Project Report for Spring 2022 EN.520.214 Signals and Systems

Johns Hopkins University

Dual Tone Multi-frequency (DTMF) Audio Decode Implementation in MATLAB

Huanying(Joy) Yeh

Supervisor: Mounya Elhilali, PhD.

Co-Supervisor: John Han, Sandeep Kothinti, ANnapurna Kala

April 2022

Contents

Chapter 1

Introduction

This project is MATLAB computational tool implementation for dual tone multi-frequency decode. Given an arbitrary audio file (.wav) with unknown duration, audio sampling rate, and frequency amplitude weighting, the tool can can distinguish and decode target audio from additive and multiplicative noise.

The main objective is to decode unknown lengths of phone number presses, as each of the twelve keys on the keypad corresponds to a sound with a specific addition of two frequencies, ranging from 697Hz to 1477Hz. It's an attempt to build a simple "spying software" with simple signal processing concepts, such as Fourier transform, band pass filters, and root-mean-square error calculation.

1.1 Motivation

Fourier transform and filtering are two essential concepts in signals and systems, allowing engineers to easily decipher complicated time-domain data and identify signal patterns through a new perspective: the frequency domain. Frequency-domain analysis allows one to extract the target signals even under heavy noise. This project demonstrate the power of Fourier analysis and filtering, showing the main principles, process, result, and areas of improvement with the code implementation and plots. With powerful MATLAB build-in functions, one can create a robust decoding tool

with ease and apply the mathematical concepts to real-life situations.

1.2 Objectives

The objective of this project is to design, test, and build a phone-key -press-sequence decoding tool. The specifications are:

Encoding:

- Encode any of the 12 phone keys (1, 2, 3, 4, 5, 6, 7, 8, 9, *, 0, #) into the standard DTMF audio signal with their corresponding dual-frequency ranges, as shown in figure 1. - Each key is composed of two frequencies, which are the frequencies located at its respective row and columnn (fig 1). The amplitude weighting of the two can be adjusted by the user in the function parameters. The user can also change the sampling rate and duration of the audio signal.
- The default parameters of the output .wav file: duration 200 ms, weight 1:1, sampling rate 8000 Hz. Values below 3000 Hz are not accepted.

Decoding:

- A MATLAB function that takes in an input .wav file with arbitrary duration, amplitude weighting, and sampling rates - Outputs a single character, which is one of the twelve possible keys. - Also outputs the sampling rate (fs) of the input signal.

Sequence Decoding wtih Noise:

- A MATLAB function that takes an .wav file as input of an arbitrary amount of key presses in noisy environments (ie. a real office). - Outputs a string of characters, each corresponding to one of the twelve possible keys.

1.3 Structure of the Report

This report goes over: - Code implementation steps
- Figures detailing the signal processing methods
- Testing procedures
- A discussion section on potential areas of improvement

Figure 1: Phone Keypad Frequencies

Chapter 2

Methods

2.1 Encoding DTMF

2.1.1 Implementation Overview

Each encoded .wav audio signal is a superposition of sinusoidal waves of two specified frequencies assigned to the key, by the DTMF convention for all cellphone producers.

Implementation steps of the DTMFencode() Function

1. Read the single character key to be encoded
2. Find the two desired frequencies of this key
3. R the sampling rate, duration, and weighting
4. Calculate the duration and amplitude of each sinusoidal signal based on the parameters
5. Add the two sin waves together
6. Plot the two time-domain base signals and the time- and frequency domain-signals of the encoded output. Label parameters on the plot.
7. Write the output wave to an audio file

Important Formulas

$$num\ samples\ total = duration\ /\ 1000(ms/s) * fs$$

$$row\ signal = weight_1 * sin(freq_1 * \pi * t)$$

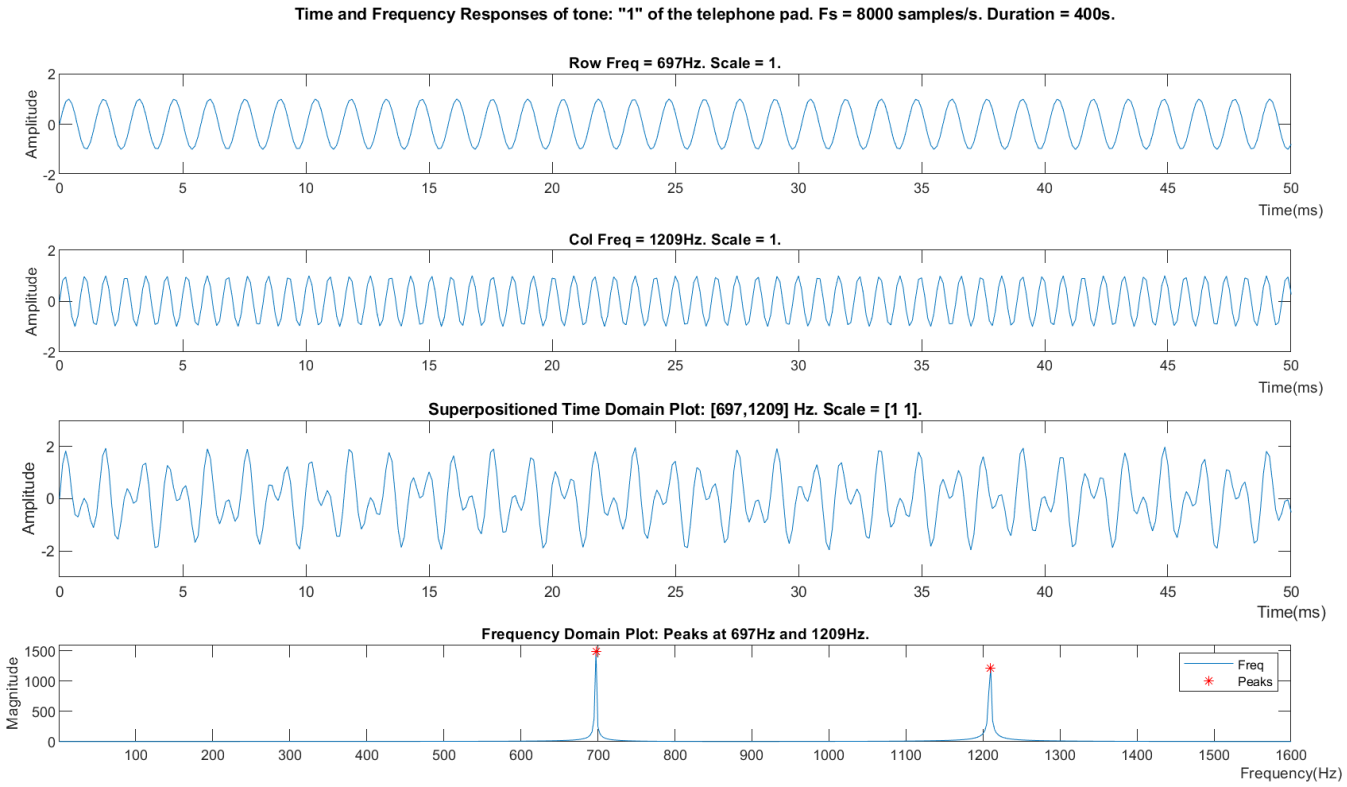
$$col\ signal = weight_2 * sin(freq_2 * \pi * t)$$

$$this\ signal = row\ signal + col\ signal$$

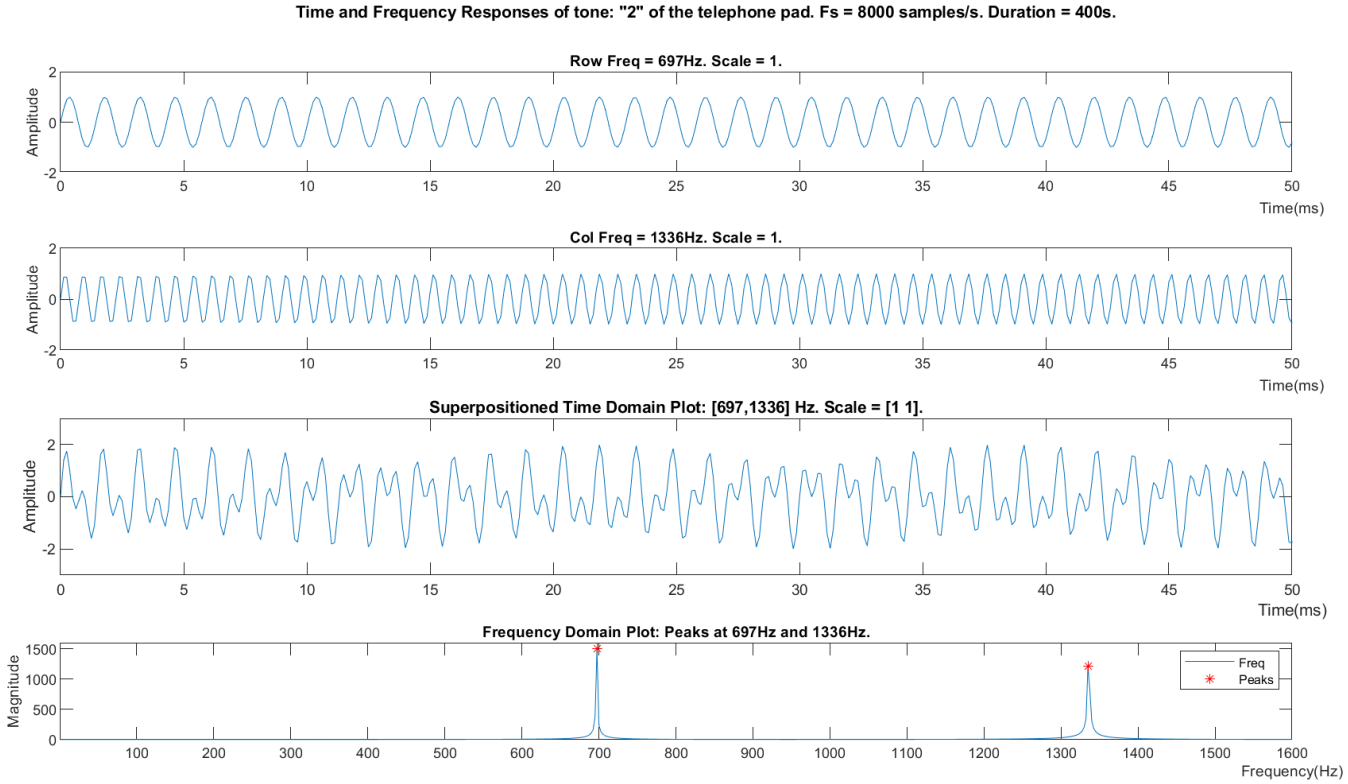
2.1.2 Plots for Each Encoded Key

The relevant plots of each key (1-12) are shown below:

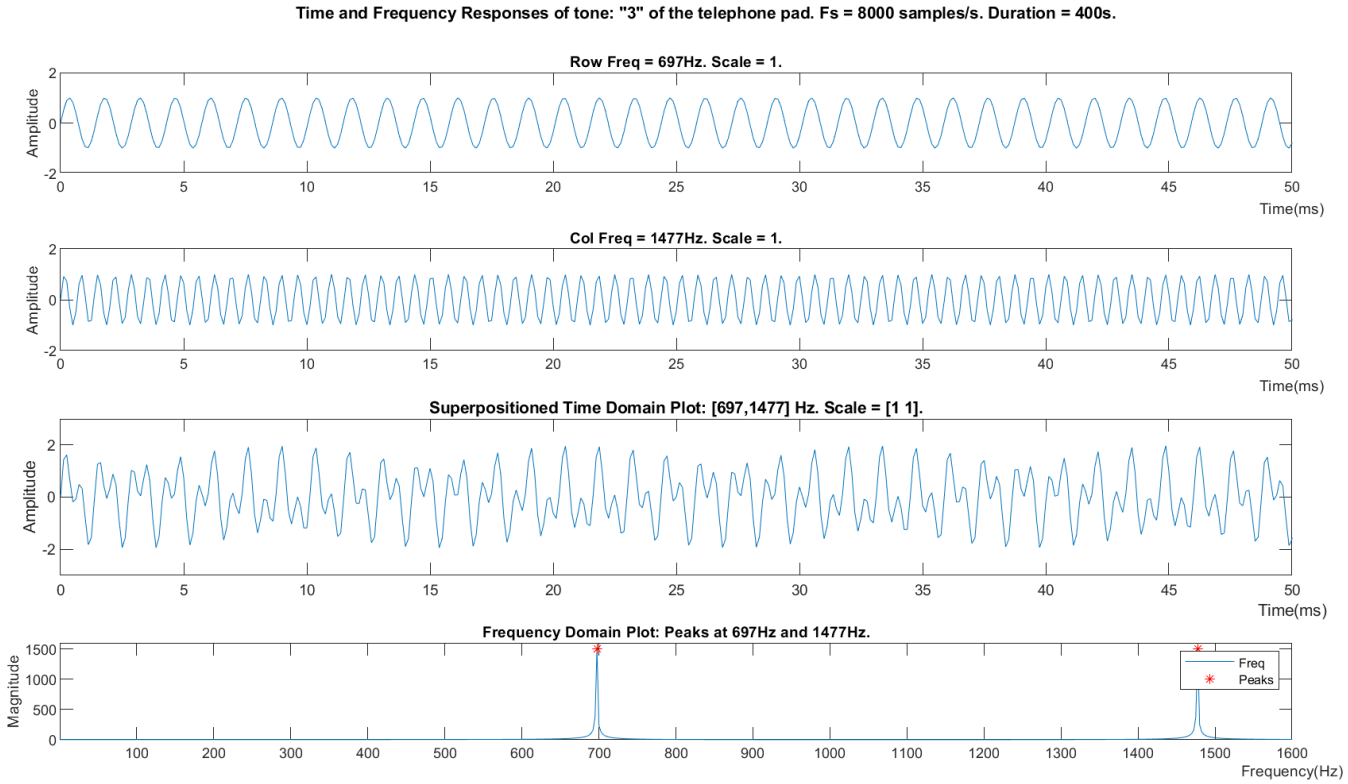
'1':



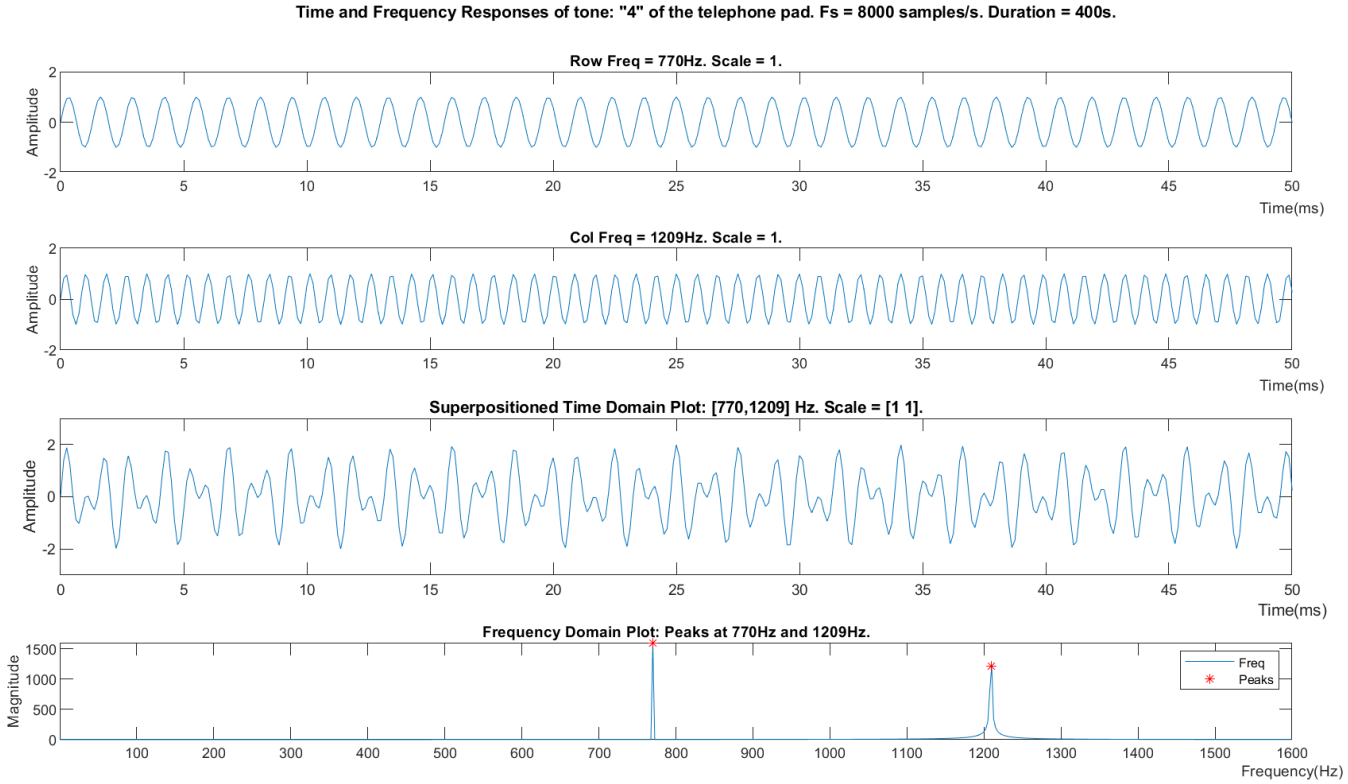
’2’:


Time and Frequency Responses of tone: "2" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 697Hz. Scale = 1.
Col Freq = 1336Hz. Scale = 1.
Superpositioned Time Domain Plot: [697,1336] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 697Hz and 1336Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

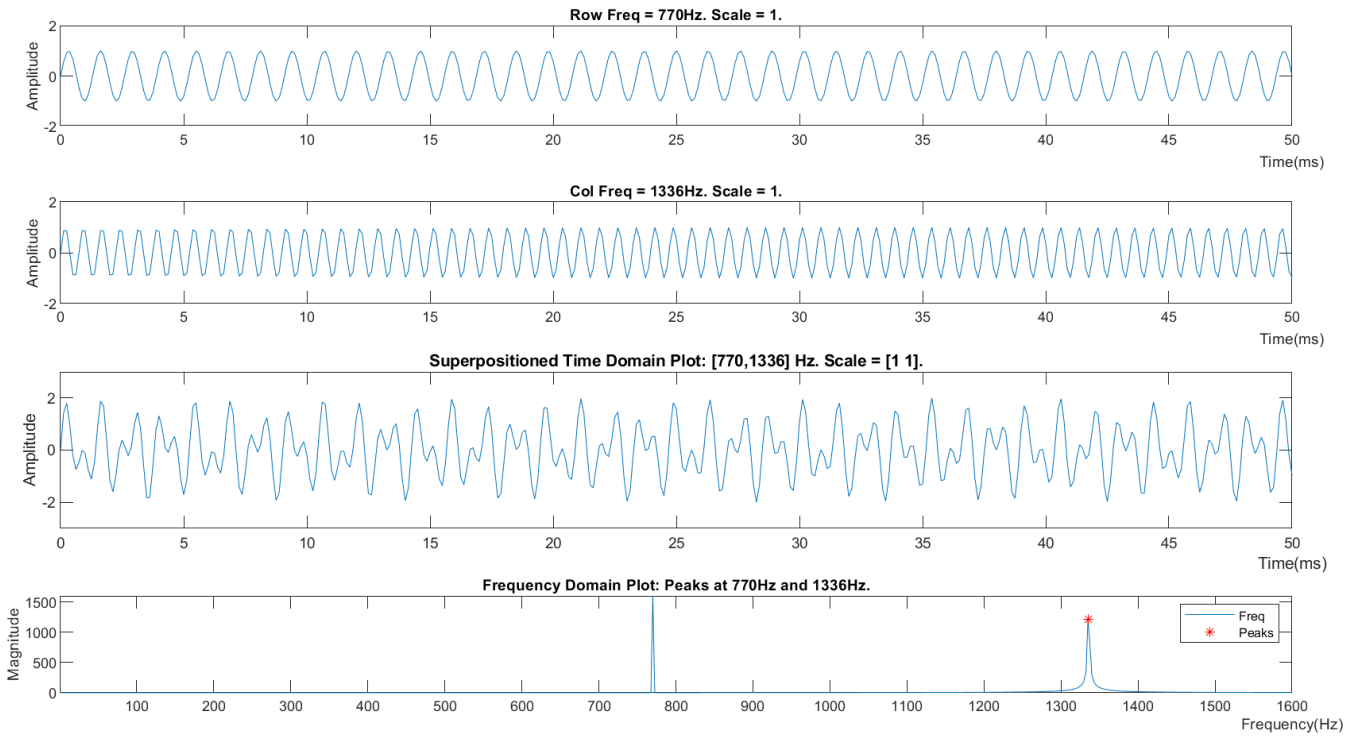
’3’:


Time and Frequency Responses of tone: "3" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 697Hz. Scale = 1.
Col Freq = 1477Hz. Scale = 1.
Superpositioned Time Domain Plot: [697,1477] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 697Hz and 1477Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

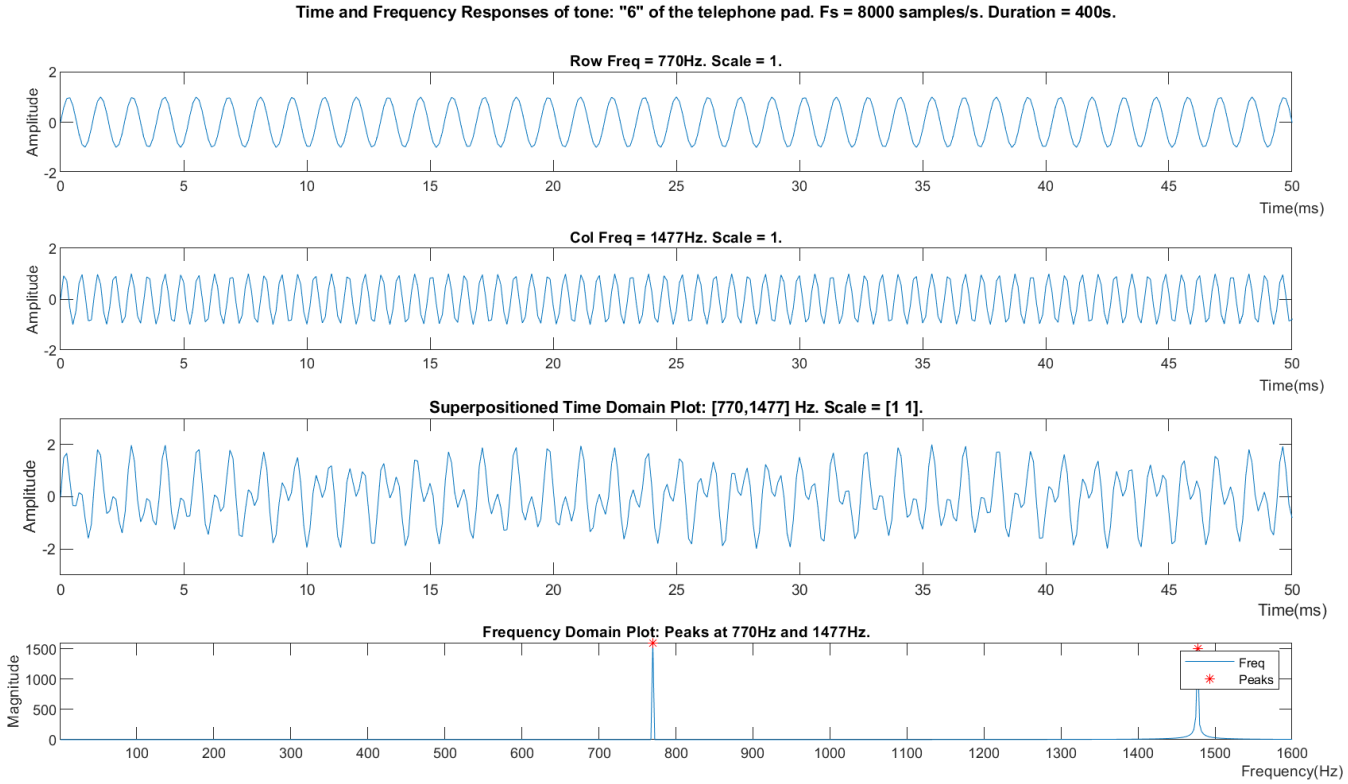
'4':


Time and Frequency Responses of tone: "4" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 770Hz. Scale = 1.
Col Freq = 1209Hz. Scale = 1.
Superpositioned Time Domain Plot: [770,1209] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 770Hz and 1209Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

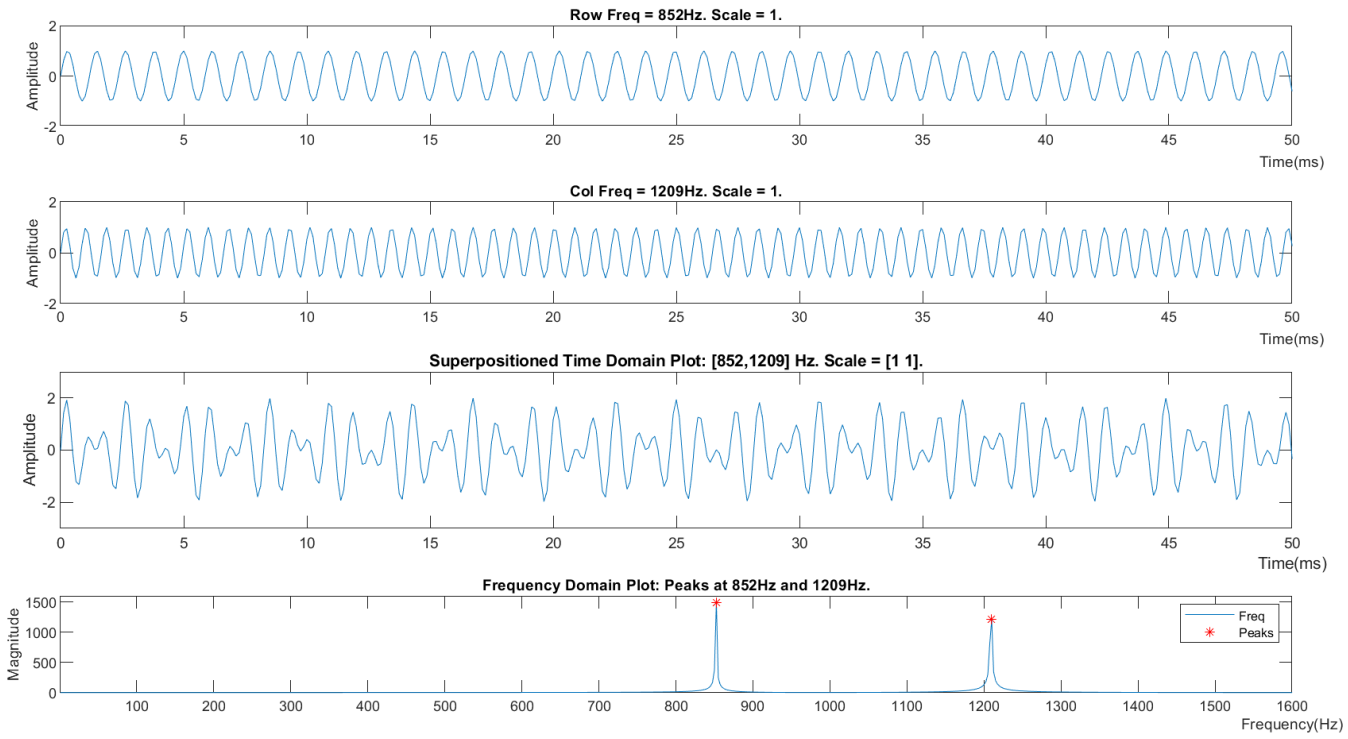
'5':


Time and Frequency Responses of tone: "5" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 770Hz. Scale = 1.
Col Freq = 1336Hz. Scale = 1.
Superpositioned Time Domain Plot: [770,1336] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 770Hz and 1336Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

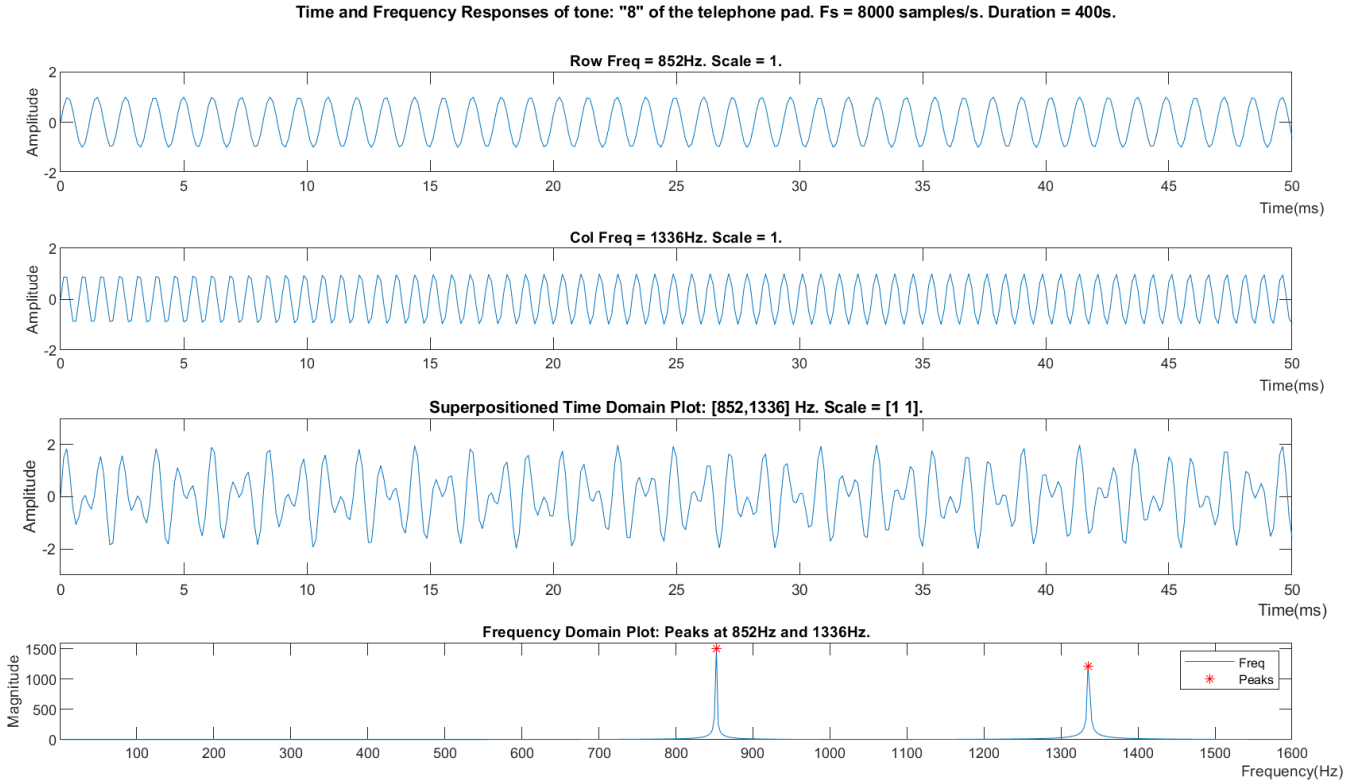
’6’:


Time and Frequency Responses of tone: "6" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 770Hz. Scale = 1.
Col Freq = 1477Hz. Scale = 1.
Superpositioned Time Domain Plot: [770,1477] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 770Hz and 1477Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

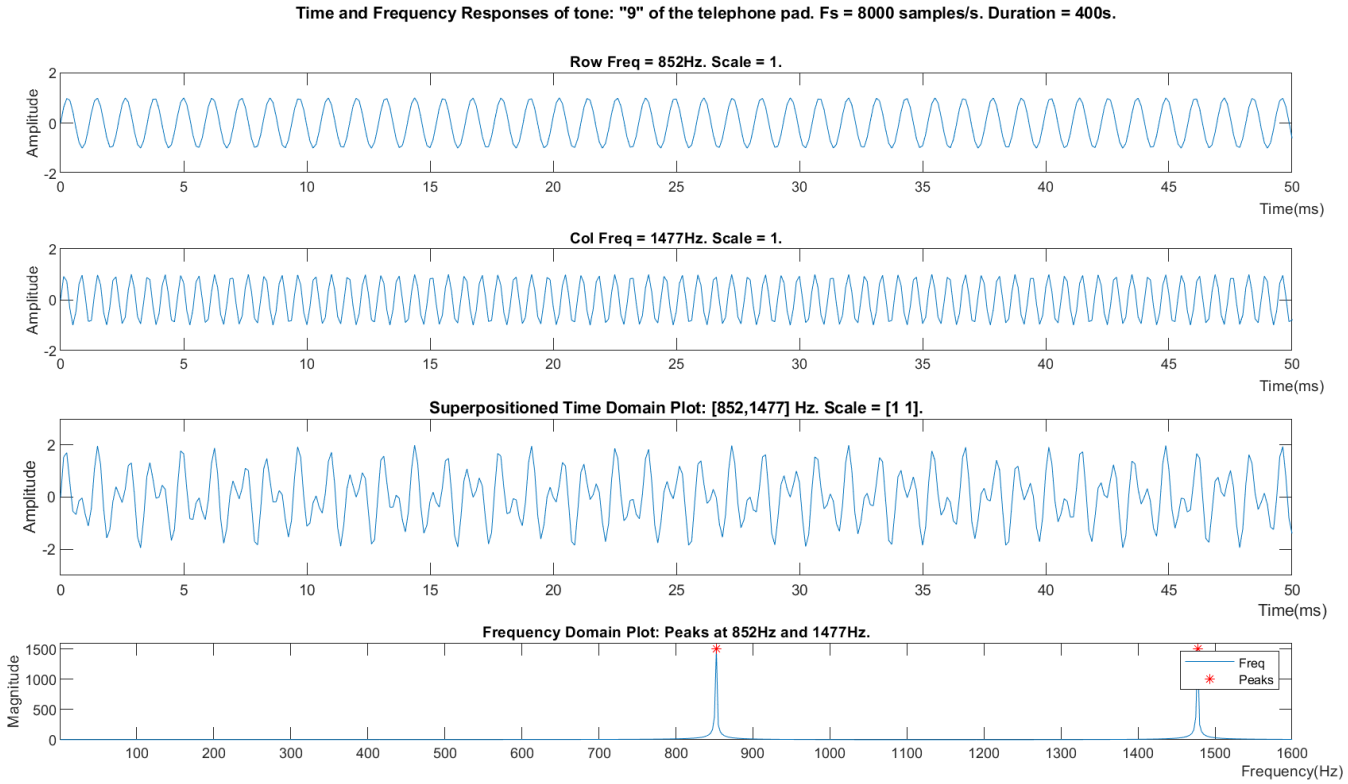
’7’:


Time and Frequency Responses of tone: "7" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 852Hz. Scale = 1.
Col Freq = 1209Hz. Scale = 1.
Superpositioned Time Domain Plot: [852,1209] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 852Hz and 1209Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

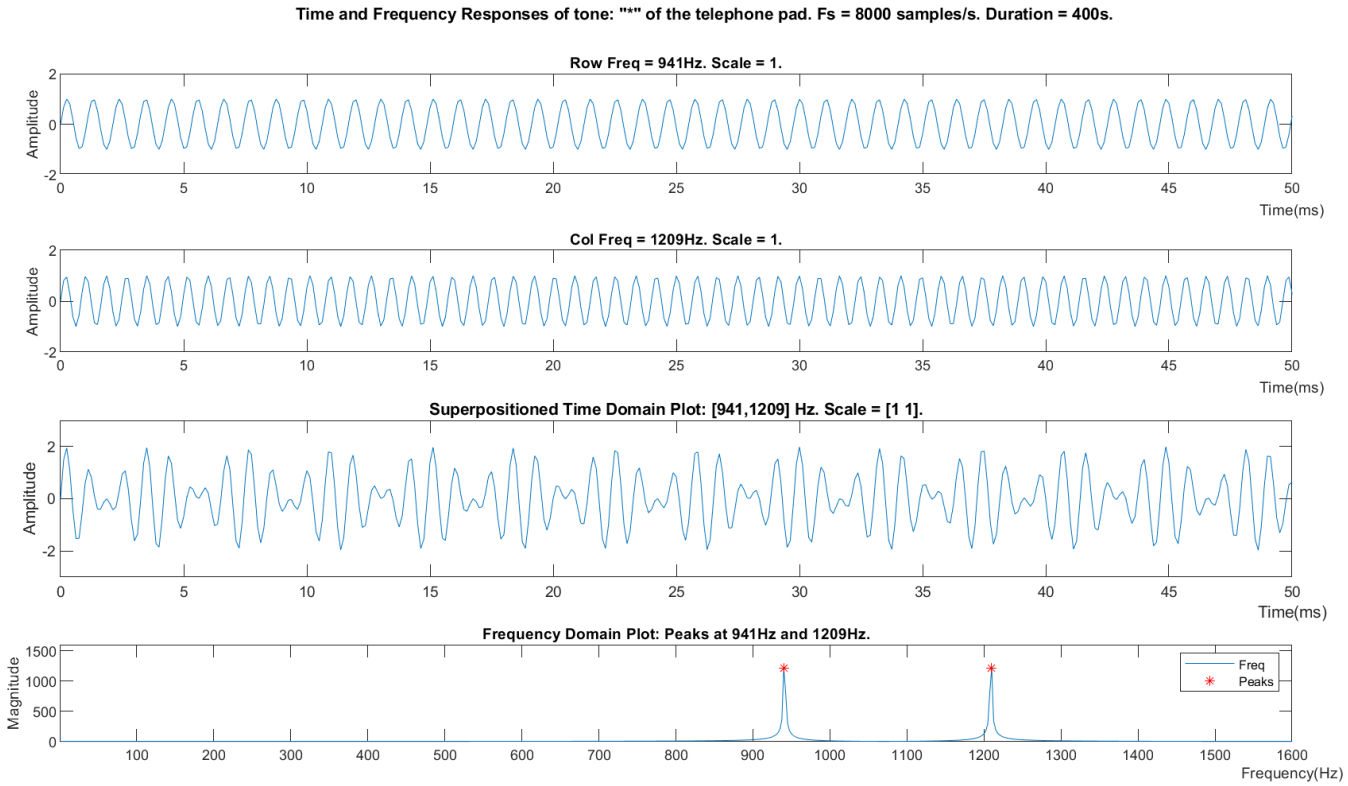
'8':


Time and Frequency Responses of tone: "8" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 852Hz. Scale = 1.
Col Freq = 1336Hz. Scale = 1.
Superpositioned Time Domain Plot: [852,1336] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 852Hz and 1336Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

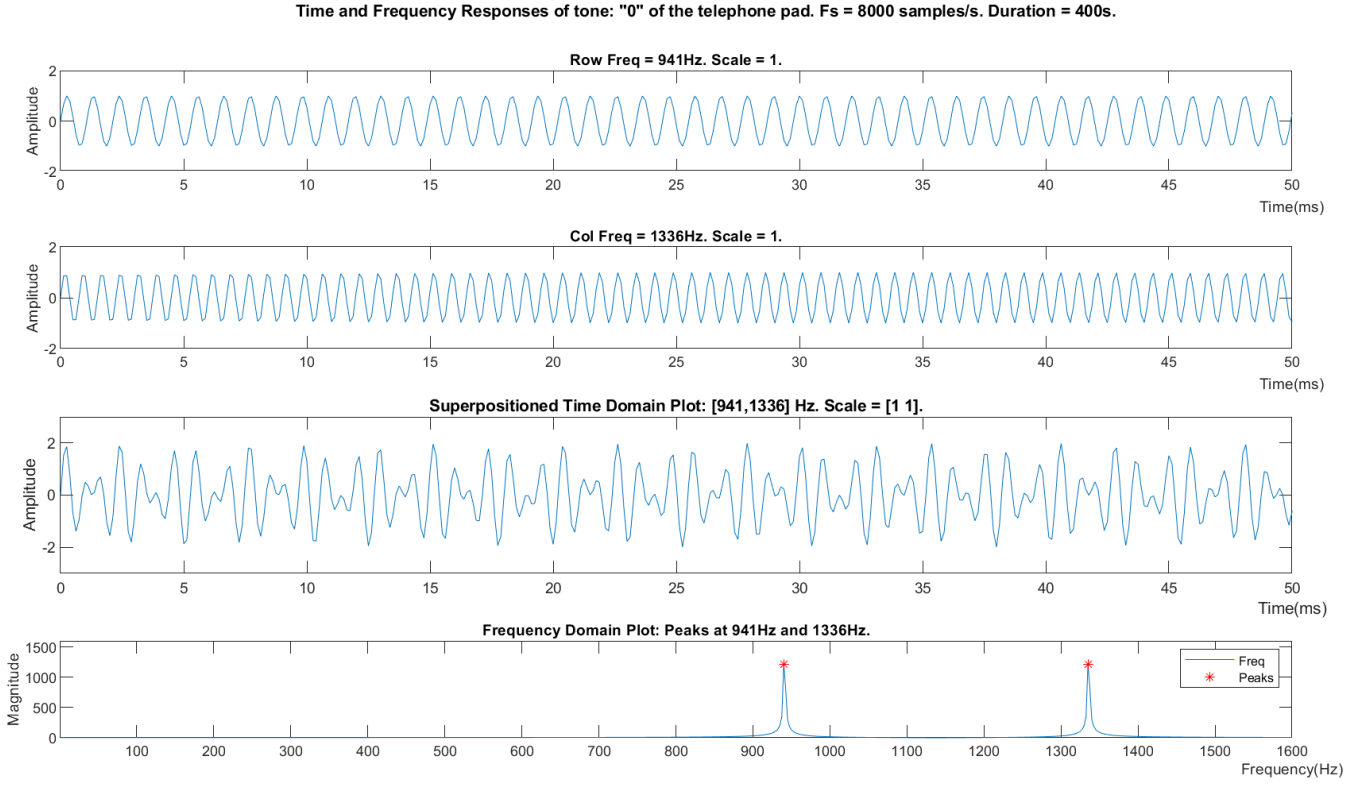
'9':


Time and Frequency Responses of tone: "9" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 852Hz. Scale = 1.
Col Freq = 1477Hz. Scale = 1.
Superpositioned Time Domain Plot: [852,1477] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 852Hz and 1477Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

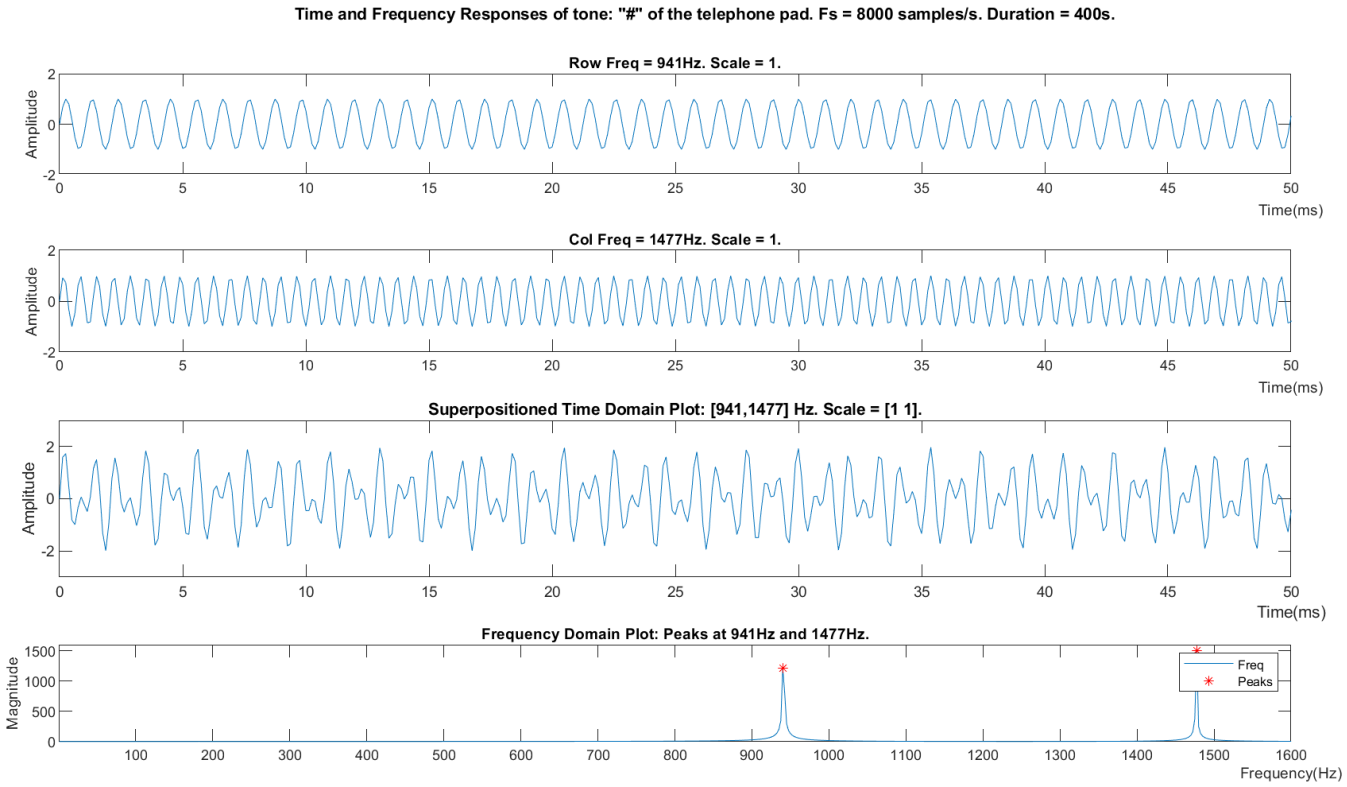
'*':


Time and Frequency Responses of tone: "*" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 941Hz. Scale = 1.
Col Freq = 1209Hz. Scale = 1.
Superpositioned Time Domain Plot: [941,1209] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 941Hz and 1209Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

'0':


Time and Frequency Responses of tone: "0" of the telephone pad. Fs = 8000 samples/s. Duration = 400s.
Row Freq = 941Hz. Scale = 1.
Col Freq = 1336Hz. Scale = 1.
Superpositioned Time Domain Plot: [941,1336] Hz. Scale = [1 1].
Frequency Domain Plot: Peaks at 941Hz and 1336Hz.
Amplitude
Time(ms)
Magnitude
Frequency(Hz)
Freq
Peaks

'#':

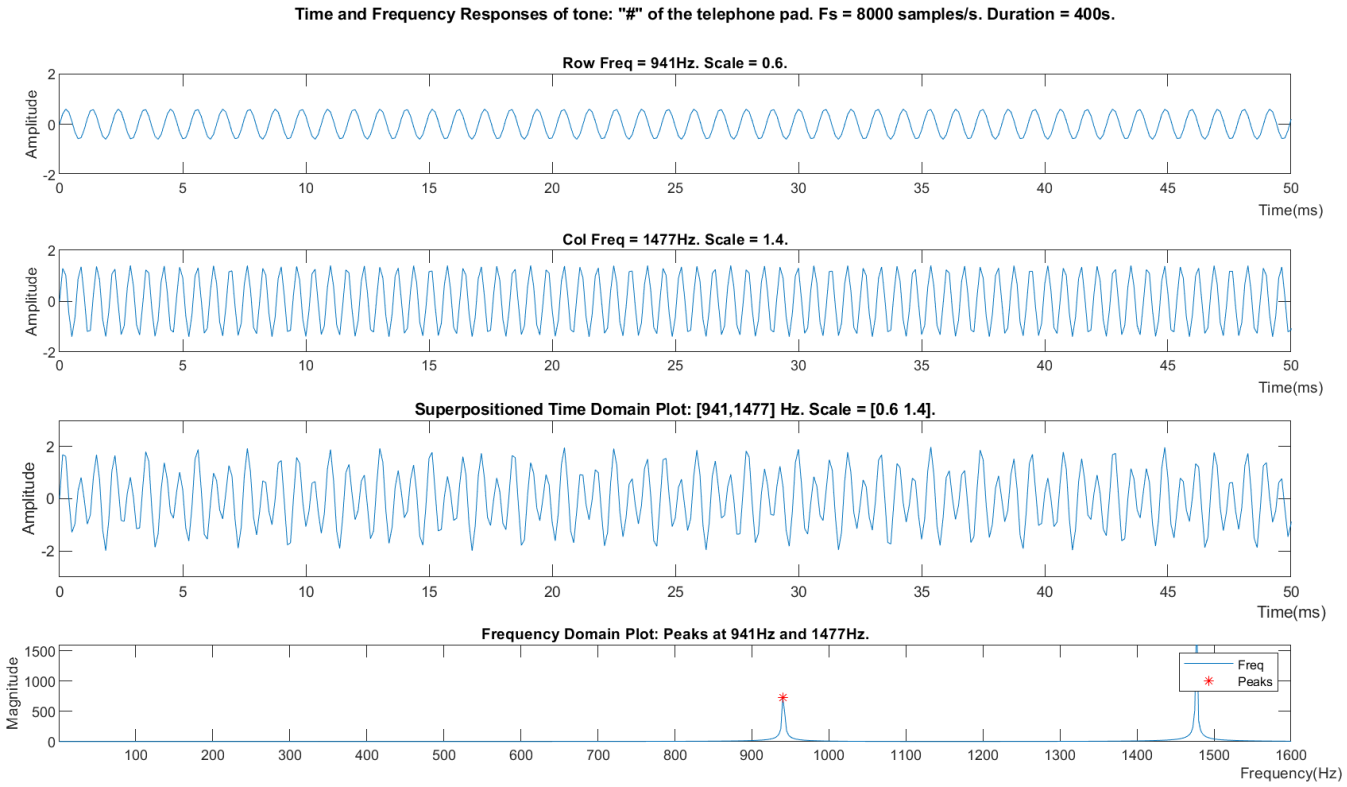


Here, we see that both input sinusoids have amplitudes of 1, the default value. Since the output is the superposition of these signals, the time-domain amplitude becomes higher. In the frequency domain plot, we can clearly see two frequency peaks for each digit that correspond to the desired frequency from fig 1.

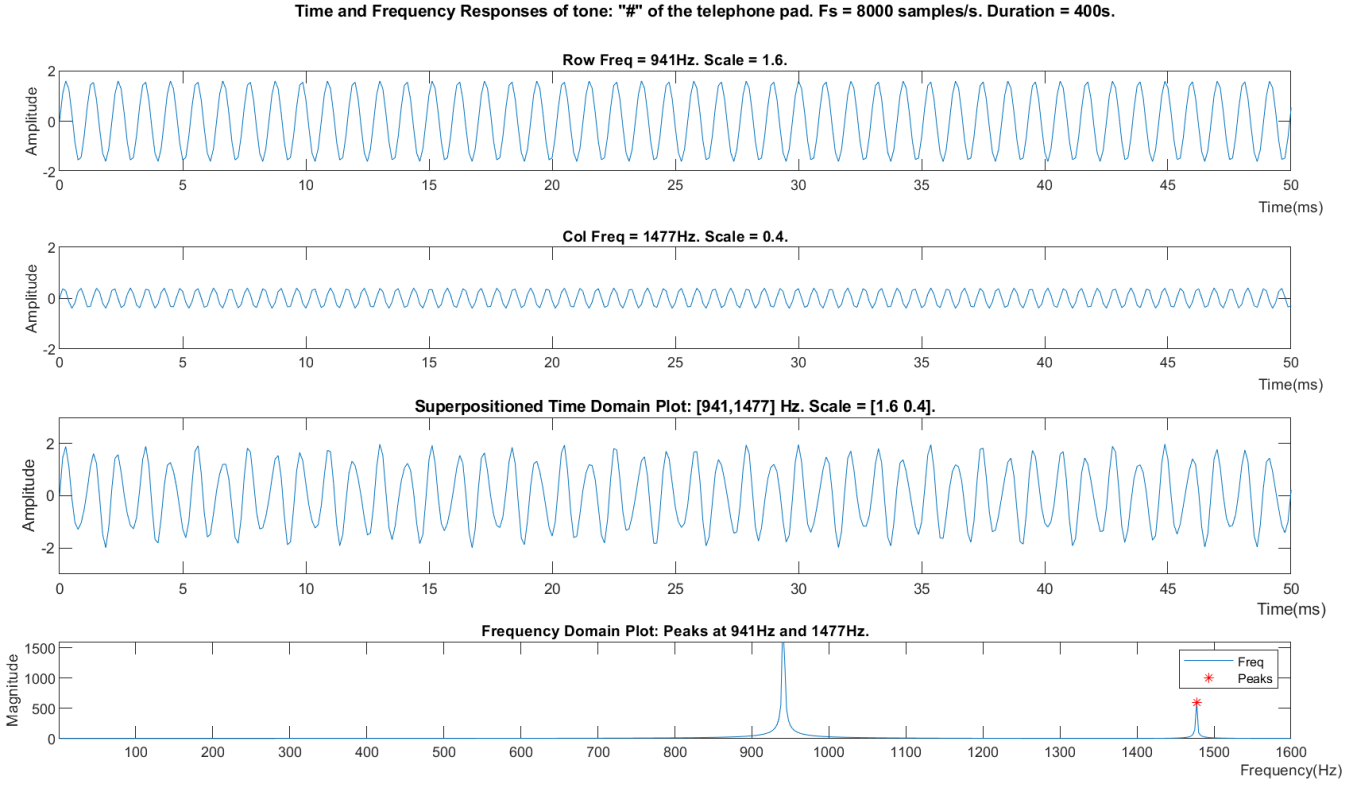
The DTMFencode() function can encode digits with different amplitudes, making one of the two more prominent than the other. When listening to the tones by ear, one can feel that the tones with heavier high-frequency weightings sound more crisp and high-pitched. Conversely, audio with heavier low-frequency weightings sound more muffled and deep. This verifies the function's implementation qualitatively.

Here, we illustrate an example of two weighting settings on digit 12, the '#' key. The first one emphasizes the high pitch, while the second emphasizes the low pitch.

'#' with weighting [0.6, 1.4]"



'#' with weighting [1.6, 0.2]"



Notice the different amplitudes of the base signals creating different output signals. However, no matter the weighting,the resulting frequency-domain peaks are still at the same frequency values (x-axis).

2.1.3 Testing Methods and Results

The DTMFencode() function is tested by:

1. Dialing numbers on a cellphone to hear the similarities between it and the MATLAB-generated audio.

2. Calling the function a few times on an actual phone number, on several repeated keys, etc. to ensure correctness of the output audio.

Note that to avoid audio clipping for the MATLAB audiowrite() function, the output x array is normalized to amplitude of 1, rather than the sum-of-sines output shown in the figures. The command for normalization is

$$x = x/max(abs(x))$$

The number of channels for audiowrite() is 16.

Now, we have an encoding function that allows us to create audio signals with any digit sequence, duration, frequency weighting, and sampling rates (>3000 Hz).

2.2 DTMF Decode

2.2.1 Implementation Overview

Implementation Steps

1. Read in audio signal + handle input errors

2. Take signal's Fourier transform with FFT

3. Plot the frequency peaks in the range 0 - 1600 Hz, since all the target frequencies are in this range.

4. Use maxk() to find the maximum values of the FFT amplitudes and find their locations on the frequency axis.

5. Compare the 2 maximum frequencies to each of the 12 target frequency pairs (each key has 2 frequencies)

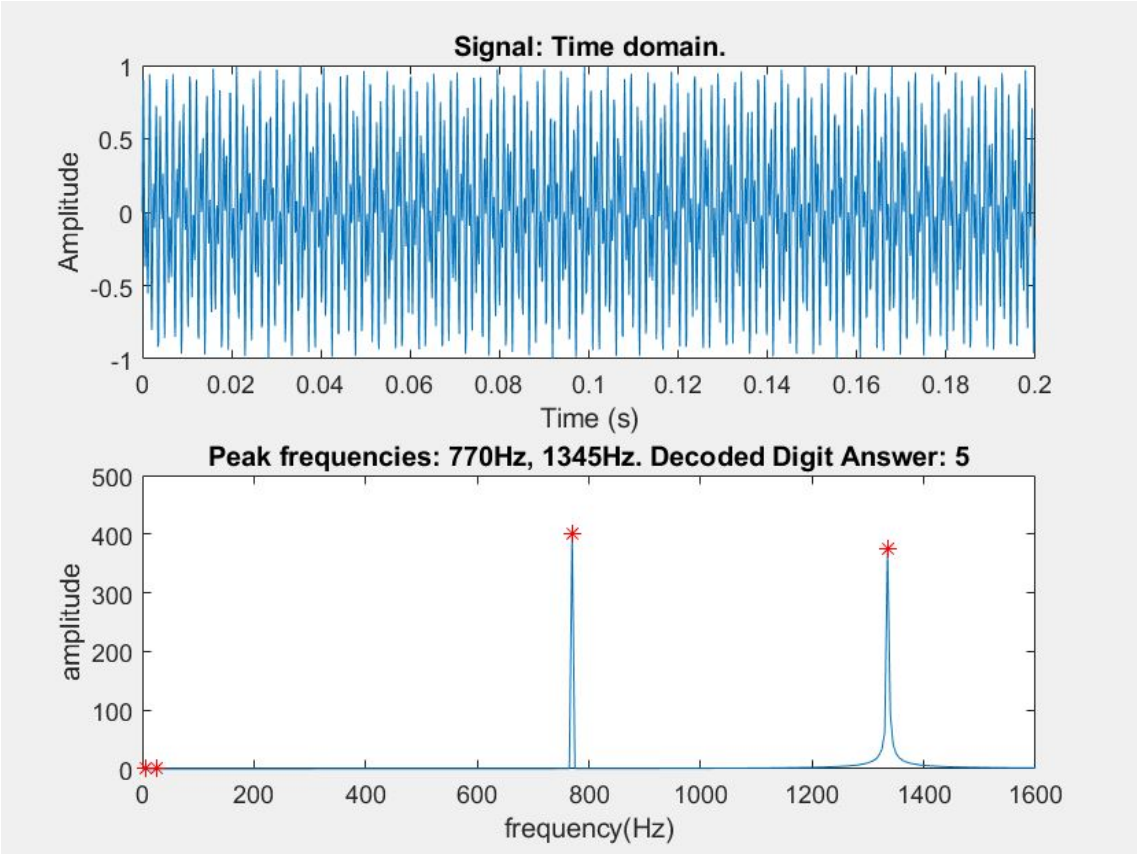
6. Calculate the mean square error between the current frequency and the target pairs. Find the key with the smallest MSE value, then assign it to the signal.

2.2.2 Sample Run

Example 1: Decoding Default Key #5

This section demonstrates the workflow of DTMFdecode(), which decodes an input .wav file of a key press of key '5'. It has 200 ms duration, [1 1] weighting, and 8000 Hz sampling rate.

'5':



The red asterisks are the local maximums of the frequency-domain amplitude signal, identified with the MATLAB maxk() function. Note that from the frequency table, key '5' is composed of 770Hz and 1336Hz signals; after decoding, our maximum frequency bands are 770Hz and 1345Hz, which are slightly different from the original.

Even though there are small deviations between encode() and decode(), the decodeDTMF() function is able to identify one out of 12 keys based on the mean square error.

Variables - MSE

MSE

1x12 double

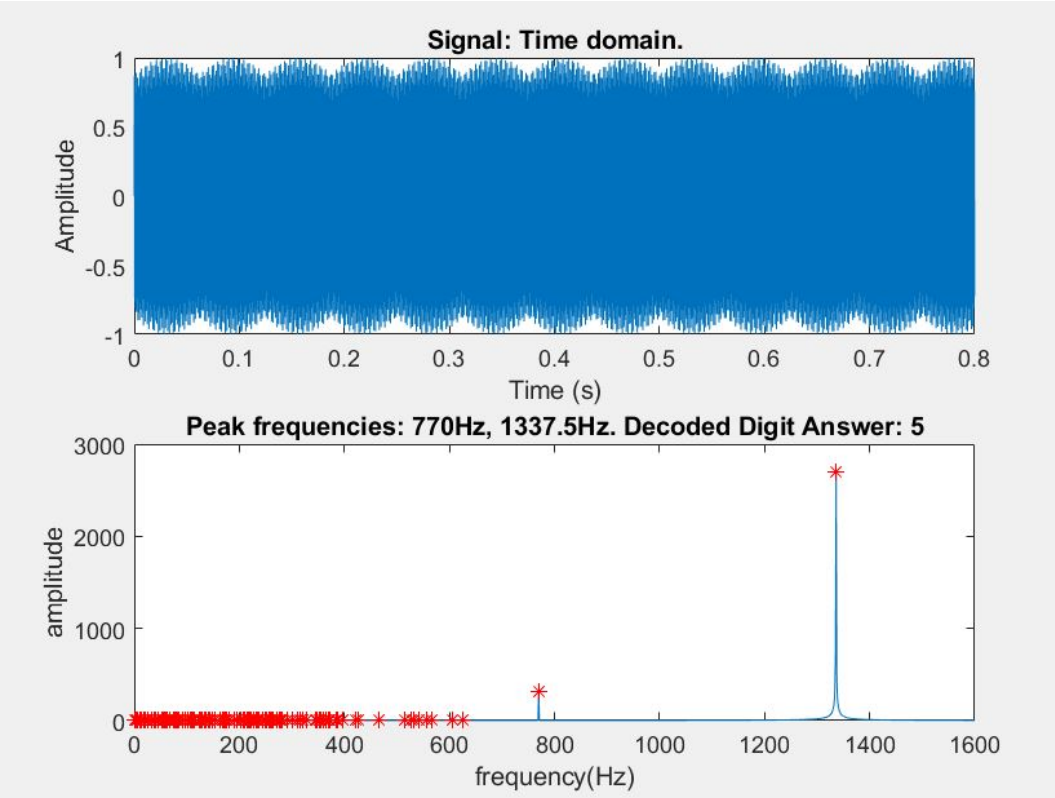
	1	2	3	4	5	6	7	8	9	10	11	12
1	1.1913e+04	2705	1.1377e+04	9248	40.5000	8712	12610	3.4025e+03	1.2074e+04	2.3869e+04	14661	2.3333e+04

When looking at the MSE array result of key '5', where the MSE value between the input and all keys' target frequencies are calculated, we see huge differences between the correct match and other matches. Therefore, a min() command can directly output the correct key.

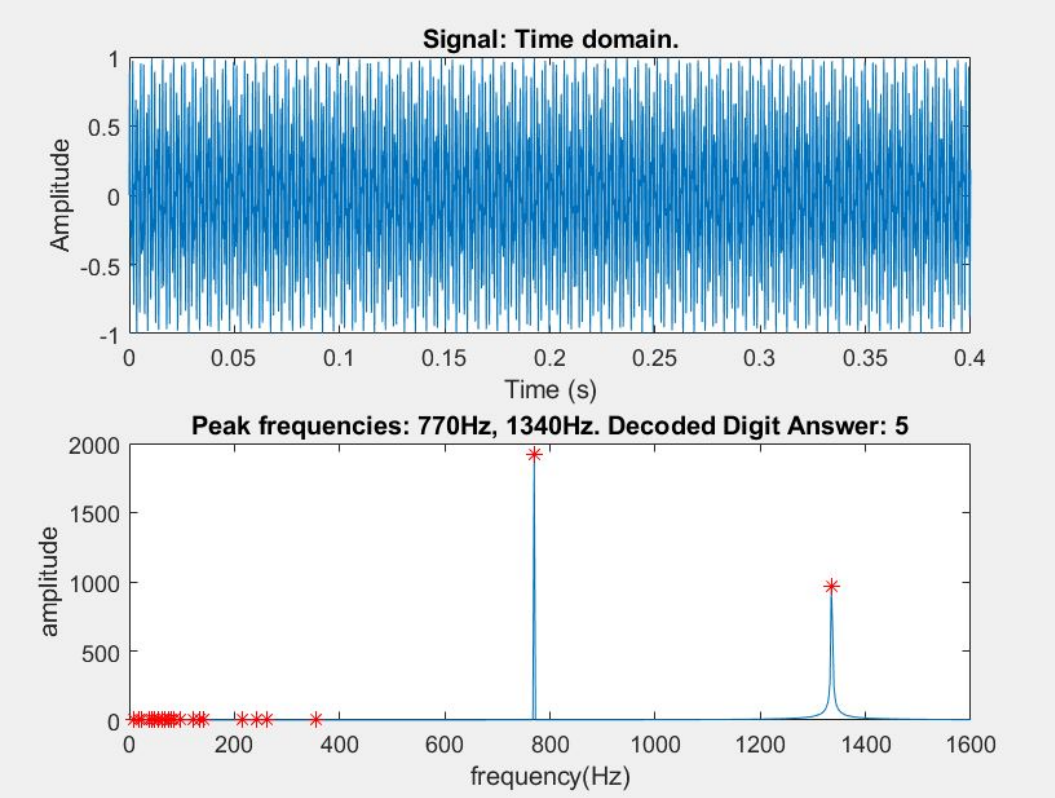
2.2.3 Testing

To test the robustness of this function, non-default .wav files are tested. Here are some more examples of the '5' key encoding with their respective parameters:

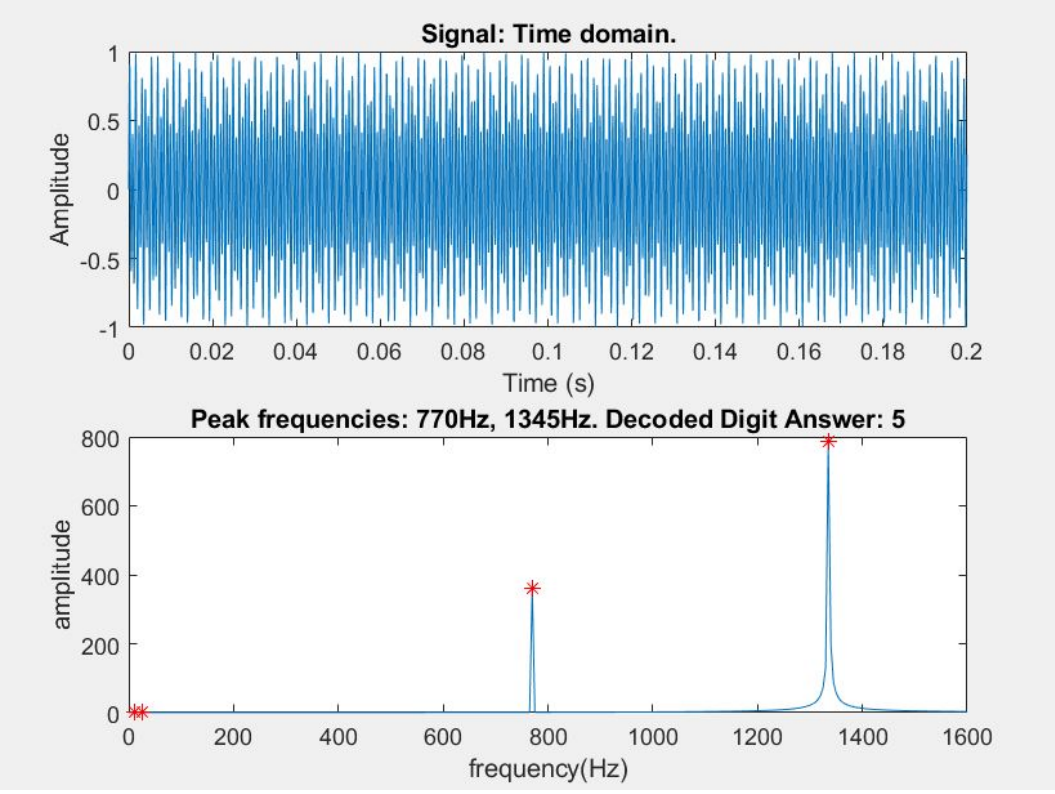
Duration = 800 ms, weighting = [1 9], fs = 8000 Hz



Duration = 400 ms, weighting = [6 4], fs = 16000 Hz



Duration = 200 ms, weighting = [3 7], fs = 12000 Hz



The function was able to decode a specific digit with various audio parameters. This works across all twelve digits. Although the amplitude and precise frequency peak locations are slightly different across trials, the general peak locations are still close enough to the frequency table that we can identify the correct answer.

2.3 Decoding DTMF Sequences

2.3.1 Implementation Overview

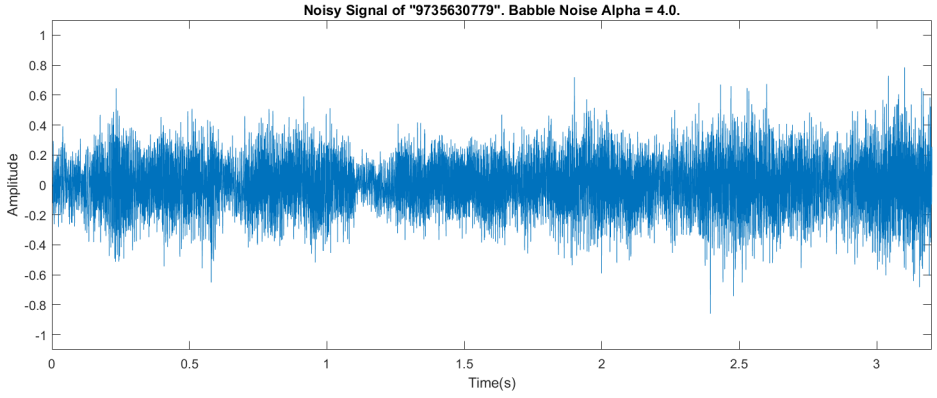
Implementation Strategy

1. The DTMFsequence() function uses band-pass filters on the original signal to filter out frequencies not in the phone key ranges
2. Then, split the total signals into "blocks" of 80 samples each.
3. The total signal energy of each block is calculated.
4. If significant energy is detected across a number of blocks consecutively, a digit exists, and that piece of the signal is passed into DTMFdecode().
5. Finally, concatenate each digit's result to output the entire sequence.

Problem Analysis and Strategy Selection

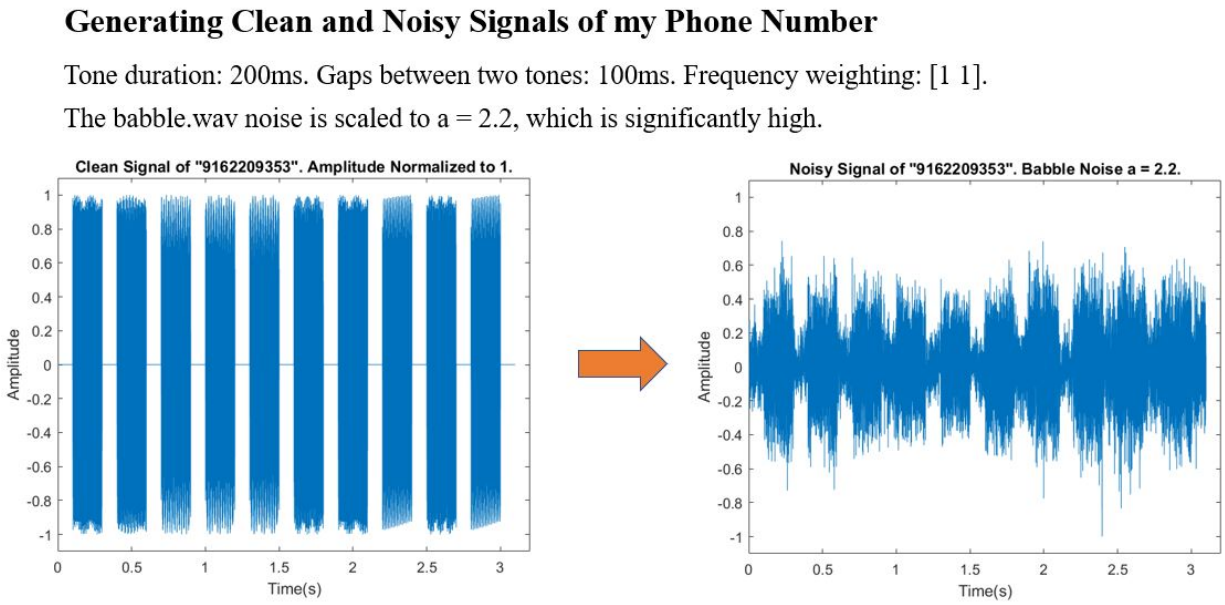
The main challenge with an arbitrary audio sequence is to extract the actual key presses from the noisy signal when the tone duration, weighting, and gaps are known. From lots of code performance testing, it is found that the digit audio extraction step is essential to the robustness of sequence decoding.

A typical noisy signal looks like this:



At high levels of noise (alpha = 4 times louder of the original noise), it's very difficult to tell digits apart in the time-domain.

Another illustration of generating additive noisy signal:



We see the noisy values having smaller overall amplitudes, ranging from 0.6 - 0.8 instead of 1.0. This is because both the clean and noisy signals are normalized to amplitudes of 1 and then added together. So the original output noisy signal (N1) would have larger maximum amplitudes up to 2. When we normalize this signal (N1 = N1/max(abs(N1))) again, the majority of the amplitude ranges will be smaller than 1, since the extreme values normalize and compress the other values.

This observation is valid across all the generated noisy signals, posing extra challenges for decoding, since 1) audio amplitudes are attenuated 2) digits are hard to distinguish themselves from the noise.

To identify where each signal starts and ends, the following approaches are attempted:

1. After using a band-pass filter and killing minuscule signals, directly determine the digit range from amplitude differences - find places of sudden amplitude increases and decreases to extract the range.
2. Decode every block of size 80 and generate outputs based on the outputs that oc-

cur the most often repeatedly. For example, if we get "???234444?????1111?2?????8 888878??", the output is "418".

3. The final implementation: Using energy bands to extract ranges.

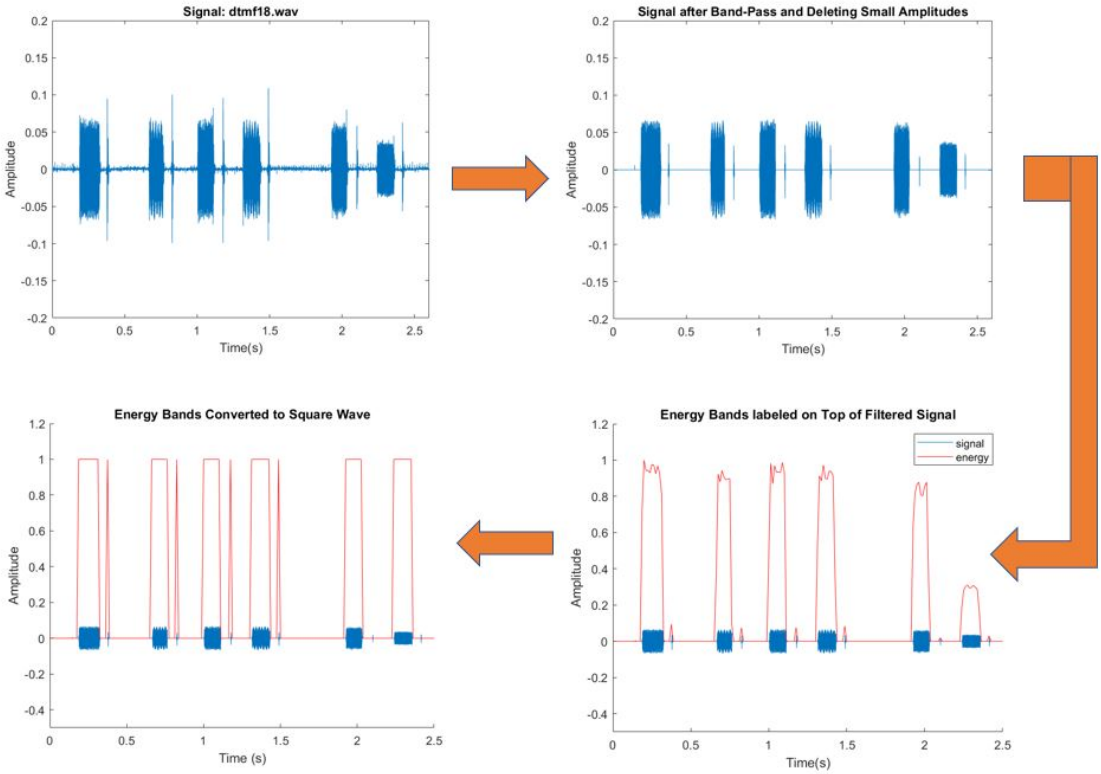
The third method perform much better than the first two because of several factors that are important to successful decoding:

- **Correct starting and ending points of the digit audio.** At high levels of noise, the noise amplitudes are similar to that of the digits. If we extract all amplitudes above a threshold and feed them into DTMEdecode(), it's likely to mistaken noise as digit audio and obtain wrong results.
- **Large enough sample size for each audio to decode.** The more samples of a digit fed into DTMFdecode(), the easier the code gives the correct output, as the frequency band spikes are more accurate
- **The correct number of digits.** Using energy bands is the best approach to mark digit ranges. By filtering out undesired energy ranges, ie. removing bands too short in duration, the number of total digits read are much more accurate.

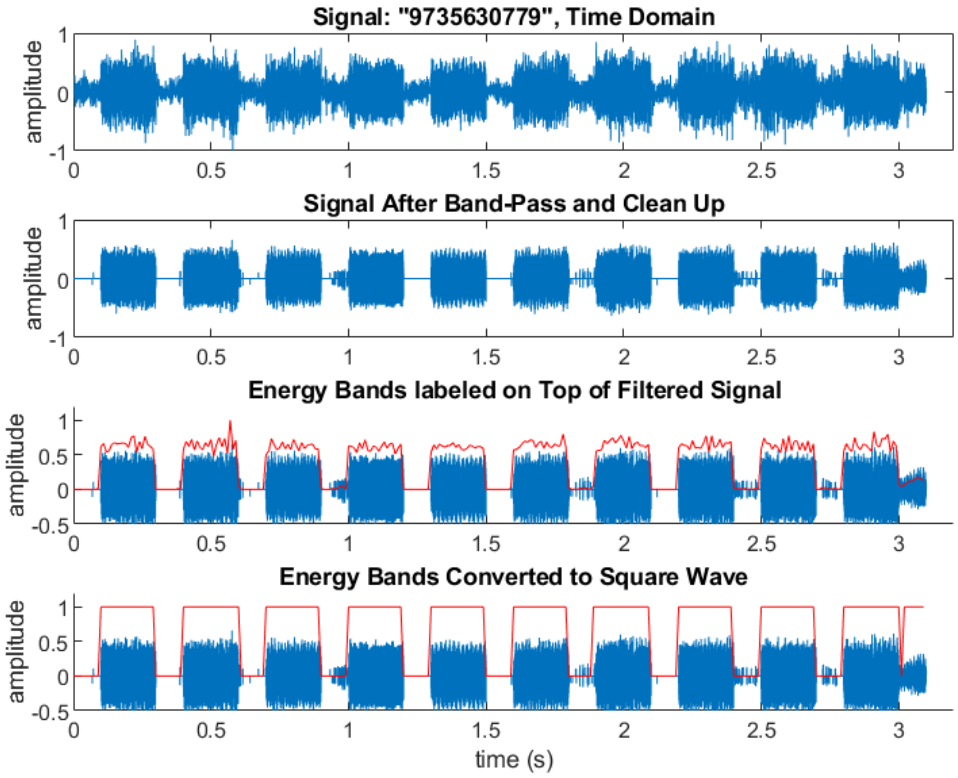
2.3.2 Example: Decoding dtmf18.wav as "121285"

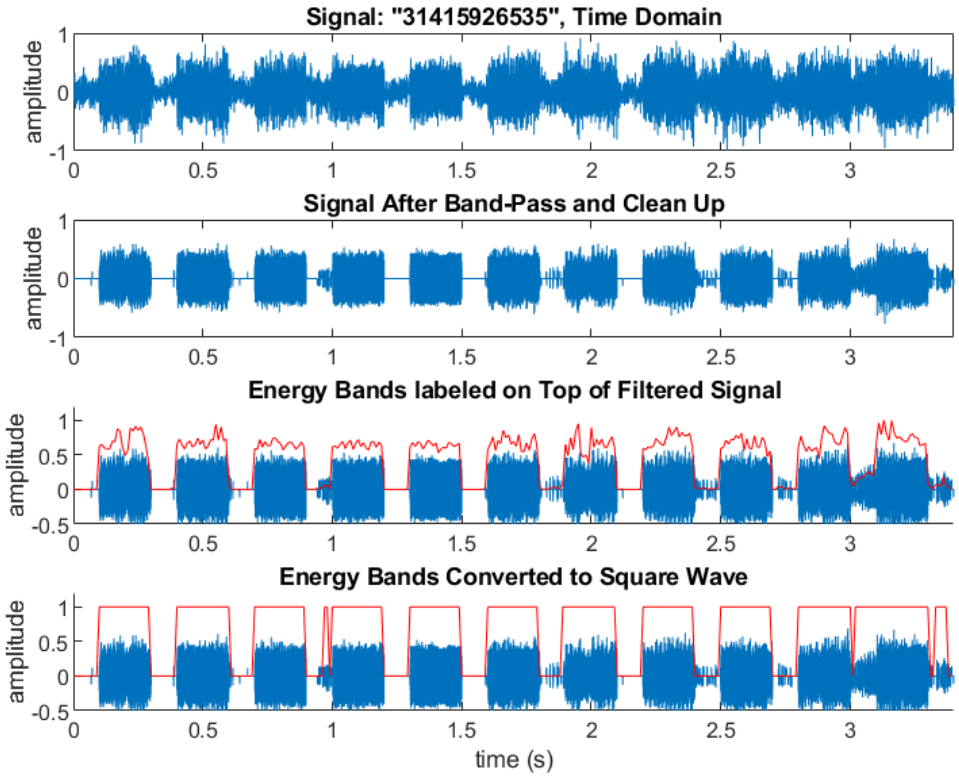
Now, we demonstrate the signal processing of "dtmf18.wav", which is a sequence recorded in an office setting.

1) Use a band-pass filter to filter out frequencies not between 674Hz and 1490Hz.
2) Calculate and plot energy bands of each block, normalized.
3) Turn the energy bands into square waves at a threshold of energy = 0.05.
4) Disregard small spikes with start and end indices < 3 blocks, since such short ranges are clearly not digits.
5) The signal under each square wave pulse is passed into DTMFencode().
6) the decoded message is of length 6, answer is "121285".



More examples:





Both signals are generated with babble noise (alpha = 1.9) and are successfully decoded.

2.3.3 Results with Sample dtmf Audio

The following are the decoded results of the DTMF audio samples from 1 - 20:

File #	Result	File #	Result
1	182846	11	6619
2	31415	12	12832
3	271827	13	20164
4	8548928	14	196509
5	926535	15	110405
6	8548928	16	20001
7	8548928	17	1071135
8	20001	18	121285
9	8548928	19	48928151
10	12859	20	1965

These results were compared and confirmed from multiple peers in this course.

2.4 Decoding Sequences in Open Environments

2.4.1 Test Setting

As shown in the previous section, the DTMFsequence() function is robust under various digit duration and noise level conditions with the energy band method.

This section uses 50 randomly-generated noisy audios with the babble noise. We examined the error tolerance of the decoding function under various levels of alpha (noise scaling).

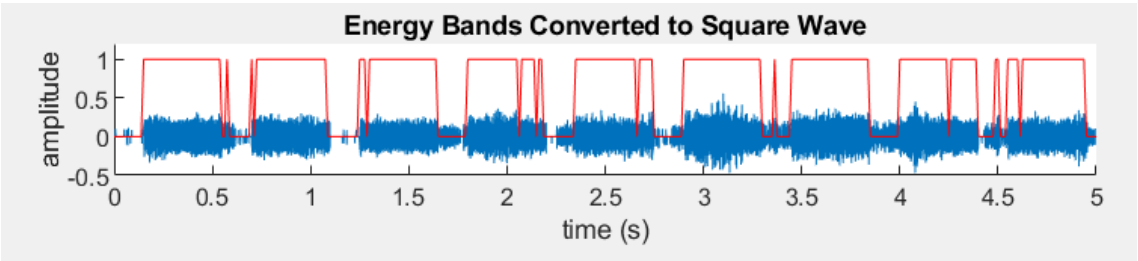
Alpha is incremented from 0 to 3 with steps of 0.1.

2.4.2 Result

All test results are included in "alpha_accuracy_tests.csv". At low levels of noise, the decoded results are highly accurate.

	A	B	I	J	K	L	M
1	Trial#	Input Sequence	a = 0.6	a = 0.7	a = 0.8	a = 0.9	a = 1
2	1	659811	659811	659811	659811	659811	359811
3	2	7197	7197	7197	7197	7197	7197
4	3	8995	8995	8995	8995	8995	8995
5	4	155145	155145	155145	155145	155145	1551455
6	5	6035	6035	6035	6035	6035	6035
7	6	63196	63196	63196	63196	63196	631965
8	7	6785138	6785138	6785138	6785138	6785138	6785138
9	8	3982	3982	3982	3982	3982	3982
10	9	44414692	44414692	44414692	44414692	44414692	44414692
11	10	22104	22104	22104	22104	22104	22104
12	11	8599	8599	8599	8599	8599	8599
13	12	1958460	1958460	1958460	1958460	1958460	1958460
14	13	617302227	617302227	617302227	617302227	617302227	617302227
15	14	55696695	55696695	55696695	55696695	55696695	55696695

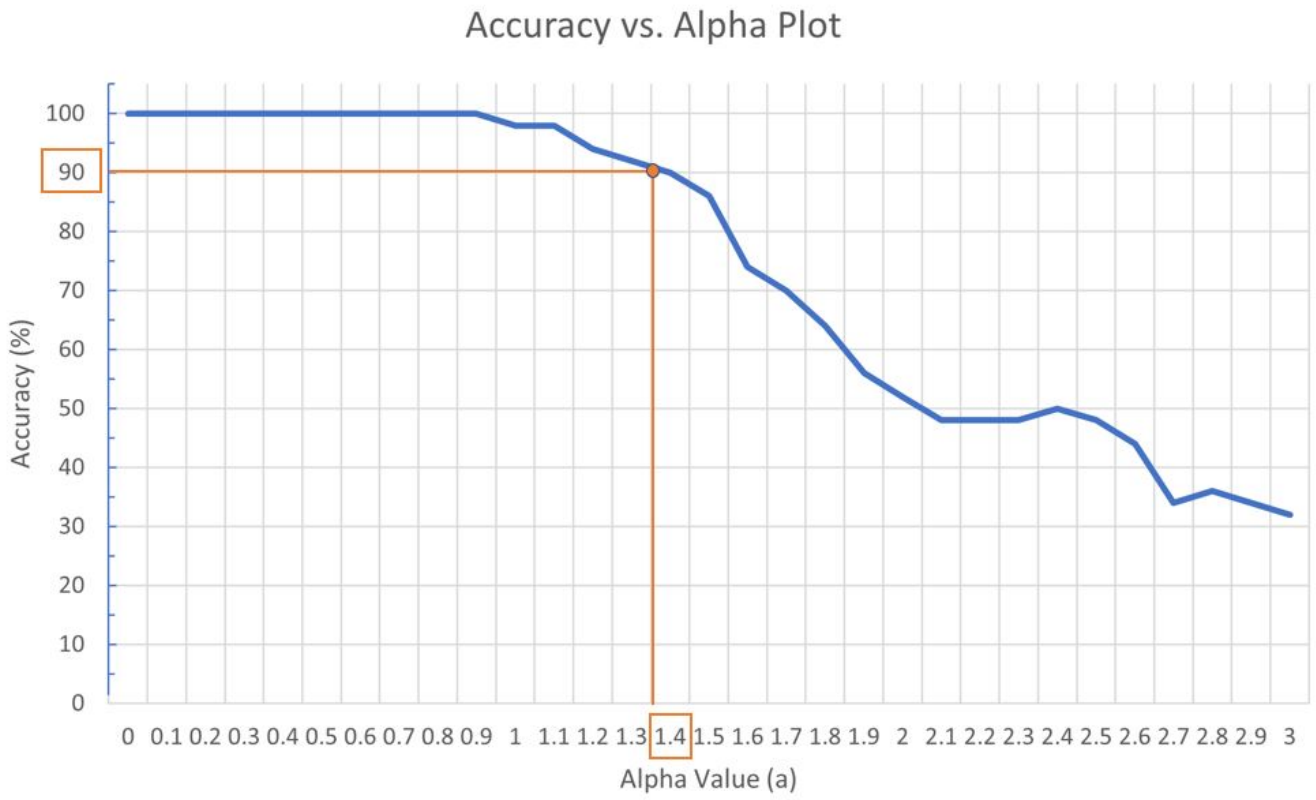
However, at higher values of alpha, the noise has such high amplitudes that it mixes up with the actual signal. Therefore, several more energy bands occur and misidentifies signal ranges. An example of alpha = 3 is shown below:



This results in incorrect digit reads and reading more digits than the sequence length. Some results illustrate the deteriorating effects:

	A	B	Y	Z	AA	AB	AC	AD	AE	AF	AG
1	Trial#	Input Sequence	a = 2.2	a = 2.3	a = 2.4	a = 2.5	a = 2.6	a = 2.7	a = 2.8	a = 2.9	a = 3
2	1	659811	359811	359811	359811	359811	359811	359811	359811	359811	359811
3	2	7197	7197	7197	7197	7197	7197	7197	7197	7197	7197
4	3	8995	1819995	1819995	1819995	1819995	1819965	1819955	1819955	1819955	1819955
5	4	155145	15514	15514	15514	15514	15514	15514	15514	115514	115514
6	5	6035	6035	6035	6035	6035	6035	6035	6035	6035	6035
7	6	63196	631296	631296	631296	631296	631296	1631296	1631296	1631296	1631296
8	7	6785138	3785132	3785132	3785132	3785138	3785138	3785138	3785138	3785138	3785138
9	8	3982	3982	3982	3982	3982	3982	3982	39821	39821	39824
10	9	44414692	14414692	14414692	14414692	14414692	14414692	14414692	14414692	14414692	14414692
11	10	22104	221045	221045	221045	221045	221045	221045	221045	221045	2210445
12	11	8599	8599	8599	8599	8599	8599	8599	8599	8599	8599
13	12	1958460	1958460	1958460	1958460	1958460	1958460	1958462	1958462	1958462	1958462
14	13	617302227	6.17E+12	6.17E+12	6.17E+12	6.17E+12	6.17E+12	6.17E+12	6.17E+13	6.17E+12	6.17E+12
15	14	55696695	5569665	5569665	5569665	5569665	5569665	5569665	5569665	2569665	2569665
16	15	94676	34676	34676	34676	34676	34676	34676	34676	34676	34676

2.4.3 Accuracy VS. a Plot



Note that **a result is only considered "correct" if the output matches exactly**. In the excel file, most digits, even at a = 3, are still correct. But for ease of computation, we only consider entirely correct results as a correct trial.

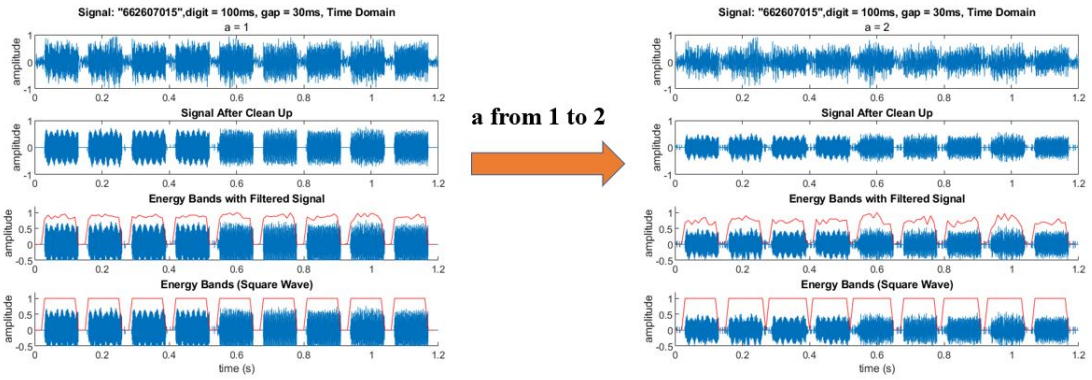
Still, the DTMFsequence() is robust and can still decode 32% sequences correctly at alpha = 3. Out of 1500 trials total, the accuracy remained above 90% at alpha < 1.4.

2.4.4 Observations and Case Studies

The following observations are made from the test results and previous testing:

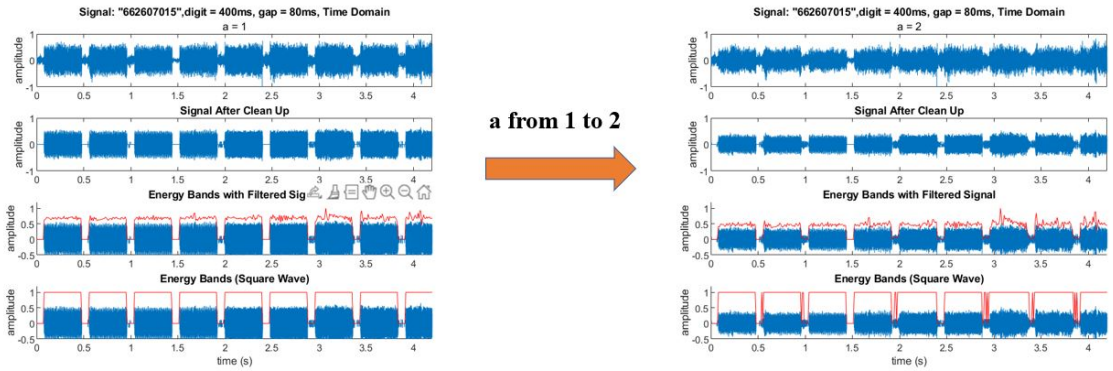
1. Longer key presses and gaps between digits create better results. When the keys last longer and are further apart, it's easier for the code to identify distinct digits and decode correctly.

To illustrate this effect, here's an attempt to decode the first few digits of Plank's Constant. Trial 1 has short keys with digit duration 100 ms and gaps of 30 ms. When Alpha value went from 1 to 2, the digit energies starter overlapping (as shown by the red square waves), causing some digits to be read incorrectly.



A = 2, result was incorrect: "662607015" became "632628016"

In trial 2, all key and gap duration are longer. Since the energy concentrated areas are further apart, the audio is more resistant to noise. At alpha = 2, the decode was still successful.



A = 2, result was still correct.

2. Frequency weighting doesn't seem to affect accuracy by much. The weightings tested across the 50 samples don't have extremely biased weightings; for example, a 1:1.8 weighting has the same accuracy compared to 1:1.

3. Incorrect results arise from misaligned digit signal ranges. As alpha increases for the same key sequences, we see that the number of "adequately high" energy bands increase in quantity and length, meaning that high-amplitude noise is being mistaken for actual tones. In the time domain, we can see significant mixing between the noise and tones when alpha > 3.

This reflects in the frequency domain as rougher and shorter frequency peaks in DTMFdecode(), meaning that Fourier Transform alone is no longer robust to tell the prominent frequencies of the audio. Therefore, the decode answers has more errors.

a = 0, outputs '123456789'

a = 1, outputs '123456789'

a = 2, outputs '123456789'

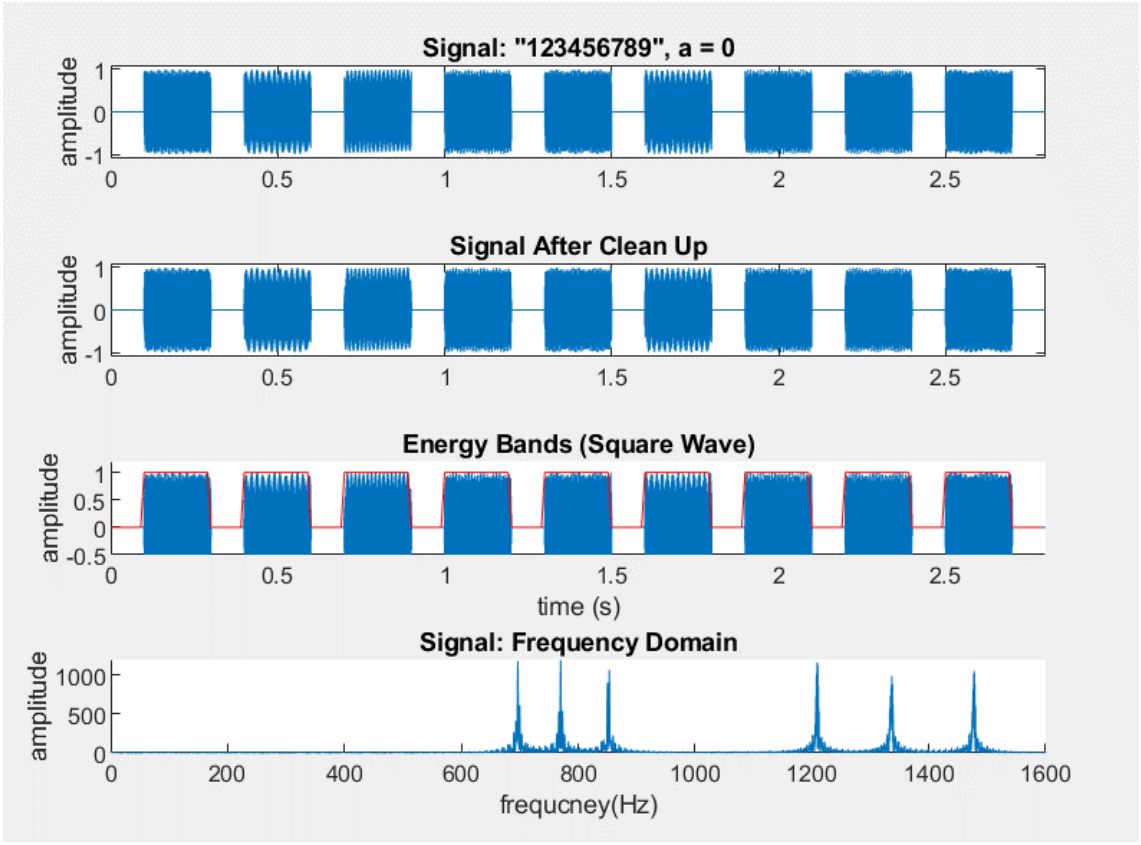
a = 3, outputs '123453786'

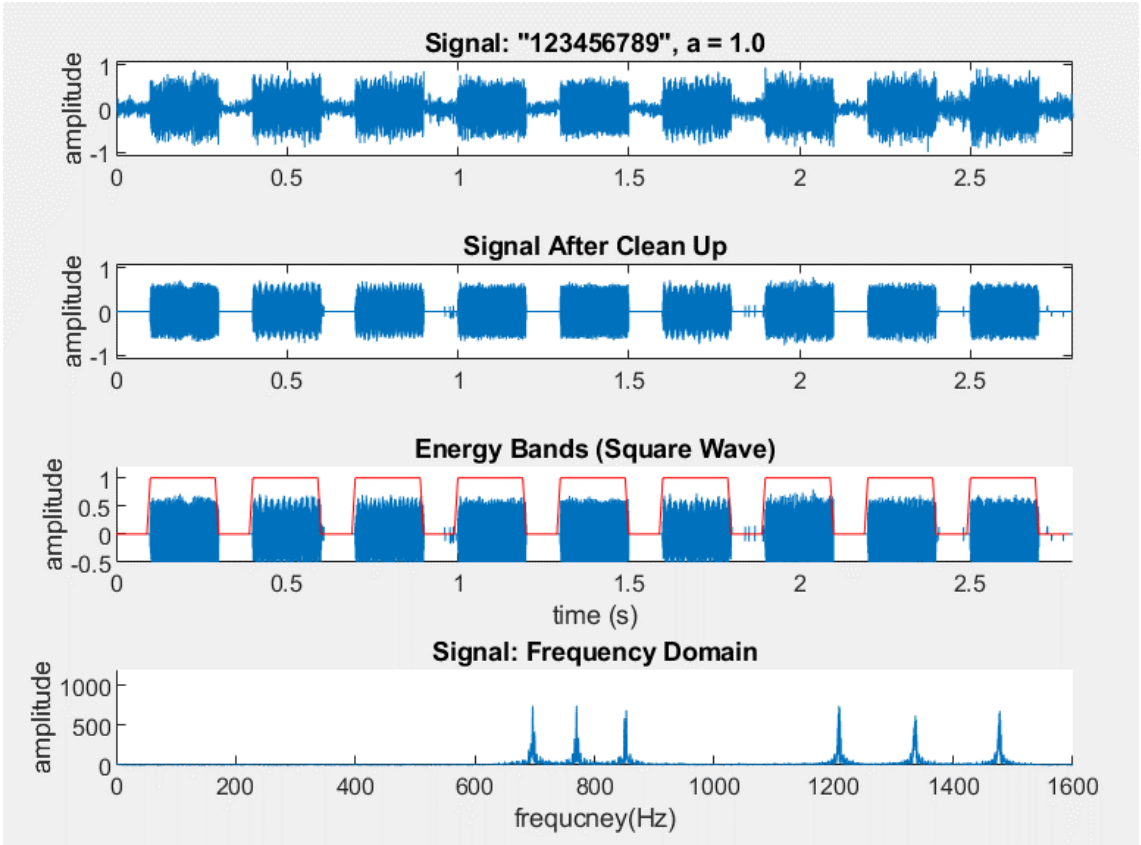
a = 4, outputs '1234567863'

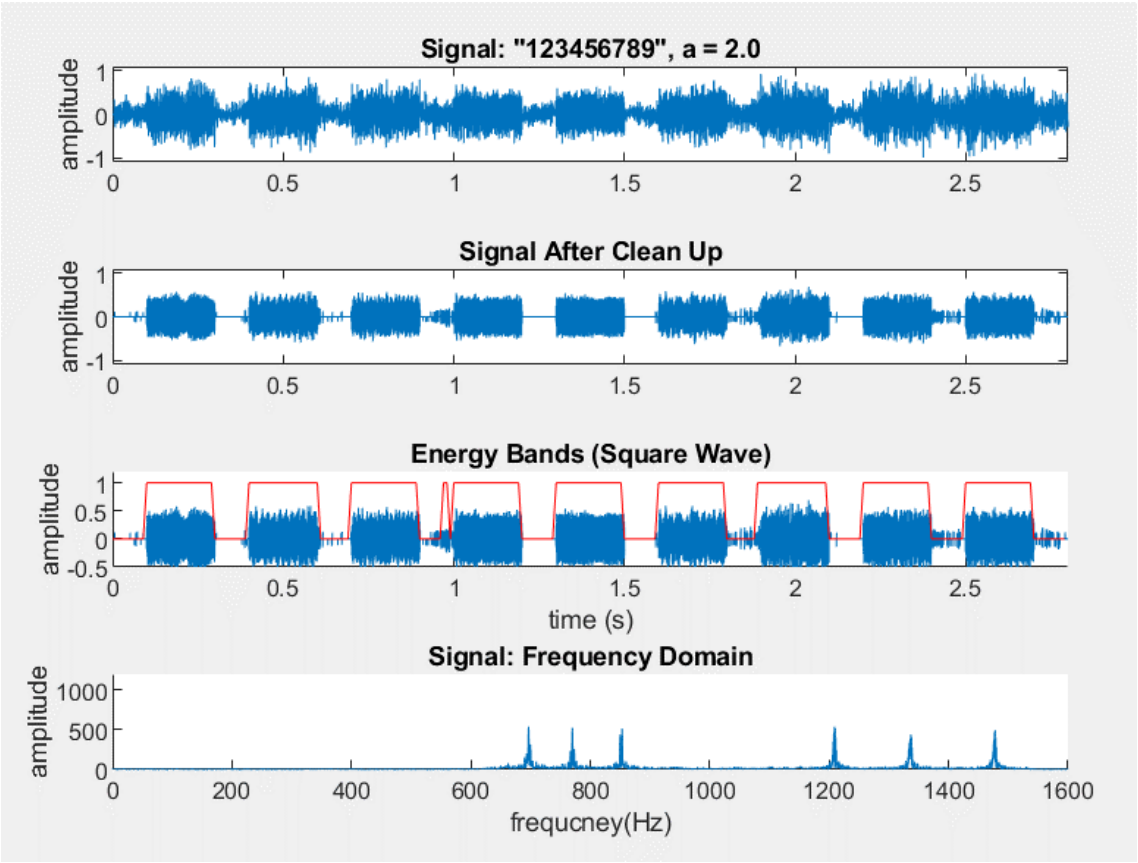
a = 5, outputs '123453793'

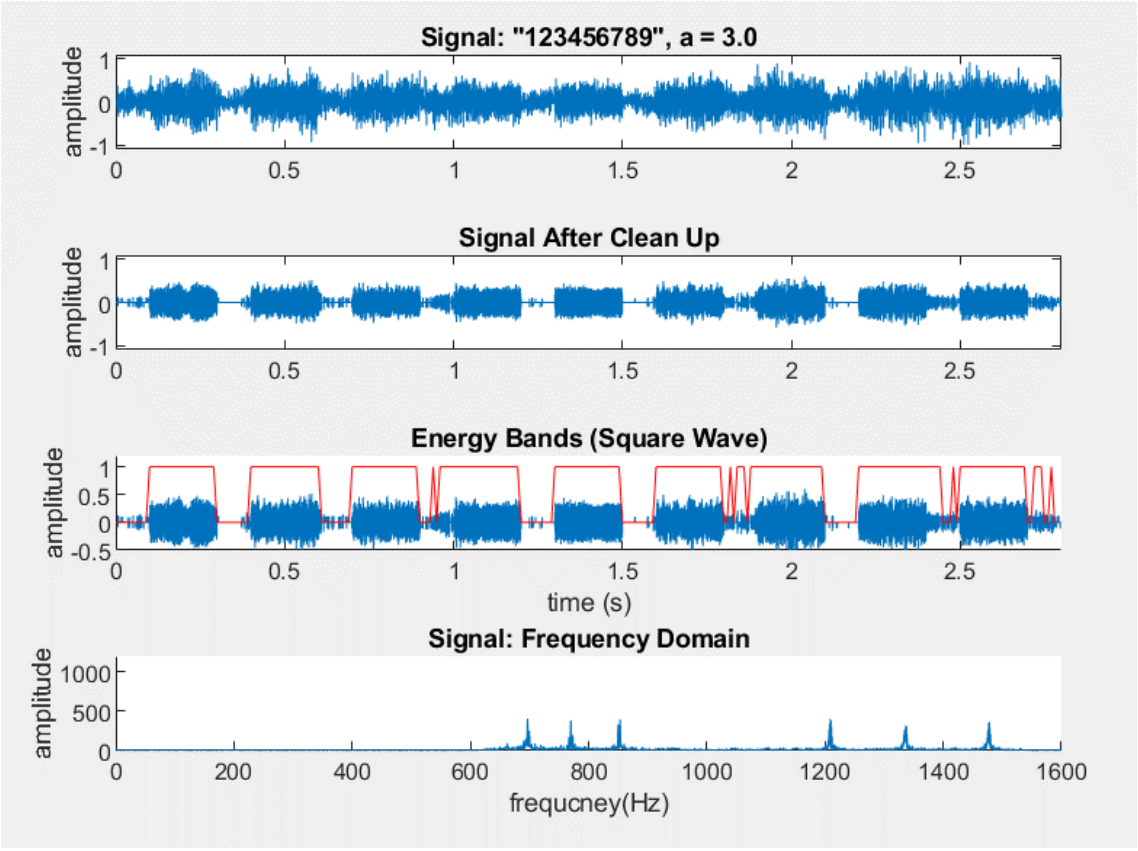
a = 7, outputs '123459'

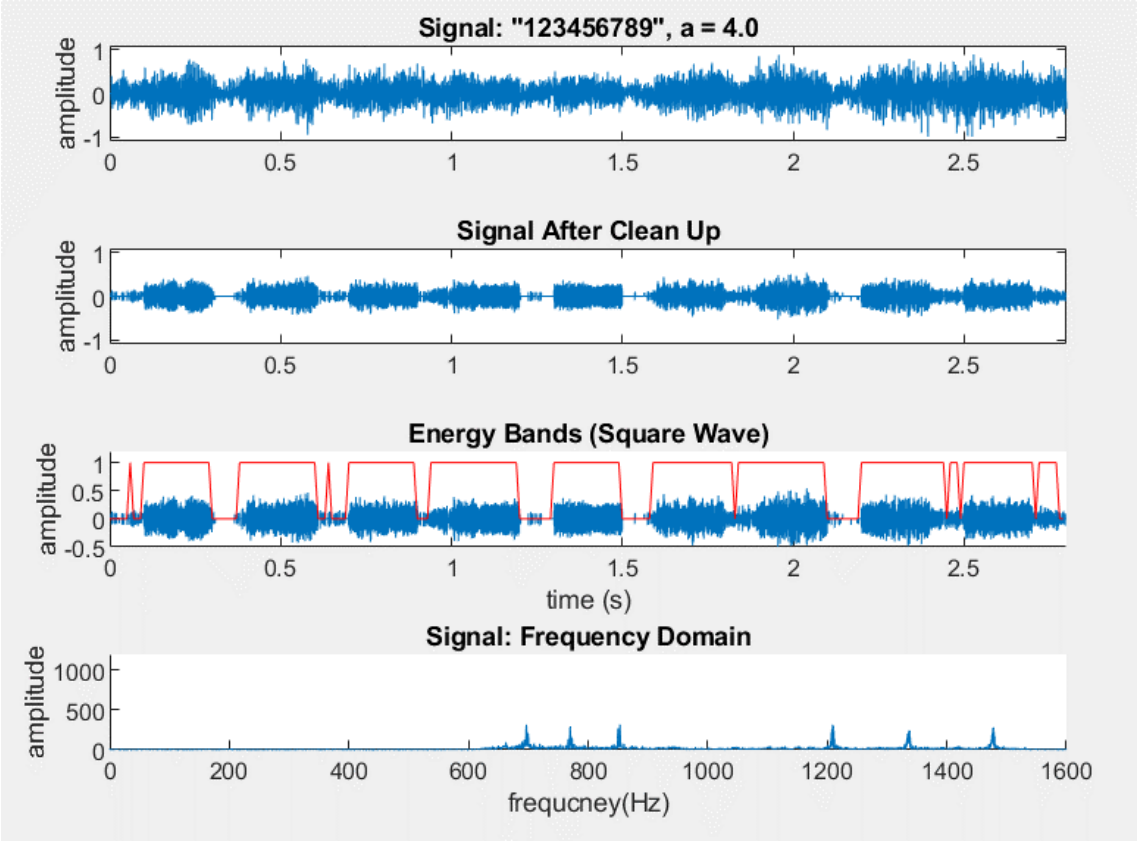
a = 12, outputs '1235521'

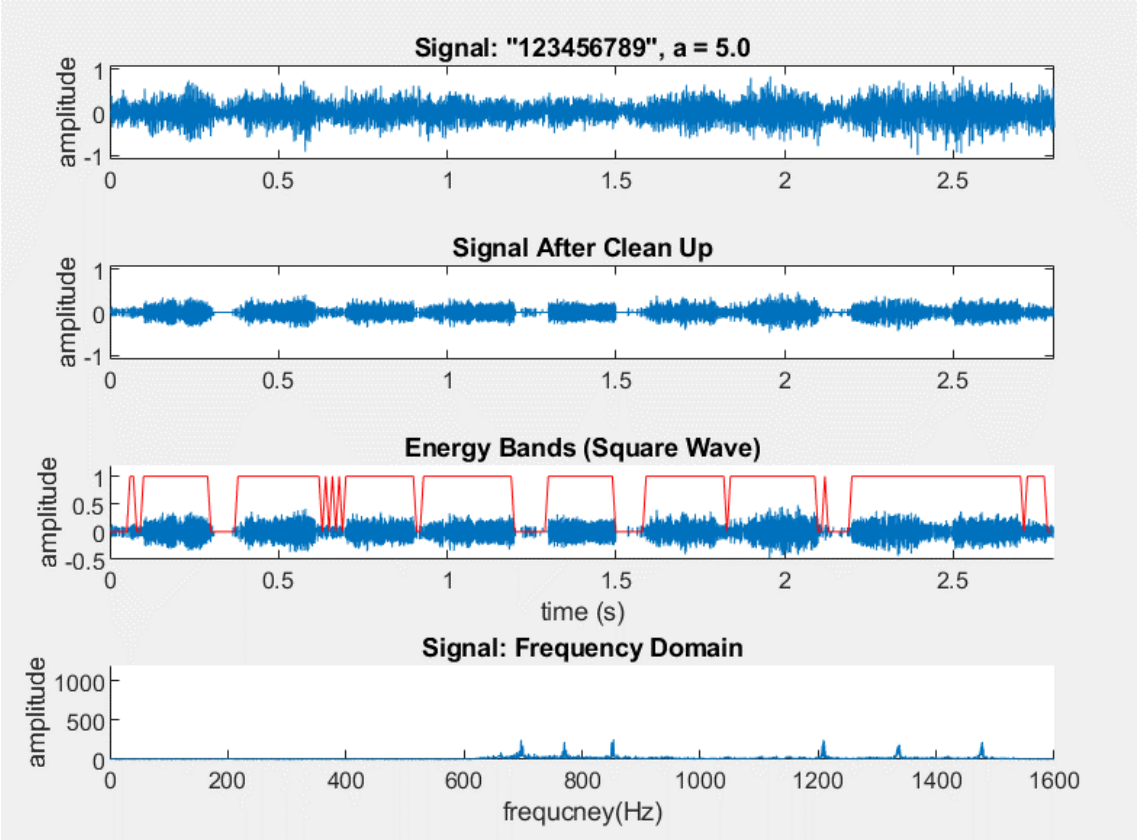

Signal: "123456789", a = 0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)

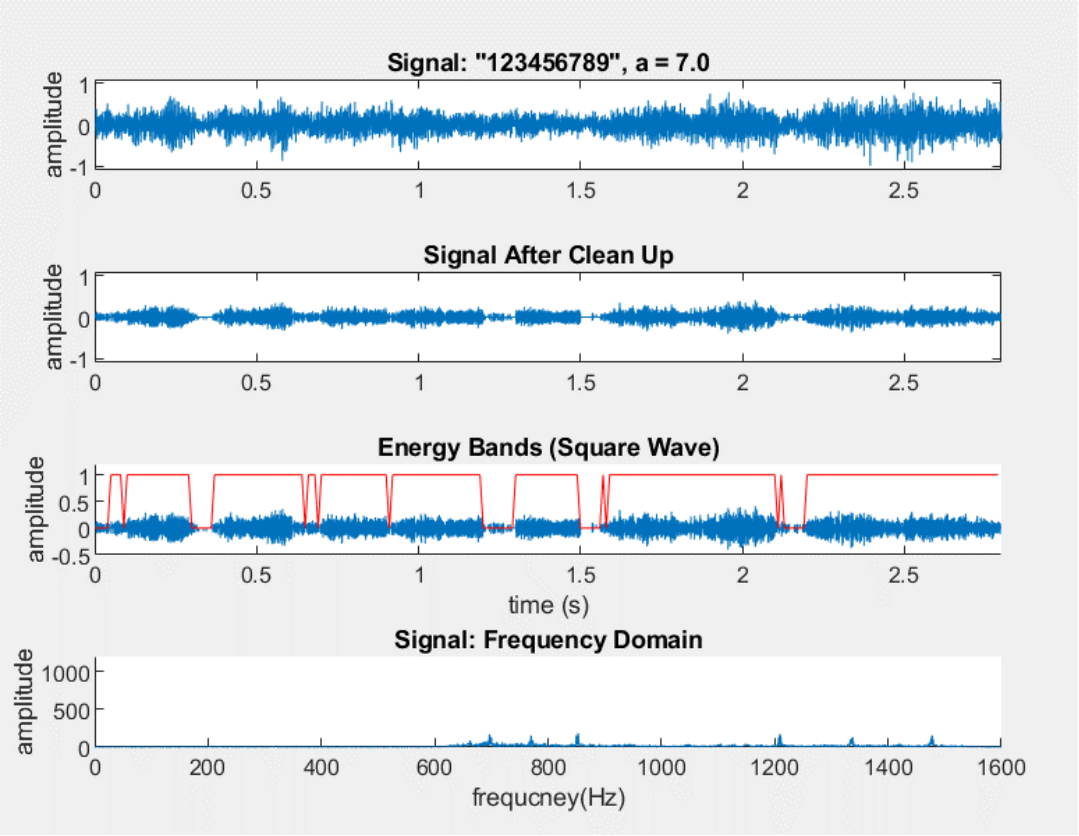

Signal: "123456789", a = 1.0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)

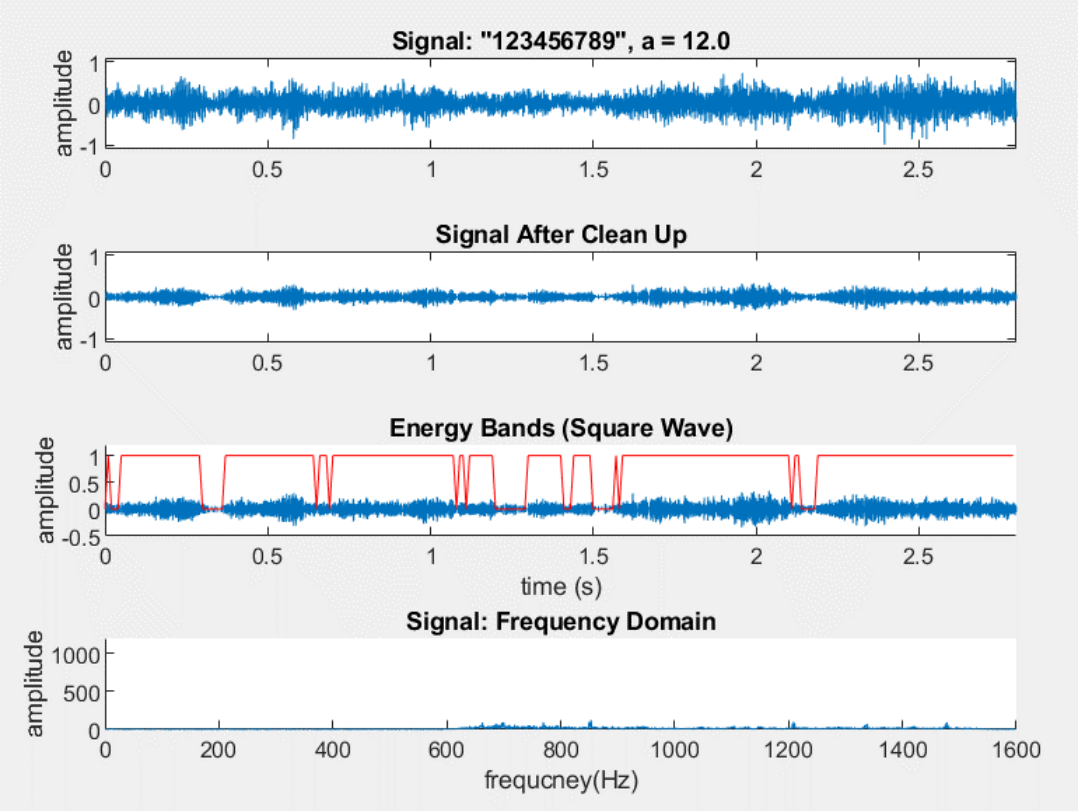

Signal: "123456789", a = 2.0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)


Signal: "123456789", a = 3.0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)


Signal: "123456789", a = 4.0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)


Signal: "123456789", a = 5.0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)


Signal: "123456789", a = 7.0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)


Signal: "123456789", a = 12.0
amplitude
Signal After Clean Up
amplitude
Energy Bands (Square Wave)
amplitude
time (s)
Signal: Frequency Domain
amplitude
frequcney(Hz)

2.5 Decoding with Reverberation

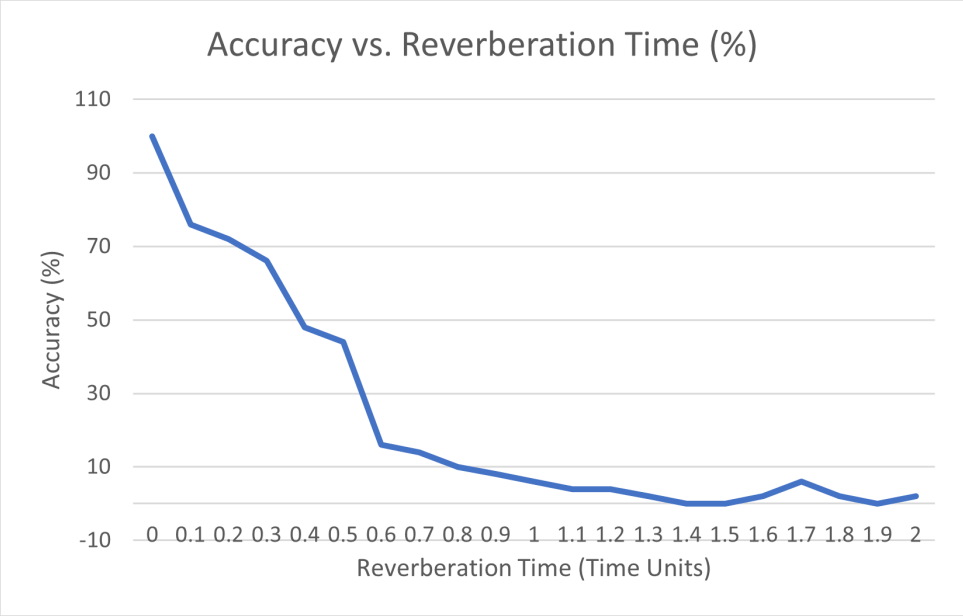
2.5.1 Experiment Overview

In this section, we use the provided addreverb.m function to generate reverberations of the key presses and investigate the relationship between decode accuracy and the frequency weighting, duration, and reverberation time.

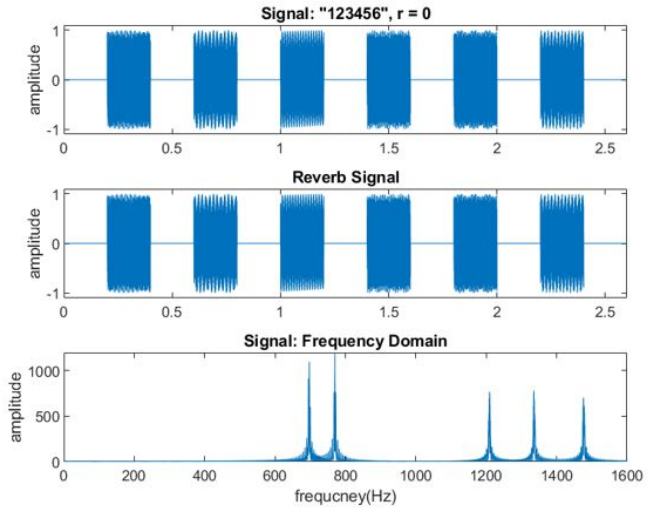
In order to test all three factors at once, the reverberation function calls are organized as follows:

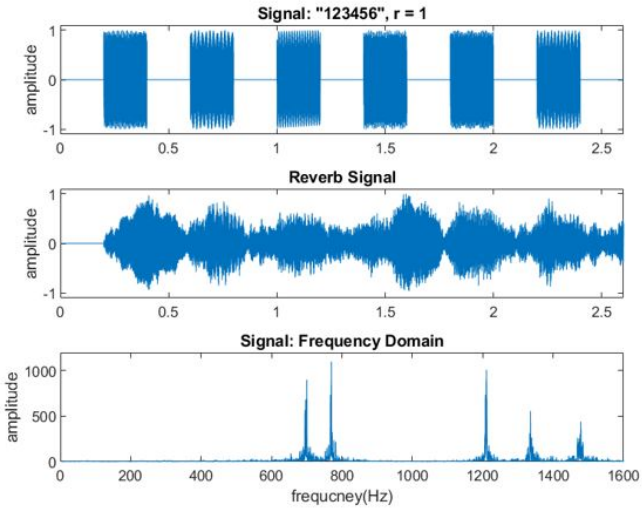
1. The 50 sequences are all random and are the same as the ones used in part 4.
2. For every 10 trials, the duration settings repeat. There are 2 trials each for duration: 100ms, 200ms, 300ms, 400ms, 500ms.
3. Every 10 trials is a group that composes a different frequency weighting, which are [3 1], [2 1], [1 1], [1 2], [1 3], respectively. Therefore, when 50 trials are put together, frequency weighting can be compared across different ranges while the sequences and key duration are all pseudo-random and unbiased.
4. The accuracy vs. reverberation time test has the default addreverb settings with reverberation time ranging from 0 to 2.0 with steps of 0.1.

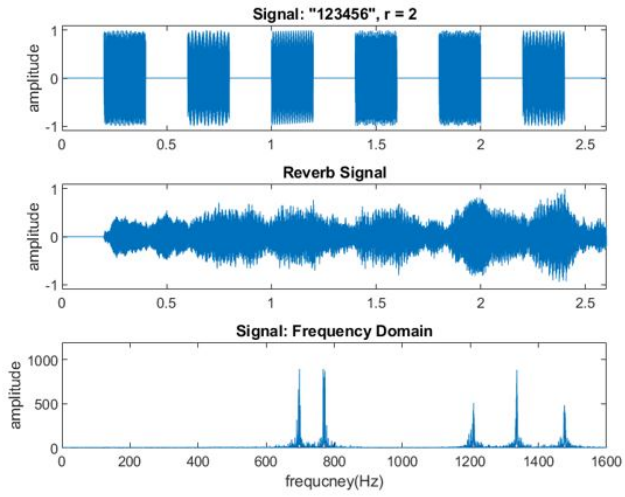
After running all trials, we plotted the Accuracy vs. Reverberation Time:

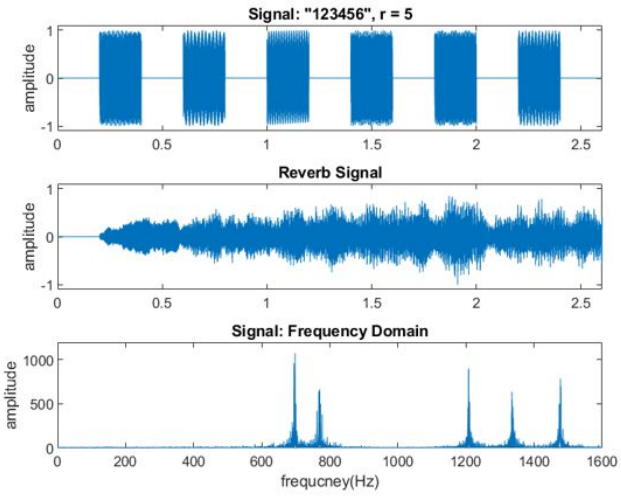


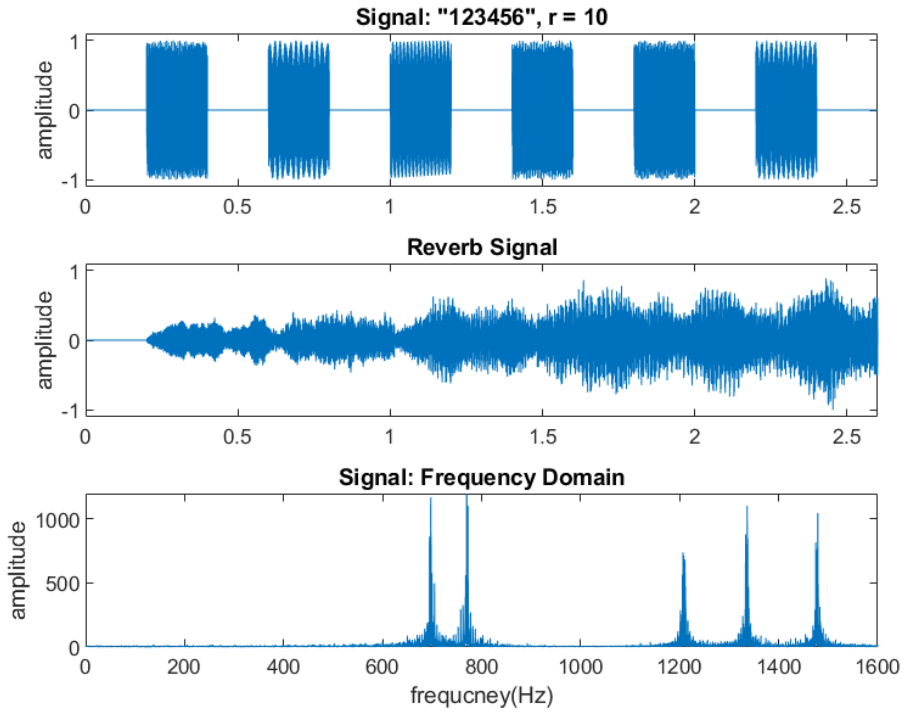
With reverberations, the accuracy falls drastically, mainly because the frequency bands overlap each other with differing amplitudes. The energies overlap each other as well. An example of deteriorating performance is illustrated below:











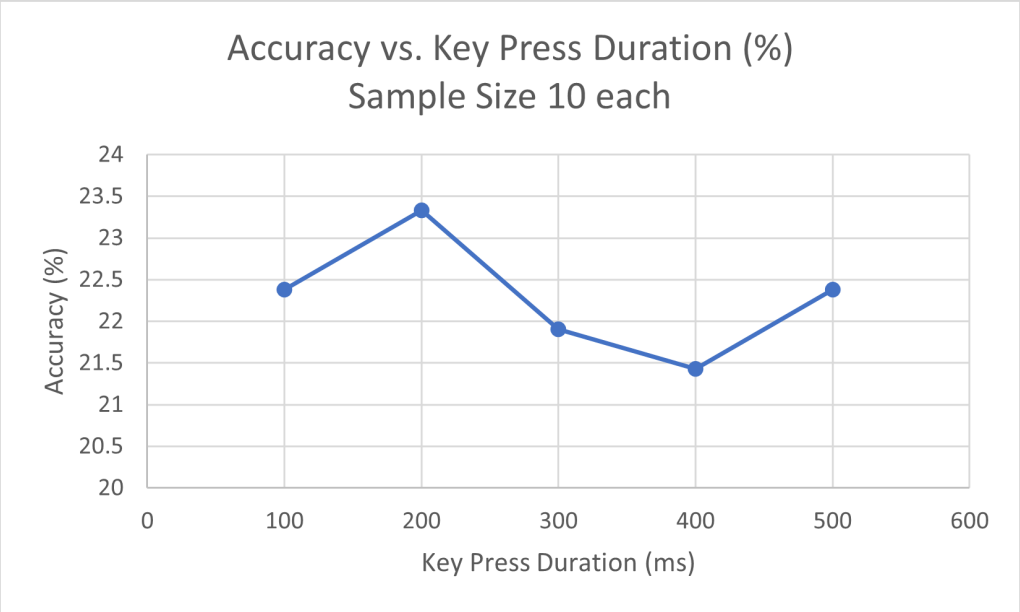
We see that reverberations significantly disrupt the time-domain signal and cause various peak heights at the target frequency ranges. When the reverberation time is very high, the frequency peaks become wider and span more frequencies. This makes it difficult for the DTMFdecode() function to output the correctly numbered and accurate result.

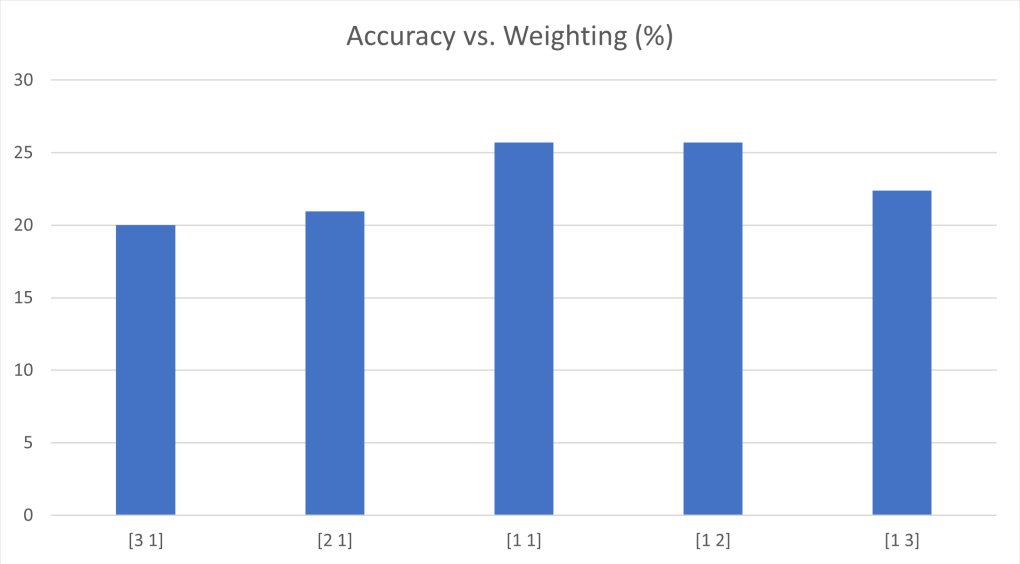
2.5.2 More Experiments with Duration and Weighting

We found that:

1. Duration: With the current implementation of DTMFdecode() and DTMFsequence(), tone duration doesn't seem to change reverberation decode accuracy significantly.

2. Weighting: More balanced tone frequency weightings create the best accuracy. This makes sense as the more equally weighted the two tones are, the less likely they would be biased toward one side. This ensures proper distinction between the signal in the frequency domain, allowing for better decode results.


Accuracy vs. Key Press Duration (%)
Sample Size 10 each
Accuracy (%)
24
23.5
23
22.5
22
21.5
21
20.5
20
0
100
200
300
400
500
600
Key Press Duration (ms)


Accuracy vs. Weighting (%)
30
25
20
15
10
5
0
[3 1]
[2 1]
[1 1]
[1 2]
[1 3]

Chapter 3

Results & Discussion

This project consists of a significant amount of MATLAB coding, testing, and data structure creation. Over 100 megabytes of audio files were created to obtain all the experimental data.

We demonstrated the principles of DTMF composition and decoding under various scenarios: white noise, additive noise, and reverberation. The encoding function is versatile enough to simulate a wide range of key duration, frequency weightings, noise levels, and other parameters.

The results show a comprehensive and detailed description on how this decoding function works: based on energy differences and band-pass filtering, it works very well with additive noise but not so well with reverberations.

Things that are done well:

- The code has a lot of calculation and plotting functions to test all aspects of DTMF decoding
- Clear and concise experimental trial design for comparison
- The signal energy-based approach to decoding is fairly versatile

Areas of Improvement:

- Test out more reverberation scenarios with non-default extend length and reverberation methods.
- Use a more advanced string matching algorithm that can match parts of the string, rather than determining output correctness by the entire correctness.
- Do more careful filtering with the energy band thresholds and explore more filtering techniques in the frequency domain.