Signals and Systems Project 2 Report

Introduction

Digital communication has been a key factor in the development of our society. With technological developments came better and more efficient ways to send and transmit information. For most of the 19th century, the telephone operator served as a mechanism to communicate parties globally, but with a rapidly evolving world came higher demand and need for more efficient communication systems. Dual tone multifrequency signals or DTMF were invented to solve this need.

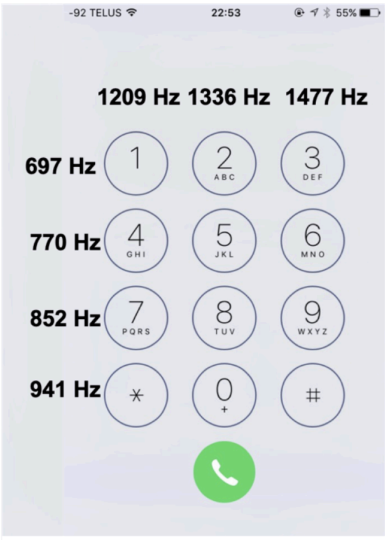


Figure 1: Frequency encoding of phone keypad.

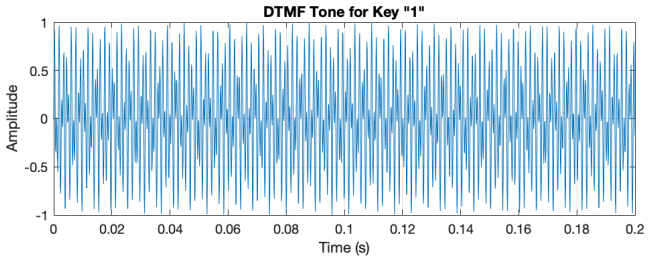
DTMF signals work by superimposing two sine wave signals of low and high frequencies respectively that correspond to a single key in a keypad. A combination of a high and a low frequencies makes it ideal transmit since it is very hard for humans to emulate such signals. Figure 1 shows a common phone keypad as well as the frequency components contained in each digit. Our goal is to build a system that can accurately encode and decode DTMF signals of different sampling rates, duration, and noise conditions.

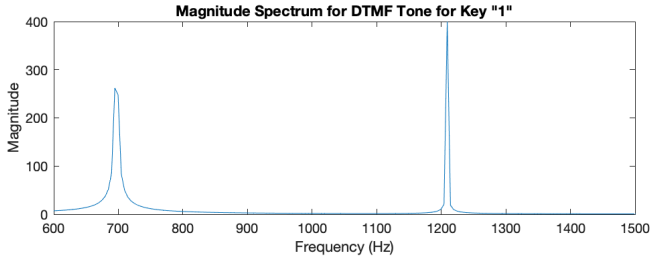
Part 1: Encoding Dual Tone Multi Frequency (DTMF) Signal

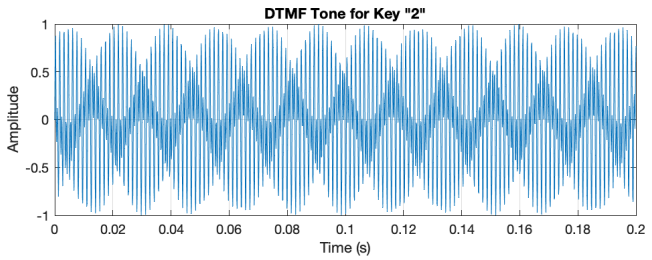
The first part of this report centers around encoding a DTMF signal corresponding to a singular key of any wanted duration and weight. The encoding process is relatively straightforward. We first used MATLAB's map function to map each keypad digit to a 1x2 array containing the frequency pairs. After knowing which frequencies to be encoded, we used MATLAB's sine function to produce the two sine signals over arrays of the desired duration and sampling rate.

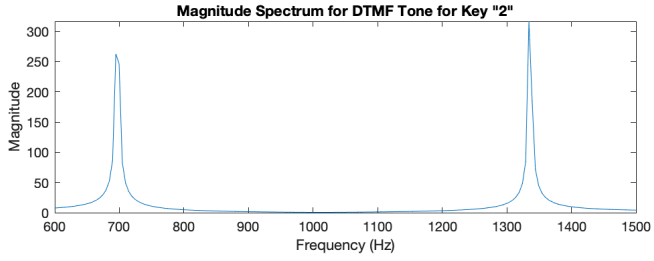
Code Checking and Results

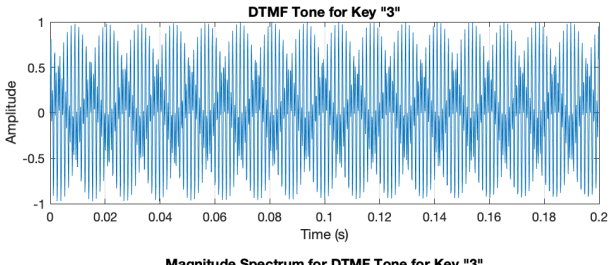
At this stage of development, we did not have any decoding mechanism to directly verify if our signals were encoded directly, so the best way of checking our signals was to plot their fourier transforms to verify the frequency components for each key match the expected frequencies as shown in Figure 1. I also went ahead and plotted the signals over time domain as well to illustrate both the sampling rate and signal duration. The following pages show the signal plots over time and frequency domain for each of the 12 keys. (*Weight [1 1], sample rate: 8000 Hz, Duration: 200 ms, I have also attached all figure files in the submission.)*.

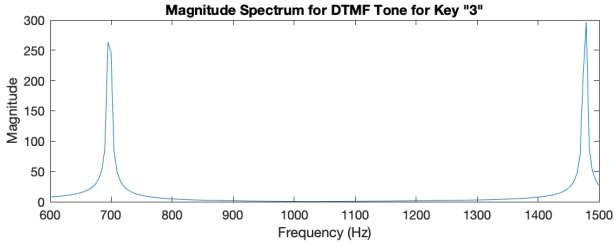

DTMF Tone for Key "1"
Amplitude
Time (s)

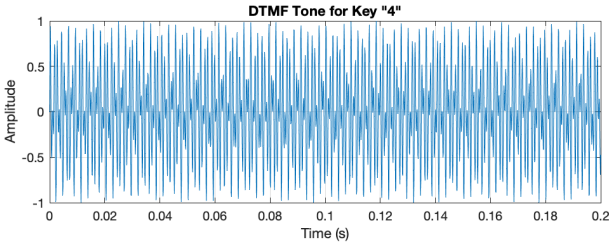

Magnitude Spectrum for DTMF Tone for Key "1"
Magnitude
Frequency (Hz)

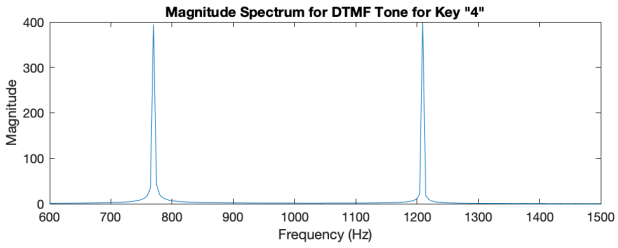

DTMF Tone for Key "2"
Amplitude
Time (s)

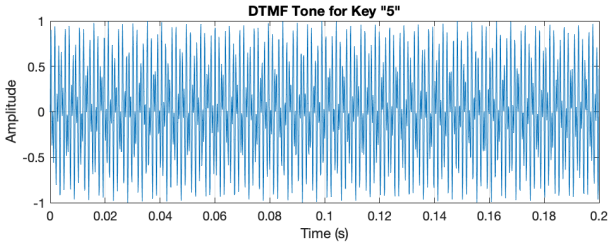

Magnitude Spectrum for DTMF Tone for Key "2"
Magnitude
Frequency (Hz)

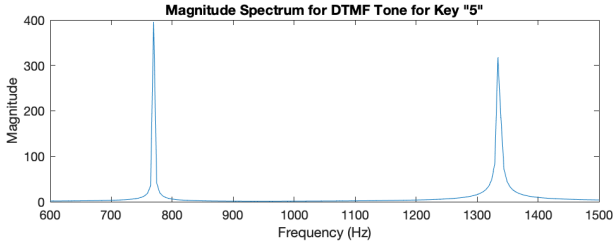

DTMF Tone for Key "3"
Amplitude
Time (s)

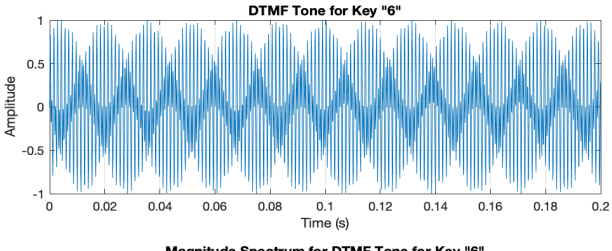

Magnitude Spectrum for DTMF Tone for Key "3"
Magnitude
Frequency (Hz)

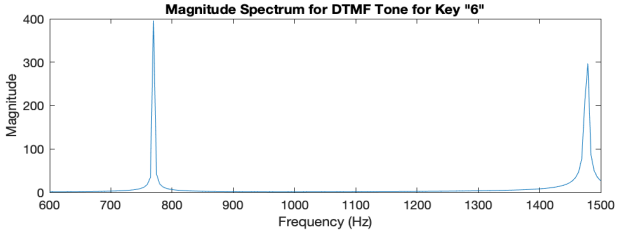

DTMF Tone for Key "4"
Amplitude
Time (s)

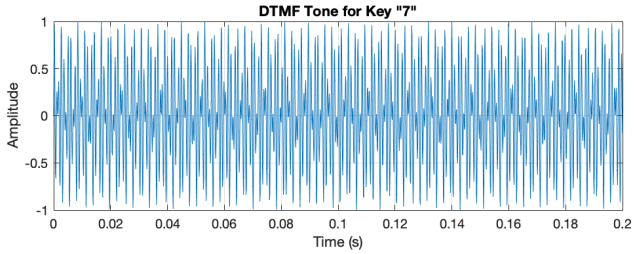

Magnitude Spectrum for DTMF Tone for Key "4"
Magnitude
Frequency (Hz)

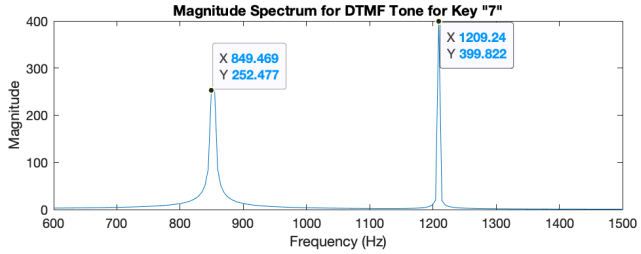

DTMF Tone for Key "5"
Amplitude
Time (s)

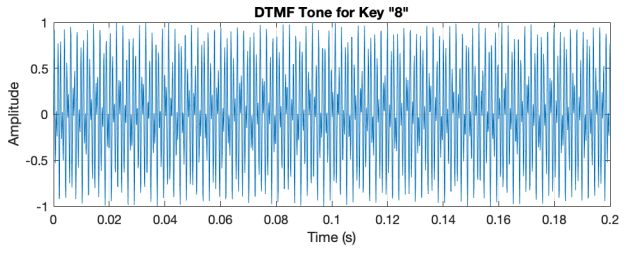

Magnitude Spectrum for DTMF Tone for Key "5"
Magnitude
Frequency (Hz)

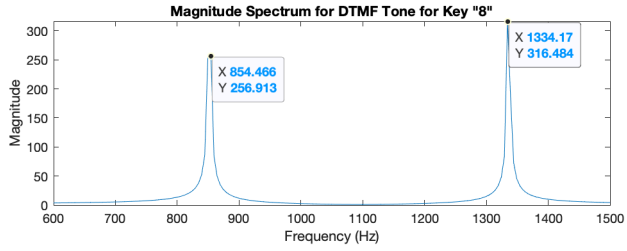

DTMF Tone for Key "6"
Amplitude
Time (s)

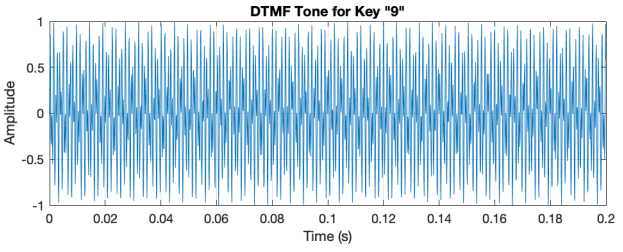

Magnitude Spectrum for DTMF Tone for Key "6"
Magnitude
Frequency (Hz)

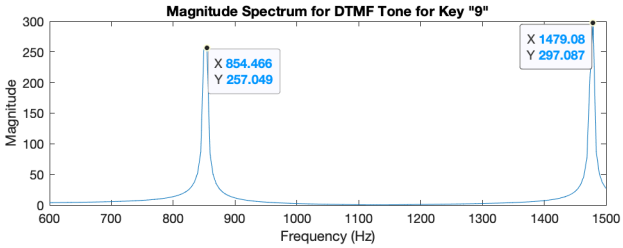

DTMF Tone for Key "7"
Amplitude
Time (s)

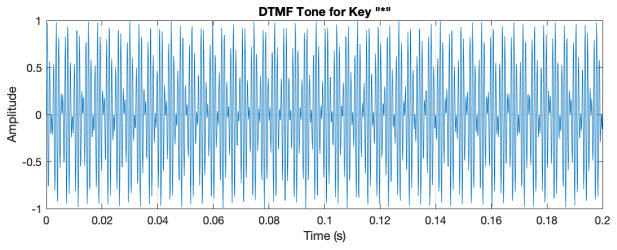

Magnitude Spectrum for DTMF Tone for Key "7"
X 849.469
Y 252.477
X 1209.24
Y 399.822
Magnitude
Frequency (Hz)

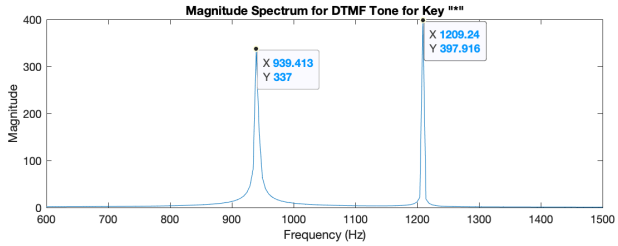

DTMF Tone for Key "8"
Amplitude
Time (s)

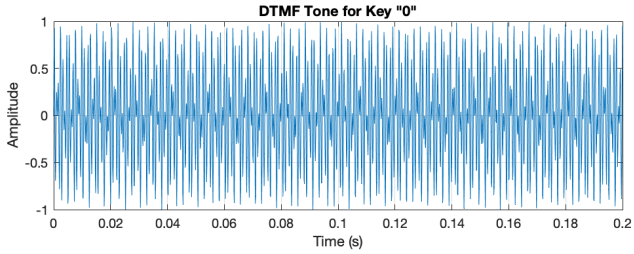

Magnitude Spectrum for DTMF Tone for Key "8"
X 854.466
Y 256.913
X 1334.17
Y 316.484
Magnitude
Frequency (Hz)

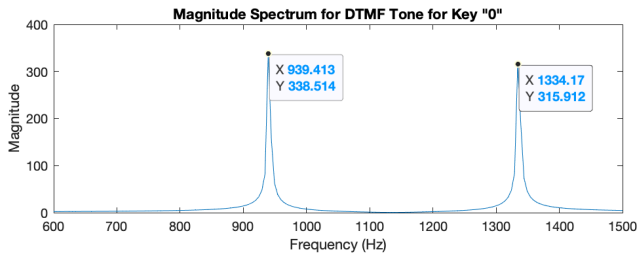

DTMF Tone for Key "9"
Amplitude
Time (s)

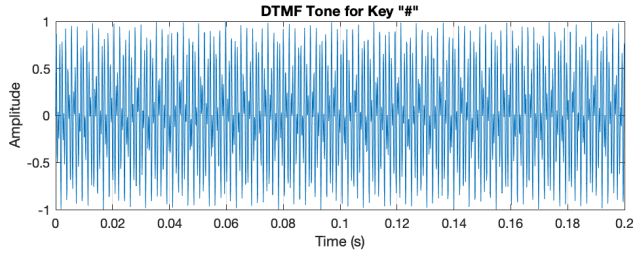

Magnitude Spectrum for DTMF Tone for Key "9"
X 854.466
Y 257.049
X 1479.08
Y 297.087
Magnitude
Frequency (Hz)

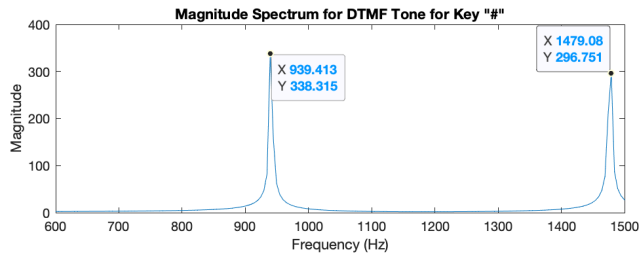

DTMF Tone for Key "*"
Amplitude
Time (s)


Magnitude Spectrum for DTMF Tone for Key "*"
X 939.413
Y 337
X 1209.24
Y 397.916
Magnitude
Frequency (Hz)


DTMF Tone for Key "0"
Amplitude
Time (s)


Magnitude Spectrum for DTMF Tone for Key "0"
X 939.413
Y 338.514
X 1334.17
Y 315.912
Magnitude
Frequency (Hz)


DTMF Tone for Key "#"
Amplitude
Time (s)


Magnitude Spectrum for DTMF Tone for Key "#"
X 939.413
Y 338.315
X 1479.08
Y 296.751
Magnitude
Frequency (Hz)

The above plots show the frequency components of each of the 12 keys. Verifying with each of the 12 keys using Figure 1 confirms all signals have correct frequency components. I have also generated .wav files in my submission for further verification. As a note, I also found it helpful to play the signals myself using MATLAB's sound() function and comparing with the keys on my phone. I found that I was surprisingly good at detecting mismatches between sounds, which helped me debug.

Part 2: Decoding Single DTMF Keys

After successfully implementing a working DTMF encoding mechanism, the challenge moved to decoding transmitted DTMF signals into the keys that were originally encoded. We designed the decoding mechanism already in Part 3 and application based uses, taking into account noisy input signals and arbitrary signal duration and weighting. The code first takes the receiving signal and applies a fourier transform on it using MATLAB's fft() function. We then used a bandpass filter to remove unwanted frequencies outside of the DTMF range, I chose to filter out frequencies outside of 1.5 % of the maximum and minimum DTMF frequencies respectively (697 - 1477 Hz). I chose this threshold specifically as it was the one recommended by the ITU - T DTMF reception tolerance.

After applying the bandpass filter, we then sampled the two largest frequencies in the signal, assuming weaker peaks corresponded to noise signals. To account for this procedure working under any arbitrary weight of high to low frequencies, I separated the fourier transform array into two sub-arrays of low and high frequencies, applying the thresholding process separately to find the two largest peaks. I did this by creating a separate frequency array that mapped one frequency to every element of $X(\omega)$. To do this I created a ones row vector and multiplied it over the sampling rate (max frequency) over the length of $X(\omega)$.

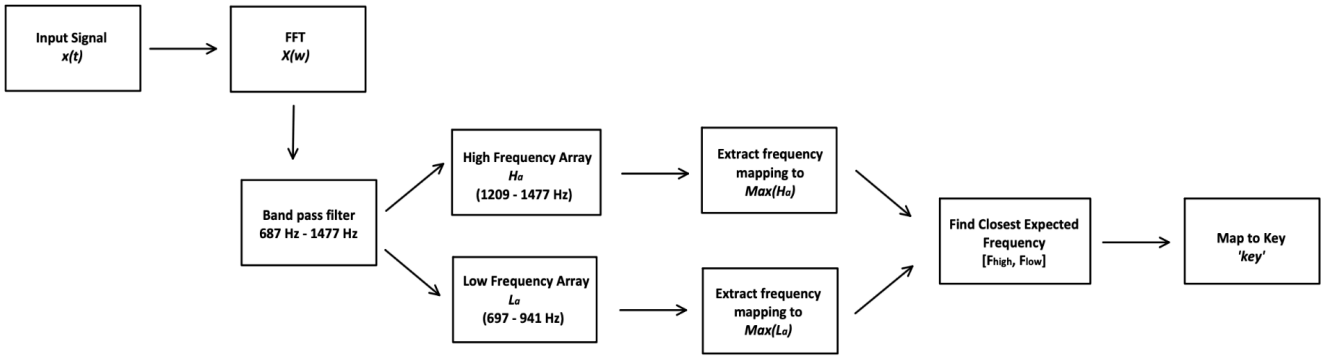


Figure 2: DTMFdecoding System Diagram

I then matched the closest expected frequency to the one obtained in my sample for both frequencies to get the final DTMF signal. Finally I mapped a 1 x 2 frequency array to the corresponding digit to get the decoded key using MATLAB's map function. Figure 2 gives a good illustration of the overall decoding design.

Testing Code Over Noise Conditions

To test the reliability of my code over noisy conditions, I created a new function called the DTMFencode function from part 1 to generate different key signals. The modification involved adding a noise array of equal length to $x(t)$ using MATLAB's randn function. The final modified signal can be expressed as $x(t) + a \cdot n(t)$ where *a* is the level of noise we wish to generate. Refer to the *DTMFencodeseq.m* file to see the full functionality. I chose the '5' key and tested it under multiple noise levels. After testing for many scalar values, I found my decode function to start consistently failing after *a = 9.1*.

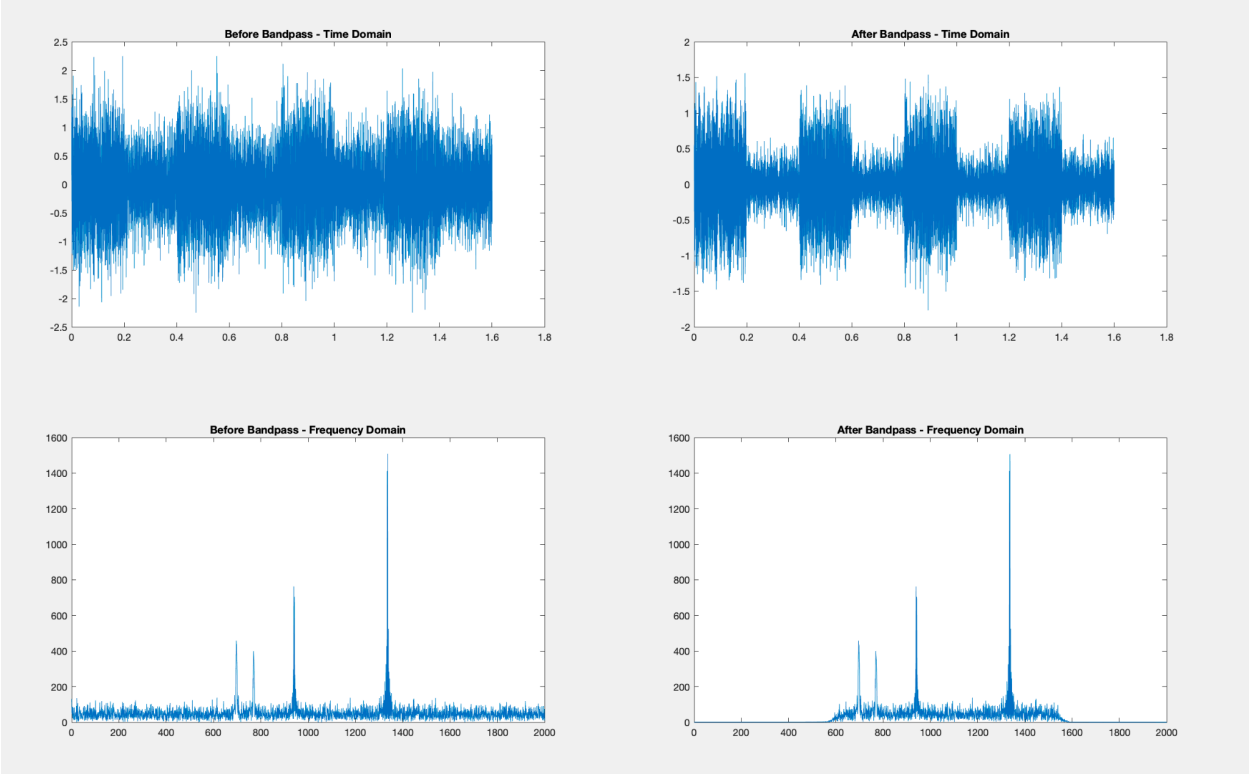
Noise level	Expected value 5
2.2	5
2.3	5
2.4	5
2.5	5
2.6	5
2.7	5
2.8	5
2.9	5
3.0	**8**
3.1	5
3.2	6
3.3	5
3.4	4
3.5	5
3.6	4

I found this performance to be quite impressive given the high sound to noise ratio. Some factors that might explain this observation have to do with the addition of random white noise which spreads equally over the entire signal, instead of concentrating in very concentrated places in the signal, especially taking into account the peak finding functionality of my signal. If we had noise concentrated over a short period of time at higher magnitude, my system would potentially confuse it for a peak and read of its frequency/

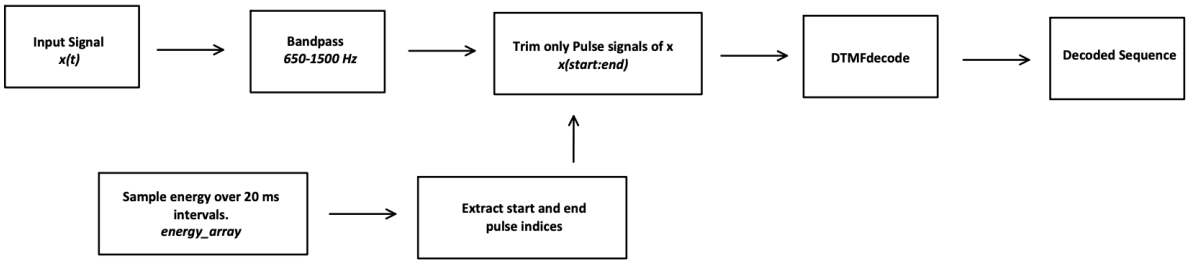
Part 3: Decoding DTMF Sequences

The natural next step in this report is decoding entire sequences of keys, rather than just a singular one. For this we have created the DTMFsequence function, which outputs a string of character keys and the sample rate, taking in a .wav file with the decoded signal as parameter. The following part of my report walks through the full system and functionality of our section.

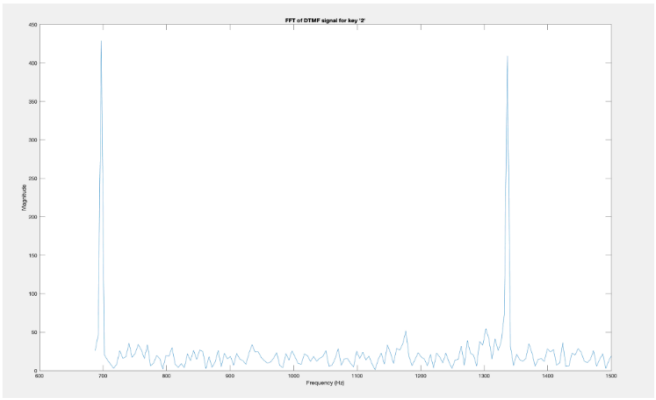
The first part of this system involves calling MATLAB's bandpass function to filter out ranges not included from 650 to 1500 Hz. Even though the DTMFdecode function applies this filtering process for a singular key, it is important to filter out frequencies once again to avoid MATLAB confusing 'silence' intervals for actual key signals that could be caused by such frequency components. The following plots show the original raw signal before and after the bandpass filter application in both time and frequency domain. It is important to emphasize the change over time domain as it illustrates the importance of adding this step to identify key pulses from silence for the string "2005".

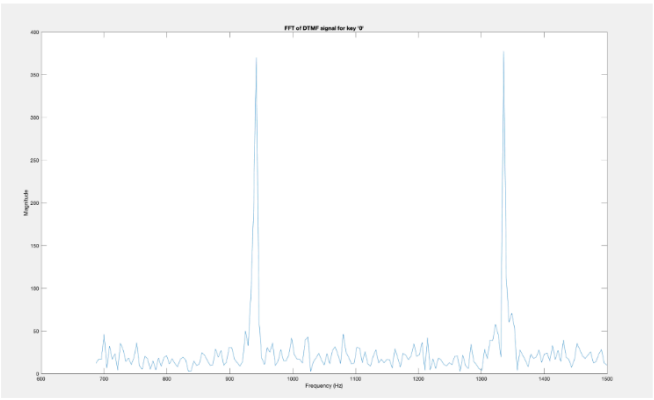


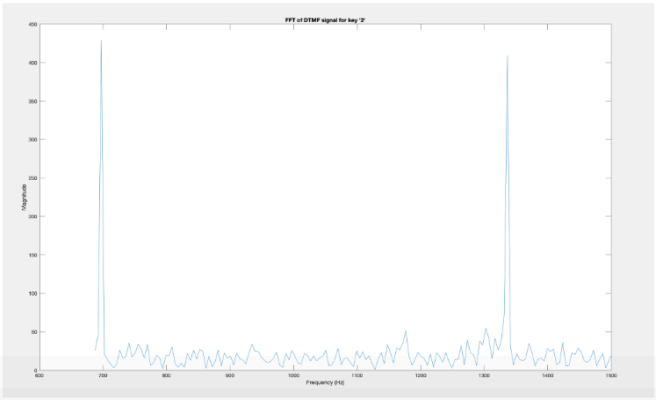
Next we use a 10 ms window to create small energy buckets (as hinted on the instruction manual) that build up over this period. Thus, the function sums the square elements of the input signal during this time window to create a singular energy element in the energy array. Even though we are mapping over 20 ms intervals, the window slides over each element, guaranteeing the energy array is of equal size as the input signal. The system then applied MATLAB's movmean function to smooth out the energy array to avoid significant energy contributions from noise that translate into abrupt changes in energy.

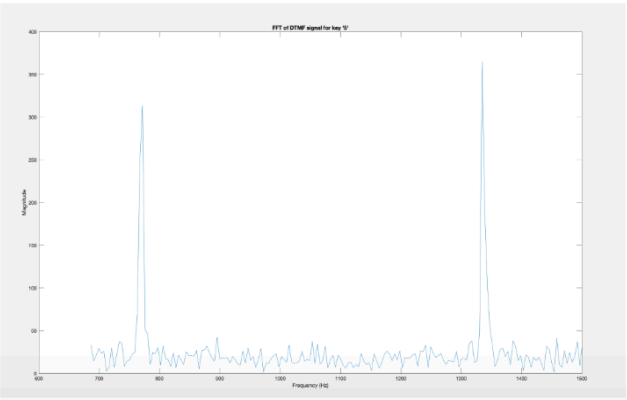


I then created an energy threshold and defined a key to start and end on the points in which the energy of the signal surpassed 20% of the maximum energy bucket in the entire array. I chose 20 % as it is often used by industry standard DTMF decoders. I then used this threshold to identify the starting and ending point indexes of each key pulse and stored them into two different arrays respectively, one containing all of the starting points indices of the pulses while one contained all the end indices. I then looped over the arrays (same length) and passed a cropped version of my filtered signal array dictated by the current iteration of the start and stop indices retreived into the DTMFdecode function from part 2. The decoded key was then stored into a string array, with each loop iteration concantinating a new character to the output string. The diagrams below show the frequency domain plots of each key pulse filtered as seen by the DTMFdecode function for the string “2005”.









Generating Test Signals

To test DTMFsequence functionality, I created *DTMFencodeseq*, which can be found in the file submissions. This function utilizes the DTMFencode from part 1 to generate a signal corresponding to a custom sequence of keys passed as a string in the input. The function also uses MATLAB's randn function to add artificial noise (normalized) to the signal (also normalized) in the form of $x(t) = s(t) + a \cdot n(t)$ for testing. The table below shows the performance of DTMFsequence decoding the signal "678" generated by DTFMencodeseq under various noise stress levels.

Noise level *a*	Decoded Sequence
Noise level *a*	Decoded Sequence
1.3	678
1.4	678
1.5	678
1.6	678
1.7	678
1.8	**6787**
1.9	6*#78*
2	678

We find the noise level starts failing after an *a* value of 1.8, meaning our decoder can reliably detect signals with noise over 1.7 times the signal amplitude.

Decoding Real Life DTMF Office Recordings

The table below shows the results decoded by DTMFseq over all 20 real life DTMF office recordings as provided in the *DTMFexamples* folder.

Filename	**Decoded Sequence**
dtmf1.wav	182846
dtmf2.wav	31415
dtmf3.wav	271827
dtmf4.wav	8548928
dtmf5.wav	926535
dtmf6.wav	8548928
dtmf7.wav	8548928
dtmf8.wav	20001

dtmf9.wav	8548928
dtmf10.wav	12859
dtmf11.wav	6619
dtmf12.wav	1283
dtmf13.wav	20164
dtmf14.wav	196509
dtmf15.wav	110405*
dtmf16.wav	20001
dtmf17.wav	1071135
dtmf18.wav	121285
dtmf19.wav	48928151
dtmf20.wav	1965