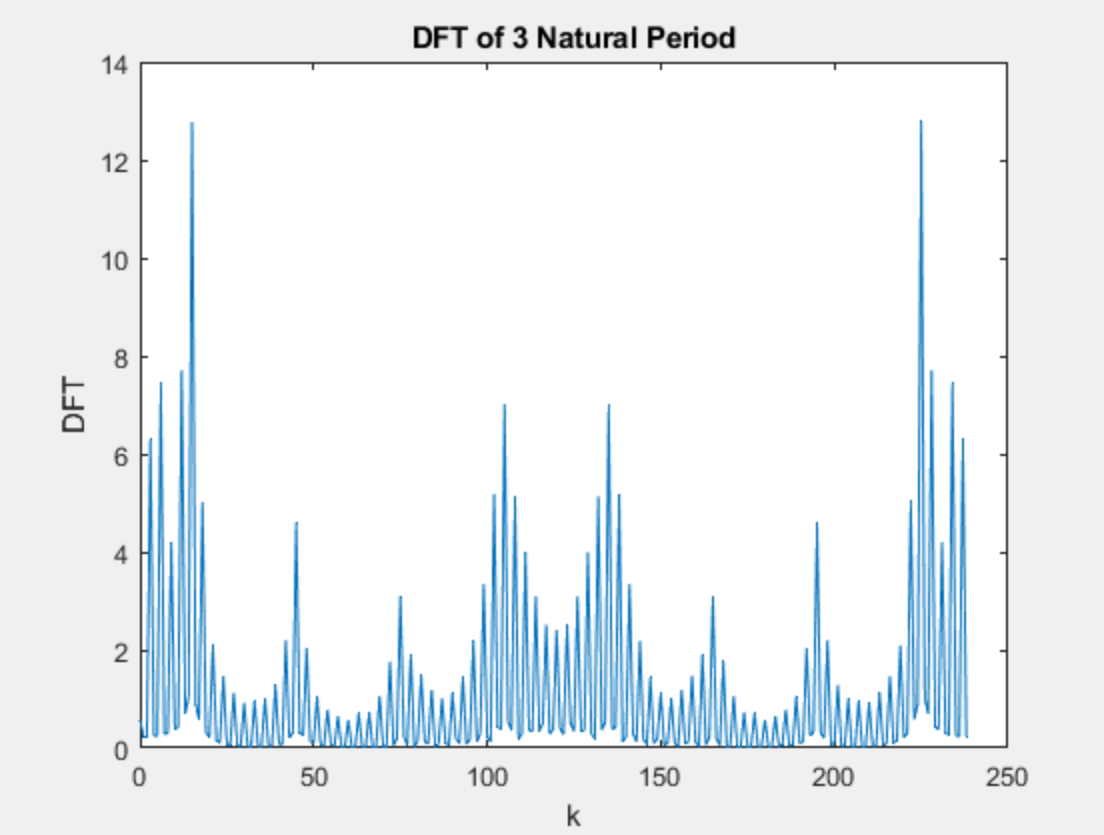

Figure 21: Magnitude of the DFT of three natural period of the audio signal

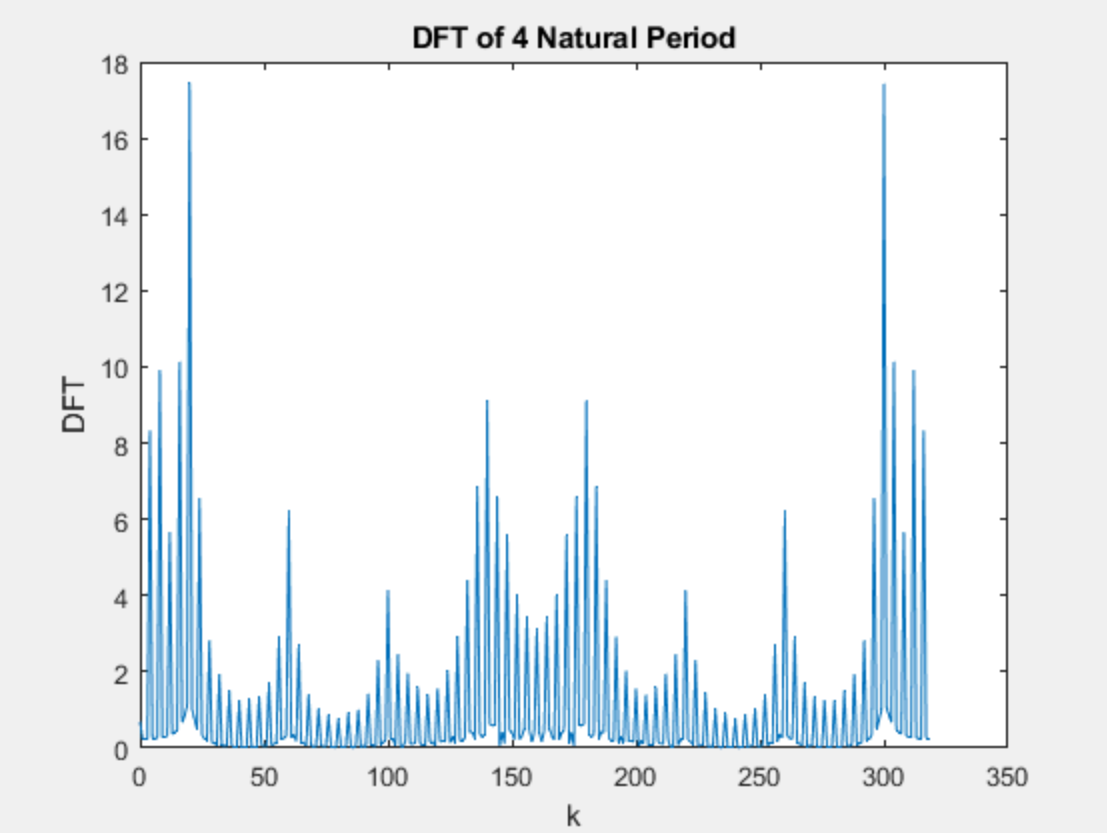

Figure 22: Magnitude of the DFT of four natural period of the audio signal

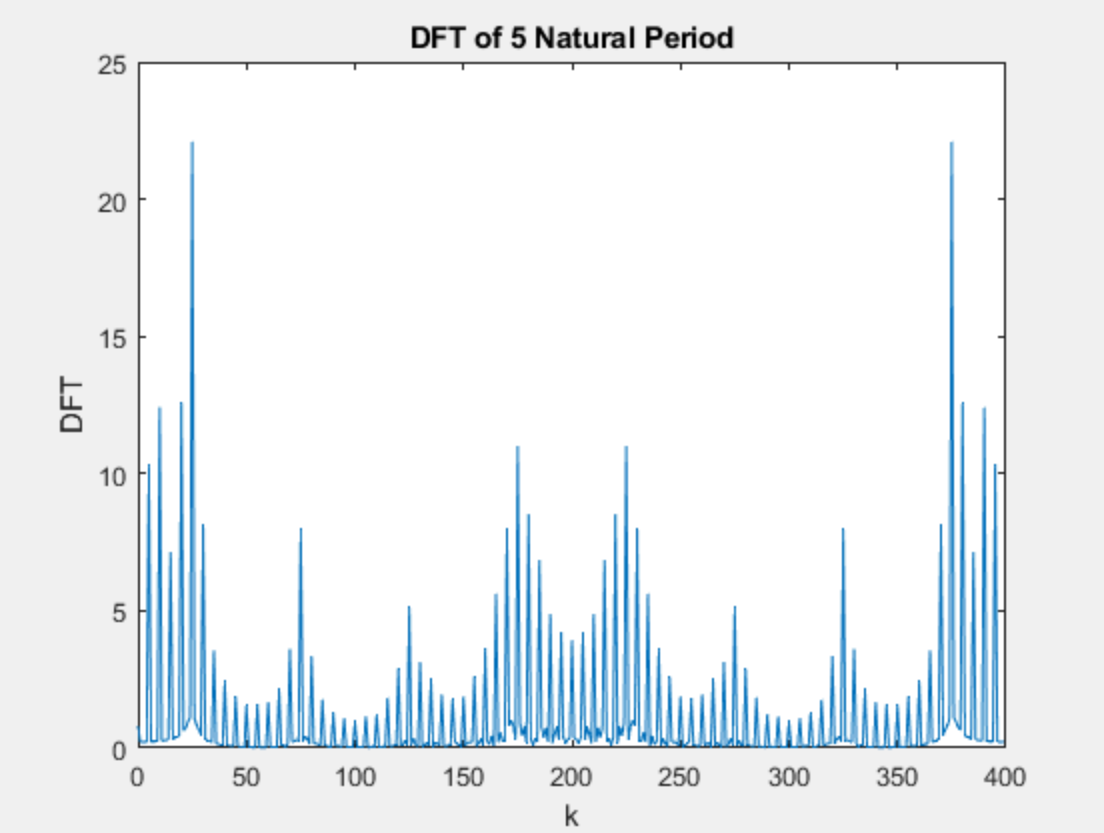

Figure 23: Magnitude of the DFT of five natural period of the audio signal

By increasing the window length, the spectral resolution of the signals are expected to improve. What this means is that it allows closer frequency components to be resolved when it otherwise wouldn't have with a smaller window length. This is evident when comparing the DFT of one natural period in Figure 19 with the DFT of two natural periods in Figure 20. For example, the region between k from 30 to 40 in Figure 19 is one pulse, while the same region in Figure 20 (with k between 60 and 80) shows multiple pulses. However, beyond two natural periods, the DFT does not look much different despite having improved spectral resolution. This is likely because the figures plot the DFT and not the DTFT, and the given number of samples does not have enough frequency sampling resolution to show the improved spectral resolution.

The figures below paint a clearer picture of the improved spectral resolution by comparing the DFT of two vs. four natural periods, with and without zero-padding.

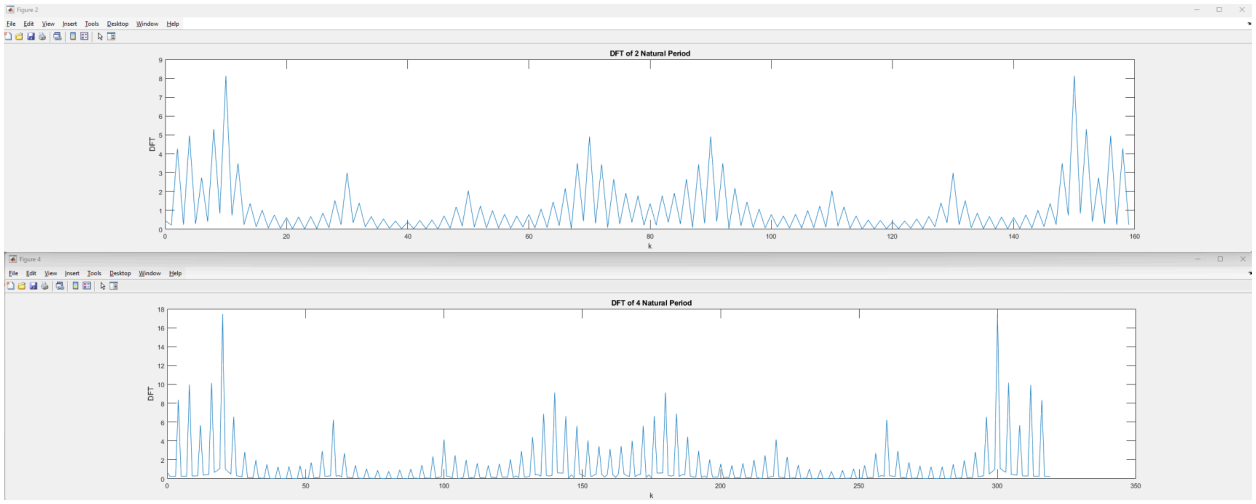

Figure 24: Magnitude of the DFT of two vs. four natural periods of the audio signal without zero-padding

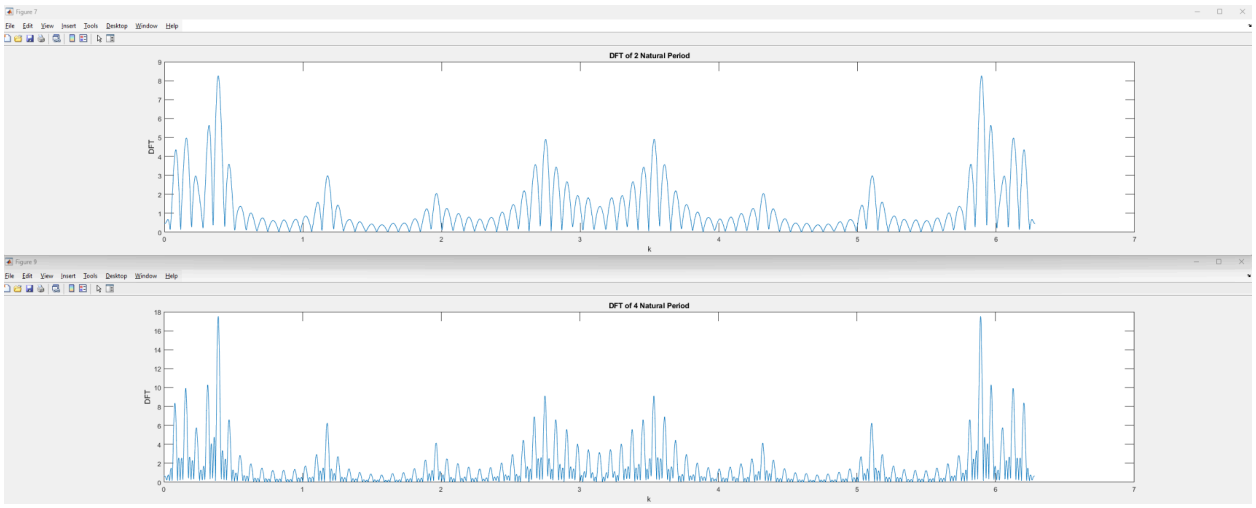

Figure 25: Magnitude of the DFT of two vs. four natural periods of the audio signal with zero-padding to N=5000

Figures 24 shows the same plots as Figures 20 and 22 but stretched to more clearly show the pulses. Figure 25 plots the same two signals but with added zero-padding to improve the frequency sampling resolution. Figure 25 is able to demonstrate the improved spectral resolution by increasing the window length.

3. FFT Application on FMCW Radar Signal Processing with the TMS 320 DSP Processor

3(b-vi) Comparison of CPU Time for the Two FFT Implementations

The first helper function was implemented by directly saving the real and imaginary components to the same location in the complex data type. The second helper function was implemented by interweaving the real and imaginary components in each memory location. For example, if there were four memory locations (1,2,3,4) and you needed to store two complex numbers(R1I1,R2I2), the data would be stored as follows:

1→R1
2→I1
3→R2
4→I2

The following plots were then produced:

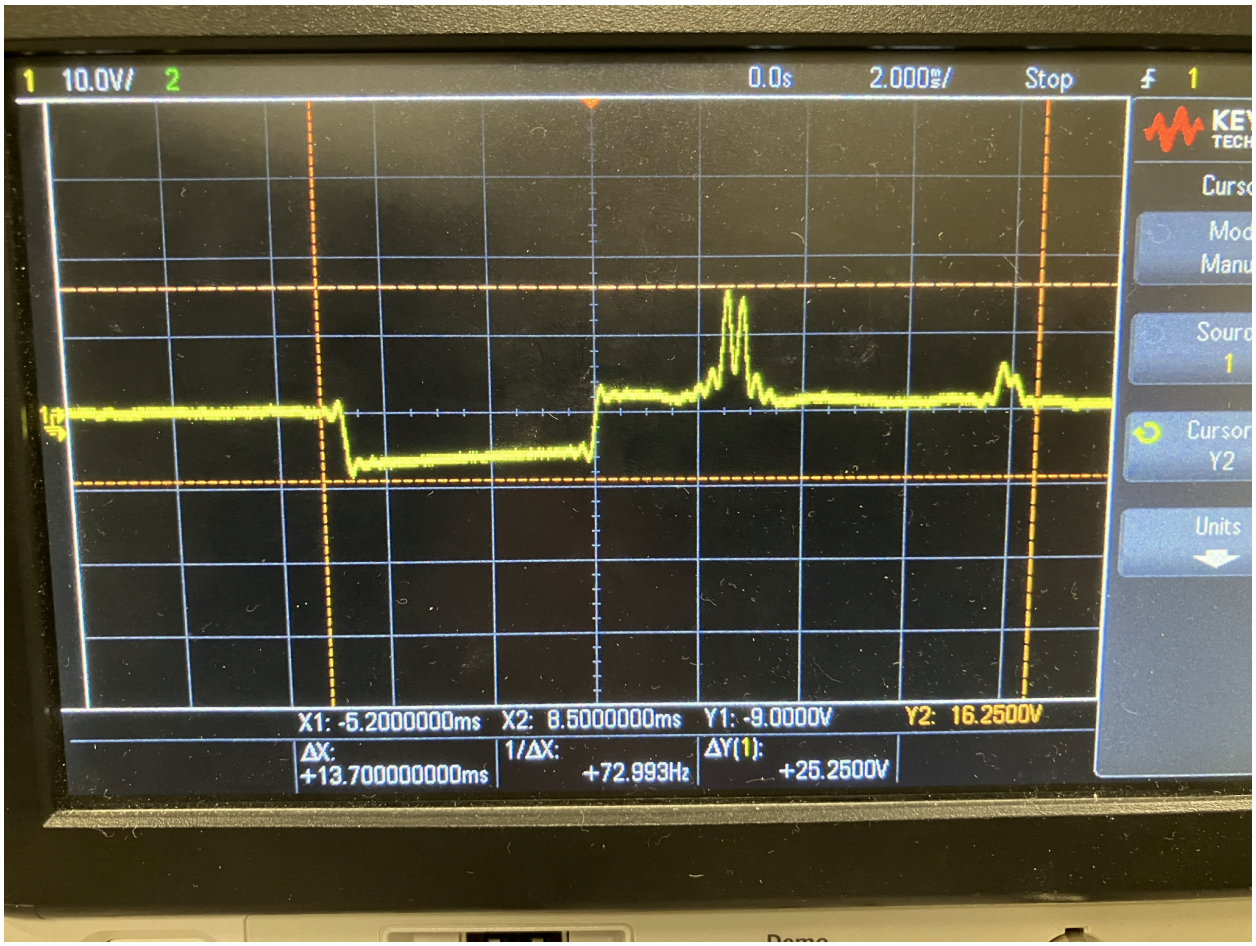

Figure 26: Plot of FFT produced from provided FFT function

Figure 27: Plot of FFT produced from DSP library FFT function

By comparing both Figure 26 and Figure 27, we see that the functions result in the exact same FFT signal. This shows that they are functionally equivalent. The real difference can be observed by comparing the compute time in number of cycles:

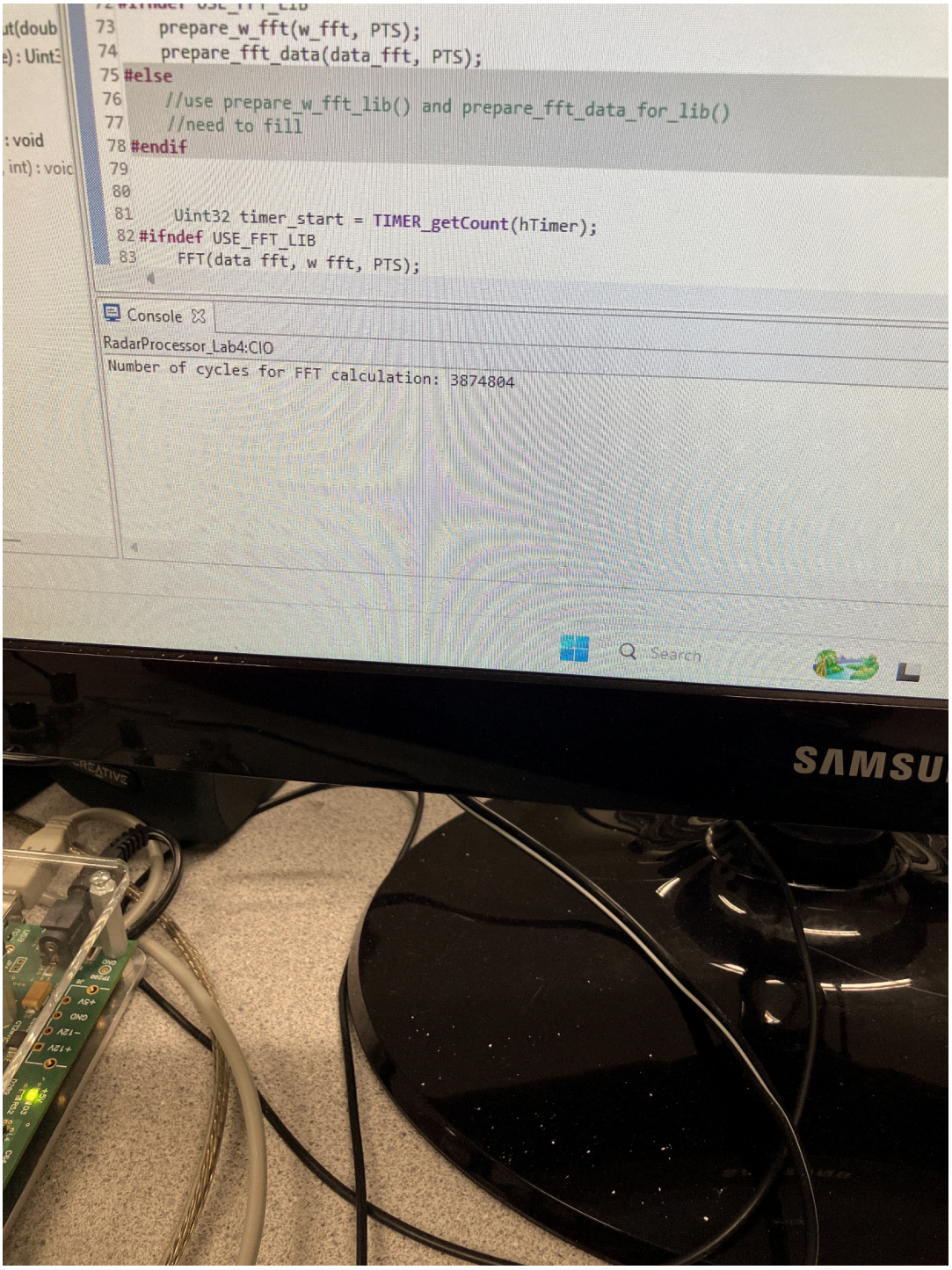

Figure 28: Compute time using given FFT function

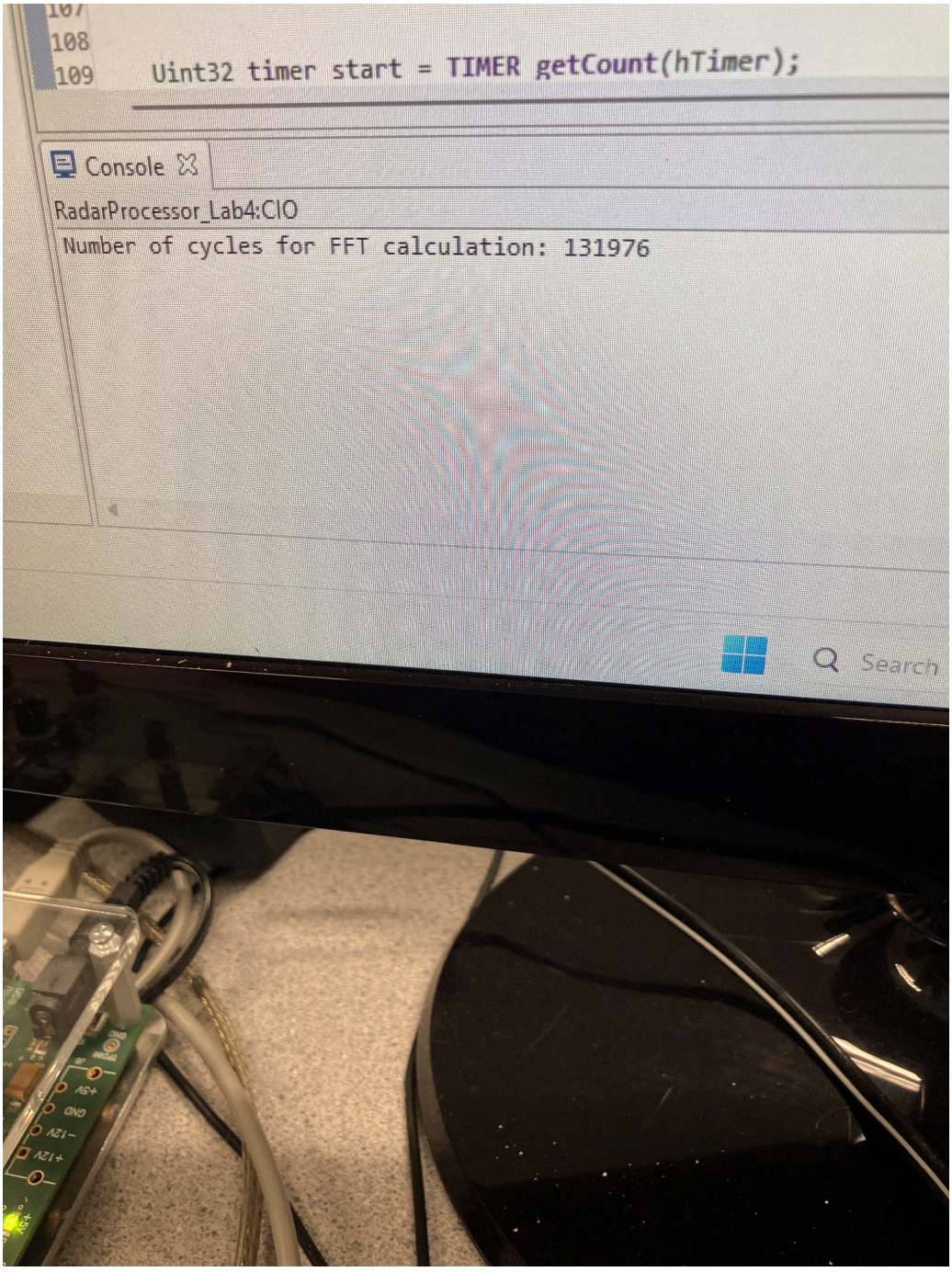

Figure 29: Compute time using DSP FFT function

From comparison of Figure 28 and Figure 29, we see that the DSP FFT has a faster compute time with 131976 cycles while the given FFT function has a compute time of 3874804 cycles. This magnitude of difference can be attributed to the given FFT function being based fully on code. The DSP FFT function however, is able to utilize the DSP board to its maximum potential. The board most likely allows for parallel processing, and since the FFT can be split into much smaller computations, this allows for the board to do many

calculations at once. There are also probably dedicated resources (multipliers, ALUs, etc.) and custom algorithms to boost efficiency.