# Project 2

LJ Gonzales

April 2023

## 1 Part I

To encode the signals, we first find the correct frequencies according to the key input by use of what is effectively a dictionary: using the *find* function on a list containing the keys outputs a value between 1 and 12, which is then used as an index for the "low frequency" and "high frequency" lists. We then instantiate *x_axis*, which represents the timepoints of the continuous sinusoidal that the discretized version will record as samples. It is a linspace range beginning at 1, ending at the user parameter *duration* (defaulted to 0.2 and measured in seconds), and with step $\frac{1}{fs}$, where $f_s$ is defaulted to 8000 in Hertz. Two sinusoidals are then generated by $sin(2\pi f \text{x\_axis})$, where f is the respective frequencies of low and high sinusoidal. Finally, the function returns the weighted sum of these two sinusoidals, defaulting to 1,1 for the weights.
We verified the output quantitatively by comparing the tone with the ones produced by a Xiaomi Note 7 cellphone, which also uses the DTMF protocol. Possible variations in weighting aside, the tones sounded similar to the respective ones produced by DTMFencode. *See appendix for all digits*

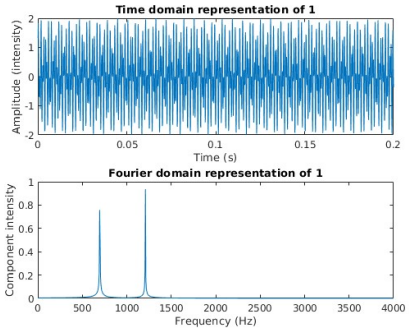



Figure 1: Time and Fourier Domain for digit 1. Rest shown in appendix

## 2 Part II

Since we are making the assumption that only one digit is pressed per recording, we can also assume that this digit will have the most salient amplitude in the frequency domain. We could relax this requirement a little bit since we know that the keytone will take one of exactly 12 combinations, and thus focus our attention fully on small intervals around those frequencies. This should be seriously be taken in consideration for environments that have high frequencies in domains relatively far from DTMF (for example, environments near roads or manufacturing equipment, which can have dominating low noise, or electrical equipment, which can contribute to the high frequency parts of the spectrum). Here however, we consider a scenario like the 'cocktail party' problem, where the most dominant noise present is in the same part of the spectra as the desired signal. In this case, we definitely need to make the assumption that the noise is less prevalent than the signal. This means we can **divide the frequency spectrum into two partitions**: the $< 1000$Hz partition which will be occupied by the 'row' frequencies of DTMF, and the $> 1000$Hz part where the 'column' frequencies will live. At this point we could simply return the frequency of the maximum frequency within each partition as the best guess for low and high frequencies of the DTMF signal.
However, to make the system more robust to slight shifts in frequency, we identify the digit a different way. Figure 2 shows an example of slightly overlapping digits where digit selection is more subtle. We identify a "threshold" at 90% of the global maximum of the partition, and **store the positions of all frequencies which exceed this threshold** in an array. In the case where we have potentially lots of frequencies, eg instead of 1209, they may be 1159, 1212, etc, we select the digit with the closest match (where the matching procedure is measured as minimum of absolute value 'distance') from the set of all found frequencies. For this and future steps, it will also help to square all datapoints in the time-domain before processing, which allows for clearer outlining of the important frequency components.

## 3 Part III

We first note that Part III is identical to Part II, with the exception that potentially multiple frequencies may be present in the recording. We of course *cannot* apply part II's algorithm directly, at it will only select one (the most salient) frequency. With this in mind and the time-domain samples in sight (see the top plot of figure 3), it is clear that the challenge has boiled down to chopping down the sample into parts, and apply the DTMFencode algorithm. We cannot exactly **'find the zeros'** as split points since the signal is constantly oscillating between positive and negative, even when a signal digit is being played. However, we can make an 'envelope" of this signal (see figure 3) and find the zeros (or values close to zero) of *that* function. Because we can safely assume that no digits overlap at the same time, we can cut the signal at these zeros and then

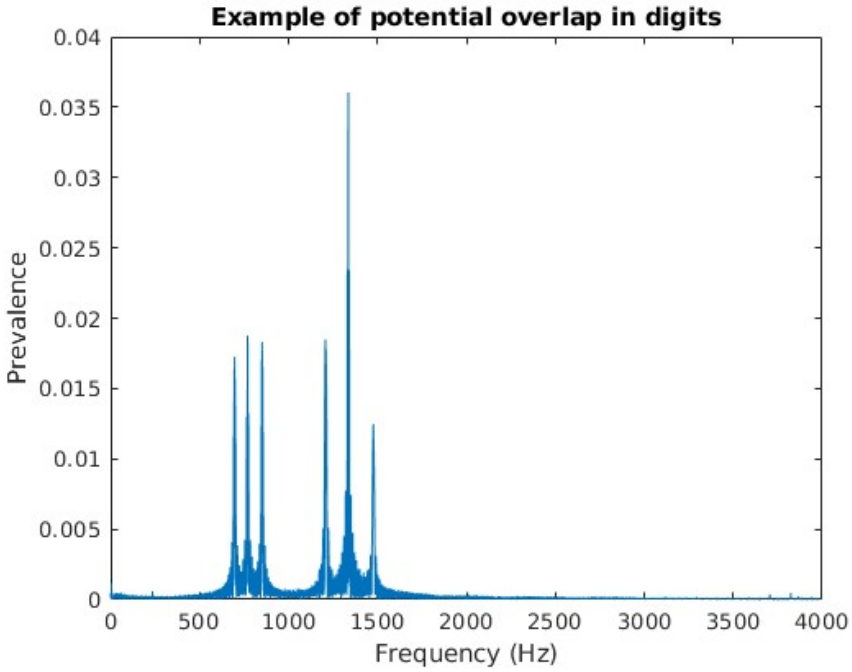



Figure 2: Example of overlap between digits where decoding the most likely digit is non-straightforward.

pass them individually through DTMFdecode.

We define this envelope function as $E[x] = \frac{\sum_{k=x-d}^{x+d} f^2[k]}{2d}$, where d is a parameter to be varied to satisfaction. Of course, a small value of d will mean that the envelope function is able to discriminate very short spans between the pressing of two numbers, but also means that it might accidentally cut a single frame in two parts if its frequency is low enough. By squaring the signal to consider only amplitude, we found that d=1 seems to work well for the phone recording. This may seem low at first sight, but considering that the period of a sinusoidal between 900 and 1200Hz is about 9 times higher than default sampling frequency 8000Hz (and nearly double even at 3000Hz, which is our ' worst-case scenario' in this project), it makes sense that the chance of finding 3 consecutive near-0 samples in a nonzero portion of a signal, is very small. Of course this will not be the case for frequencies must lower than this (simply consider points near the y-axis intersection)

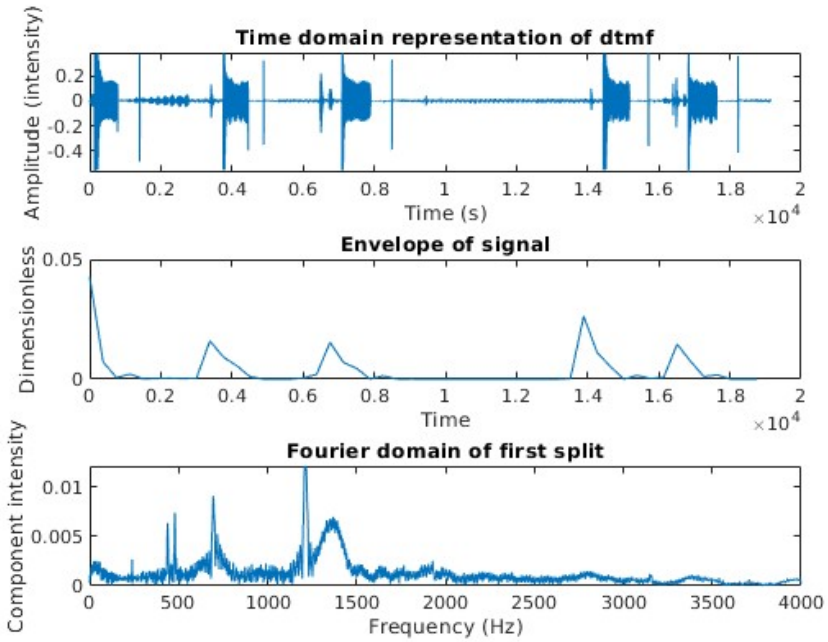



Figure 3: Example sequence, envelope, and fourier domain

# 4 Part IV

Following suggestions from the instructions, we implement a 3-level for loop sequence. The inner loop iterates through all 50 generated audio files with varying duration, weights, and frequency. We observed a steady, but non-dramatic decrease in accuracy as the noise factor was raised from 0 to 3 in increments of 0.1. The algoritm did start to seriously break down at noise amplitude/original signal amplitude ratio greater than 3. The cause of deterioration at high alpha was not so much the Fourier analysis step of the algorithm, which stayed clear and decisive, but finding the "zeros" of the envelope, as shown in figure 4. Figure 5 shows the plot of accuracies vs. noise amplitude coefficient alpha.

# 5 Part V

So far in this investigation, we have assumed that the relative weighting of low and high frequencies are constant thoughout a single sample. In particular, since every tone is composed of exactly one high and one low tone, we expect the amplitude of the signal to be constant across key presses (additive noise nonwithstanding). As is shown clearly in figure 7, this same assumption cannot be made with reverb signals. Figure 8 shows the accuracy using a naïve

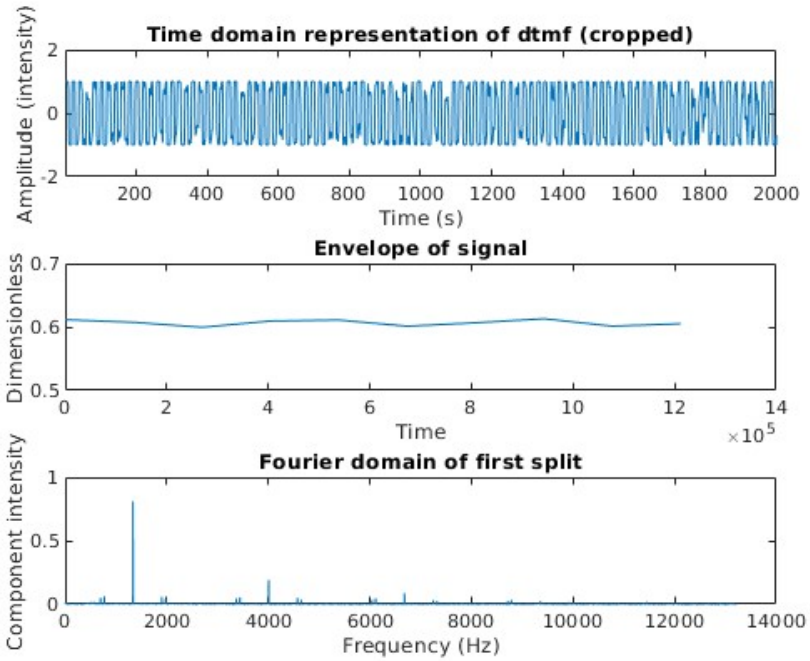



Figure 4: Analysis of a failed sequence "254*603422", with noise factor 3, which the model reported as "2". This is because the envelope hardly differentiates digits at all, so it recognizes the sequence as a single, long digit: Pay special attention to the y-scale of envelope compared to figure 3

implementation, that is, one unmodified from parts I-IV. It performs terribly, essentially only retaining the slightest bit of performance at reverb time of 0.

There are two primary ways for which the current model is unfit for reverb noise: first, it does not take advantage of the fact that the zeros of the envelope will be very well defined, even at high levels of reverb: this is due to the convolution-based origin of the noise, where 0 convoluted with anything remains 0. Second, and related to the first, is the fact that we squared the signal in the time domain to get a sharper understanding of where the digits occurred. Not only is this not necessary, but it is actually counterproductive here, because the amplitude changes across the signal.

We should not expect the Fourier Transform of the algorithm to change; multiplication in the time domain is a convolution in the frequency domain, and assuming that the fourier domain is composed of a sum of deltas, they should (at most) be scaled by some factor. This is a transformation to which our algorithm as described in part II stays robust. Figure 6 shows this, with a reverb time of 5. The amplitude of the spikes is greatly reduced, but they are

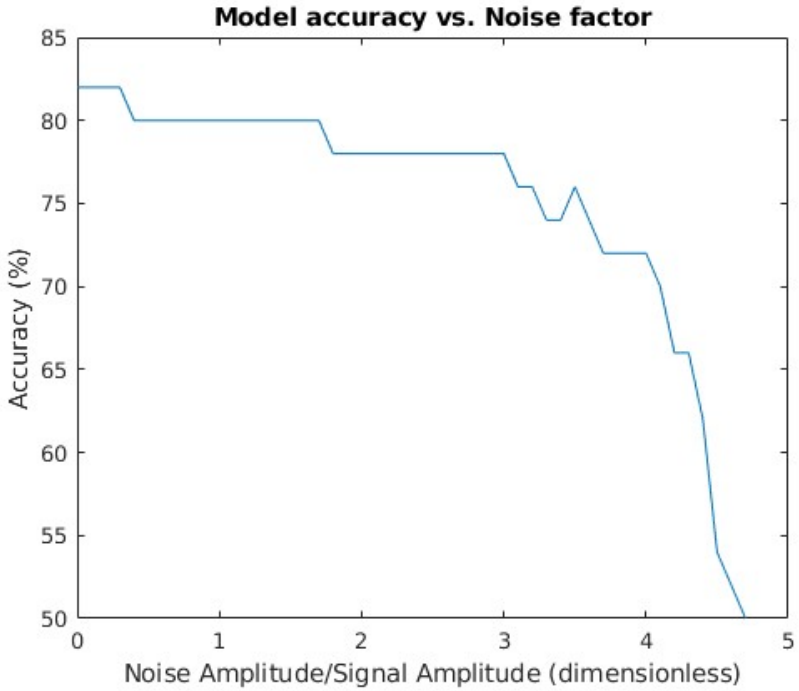



Figure 5: Accuracy vs noise

still very differentiable.

In order to fix the abbhorent performance of our original model, we sketch a new one that addresses these issues. In particular, in computing the envelope, **we apply the absolute value function on the time domain $x(t)$ instead of squaring**, to reduce large differences that lead to bad envelopes.

We also reduce the threshold of the split to 5% of the global maximum signal amplitude rather than 20-30% in the first part. Though we could potentially reduce it all the way down to 0, our model would perform significantly worse with even a little bit of additive noise, and we note no significant increase in performance to justify this loss. We also see that at very high reverb noise levels (2 and greater), even the originally silent moments of the recording gain amplitude. (Compare figure 7 with 10). This may be because the reverb is so great that even small background noise gets amplified. **Ideally, we would have resilience to both additive and reverb noise, but from analysing the required conditions for each, we see that they are contradictory to some level**. Figure 9 shows the improvement of the model. As we can see in figure 10, the model still fails to discriminate different digits, and only sees the most salient one.

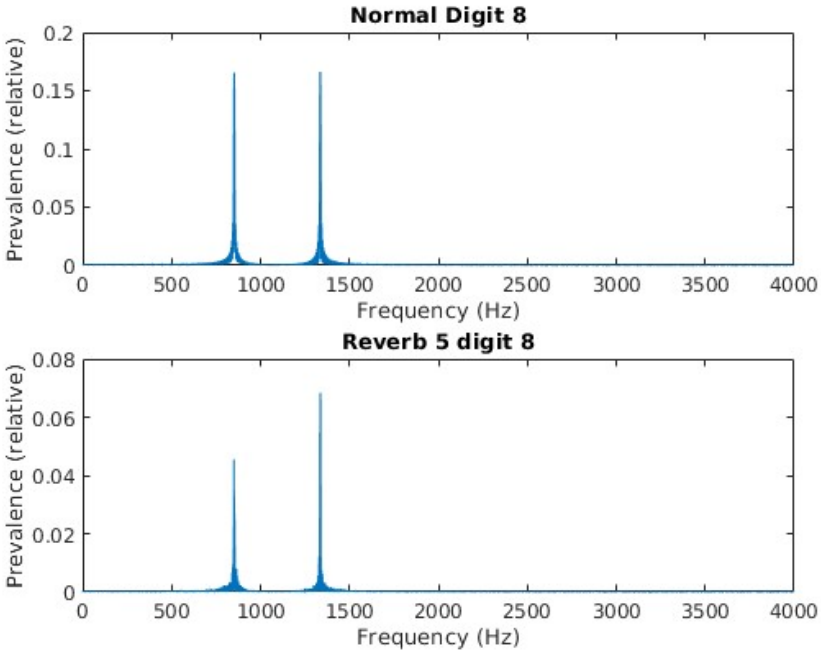



Figure 6: As discussed, reverb noise only affects the fourier domain through rescaling of delta spikes

We note that the model worse with increasing duration of tones (this is perhaps intuitively clear from the nature of reverb). It also seemed that a larger spacing between different key presses lead to better accuracy. Regardless of reverb, though, our model performs better with more spacing between keystrokes, so it is not clear how much is actually attributable to it. .

# 6 Appendix: all single-digit representations

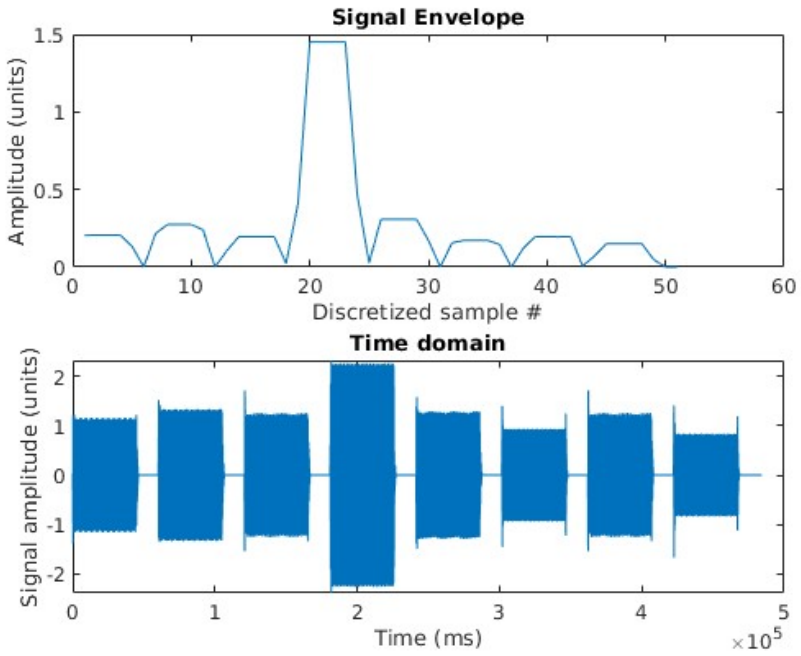



Figure 7: Computed Envelope and original signal for reverbed signal with duration 1. The squaring of the amplitude (originally meant to reduce additive noise) exacerbates differences in amplitudes in the signal. This is why we opt for absolute value rather than squaring in computing the envelope.

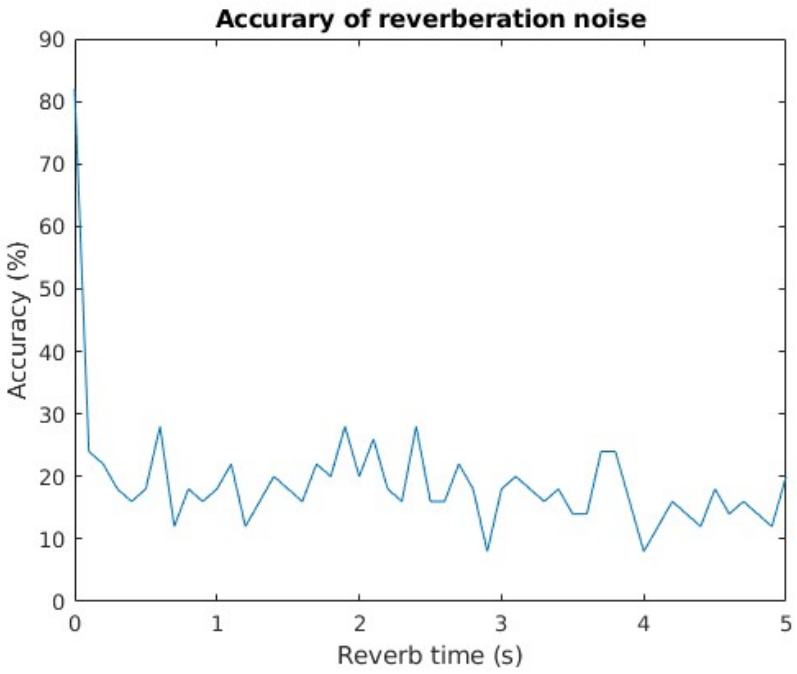



Figure 8: Naïve model accuracy on the reverbarated dataset.

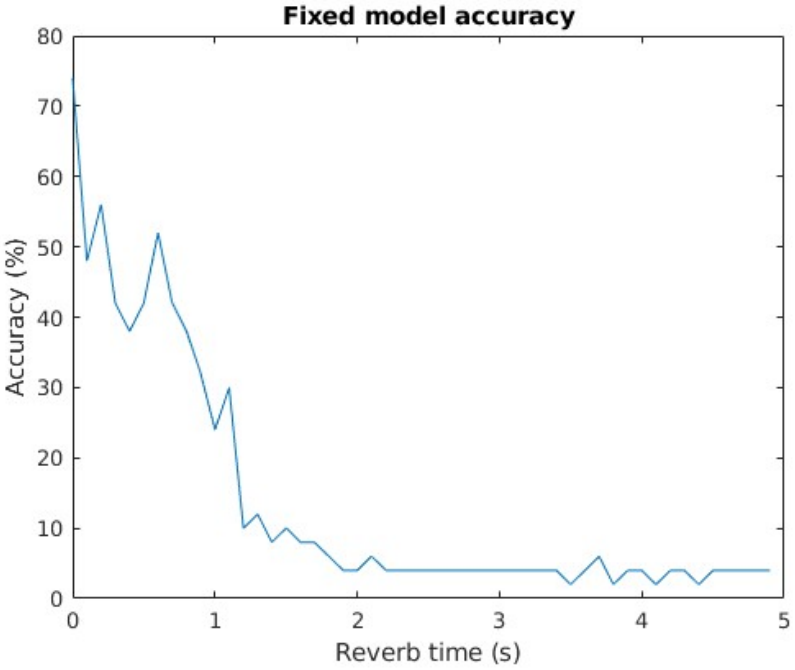



Figure 9: The fixed model performs better on a wider range of reverb noise. This is at the cost of robustness with respect to additive noise.

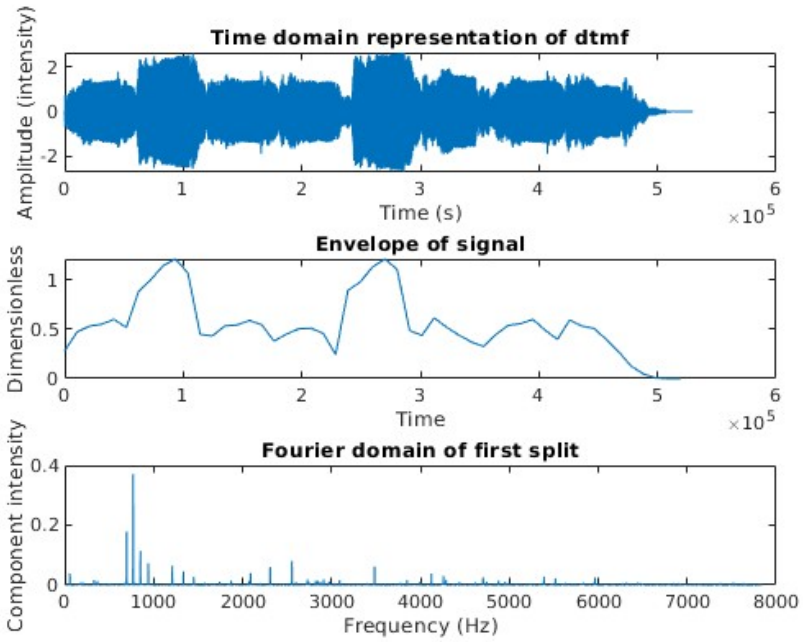



Figure 10: Time domain, Envelope, and Fourier Domain, for the sequence "152*4728" with reverb time of 3 seconds. The model correctly identified a correct digit as 2, but did not notice any others. 2 was most likely selected because it appears twice in the sequence

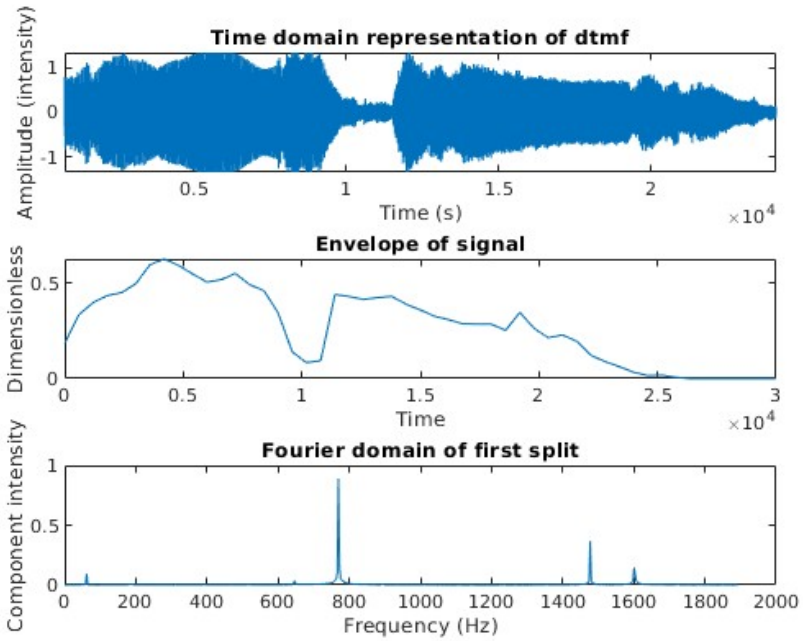



Figure 11: largeinterdigit

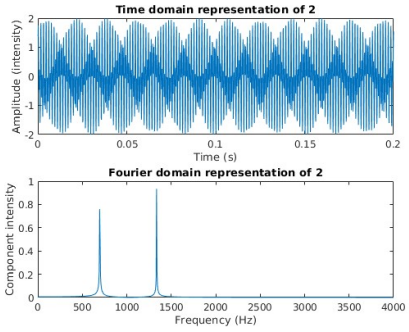



Figure 12:

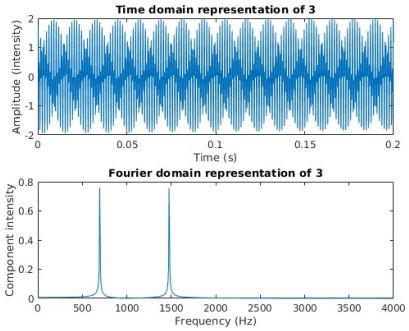



Figure 13:

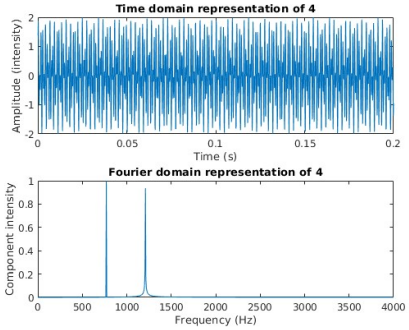



Figure 14:

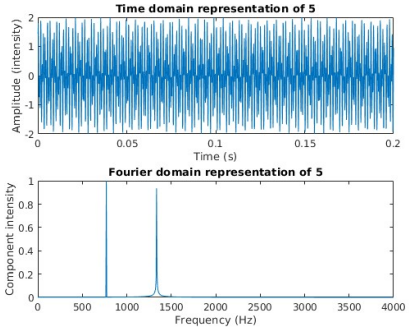



Figure 15:

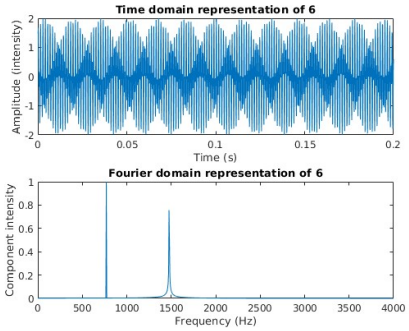



Figure 16:

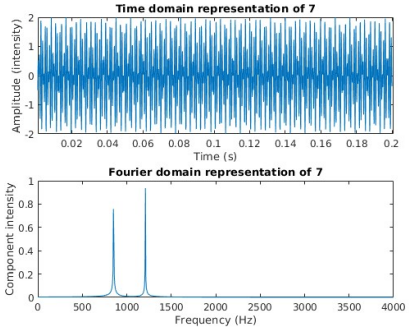



Figure 17:

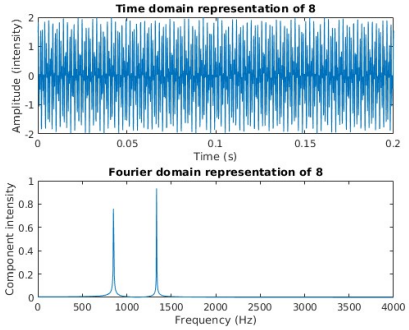



Figure 18:

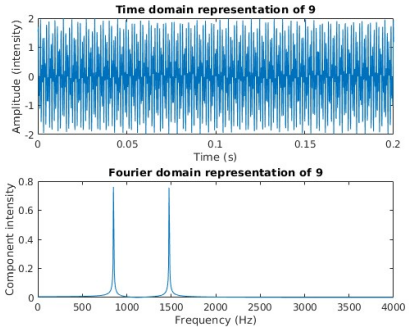



Figure 19:

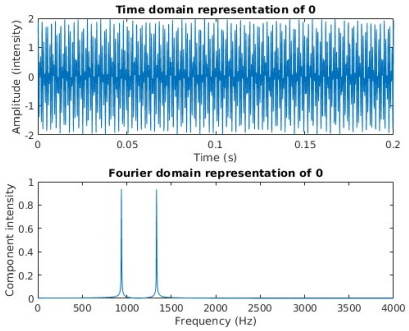
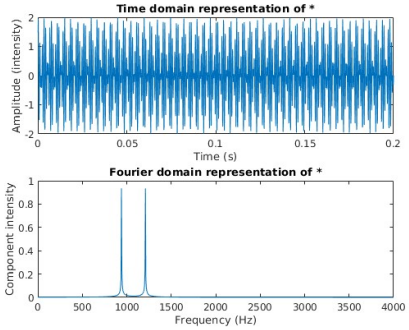



Figure 20:



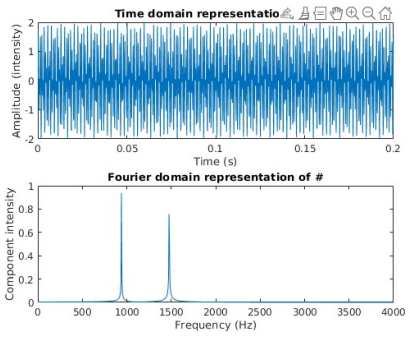

Time domain representatio
Amplitude (intensity)
Time (s)
Fourier domain representation of #
Component intensity
Frequency (Hz)

Figure 21: