DTMF Signals Analyzer

Name: Noam Navon
ID: 322937384
Lecturer: Dr. Tom Trigano

Objective

Realize what numbers are typed from phone number recordings with a Matlab code.
- Implementing theory
- Managing a Matlab project
- Preparation for future research
- Practice in Engineering code

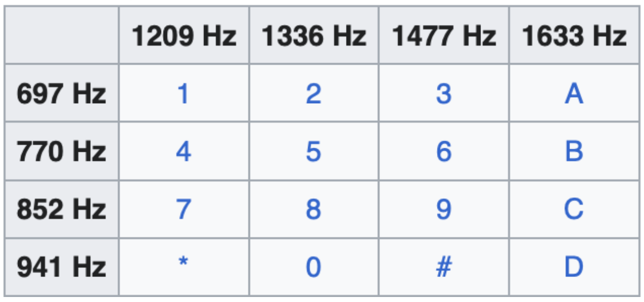
How to use

1. Insert your sound files in a WAV format to the “recordings” folder
2. Open Matlab and enter the “Signals & Systems project - Noam Navon” directory
3. Add “recordings” and “functions” to the path if necessary
4. Enter ‘main’ into the Command Window
5. Choose the number of the recording you want

---

1 Abstract

DTMF- Dual Tone Multi Frequency is a way to code characters with sound. Every character is coded with two frequencies that play together as long as the key is pressed.



	1209 Hz	1336 Hz	1477 Hz	1633 Hz
697 Hz	1	2	3	A
770 Hz	4	5	6	B
852 Hz	7	8	9	C
941 Hz	*	0	#	D

Figure 1.1

This table describes the coding of combinations as a pair of two frequencies. In this way we have the relation between sound of phone typing and phone numbers. That fact can be used to determine numbers based on recordings.

From a human perspective, listening to a recording, as long as a phone typing sound is noticeable, we can determine precisely where phone typing happens and find the exact phone number we hear with slightly more effort. Writing a code, however, for this purpose might be challenging since our hearing perception is much more complicated and sophisticated than we might think. Different real-life problems blur the signal's physical shape, making analyzing the phone number almost impossible in some scenarios.

In this article, I will go over the following topics:
1. Suggested approach to obtaining actual numbers from recordings.
2. Implementing these strategies in code.
3. Weak spots of the algorithm.
4. Further exploration and development.

2 Algorithm

Fourier Transform-

Given a recording, one of the ways to analyze it is to check the Fourier transform in the desired frequencies; that is, finding the amplitude of the projection of the signal in each of these frequencies should provide us with a numeric way to judge the presence of the sound we seek:

$$2.1 \quad |F(j\omega)| = \left| \int_{t=a}^{b} f(t)e^{-j\omega t}dt \right|$$

Signals in the computer are sampled so that a better representation would consider a discrete sum of the sampled signal in the desired samples interval:

$$2.2 \quad |F[j\omega]| = \left| \sum_{n=a}^{b} f[t]e^{-j\omega n} \right|$$

This approximation improves as the sampling frequency increases, but it should give us valid results even with a relatively low sampling frequency, such as 8k Hz. It can be implemented using Matlab's “sum” function or the “trapz” function.

One thing to consider when transforming a section of a signal is the Uncertainty Principle.
Given a time-bounded signal, we have the time-frequency precision tradeoff. This tradeoff is a mathematical fact, not a measure error. It states that knowing the exact time of the transformation gives us no information about the precisely measured frequency. On the other hand, knowing the precise frequency gives no information about when it happened.
This claim, known as the Fourier Uncertainty Principle, is similar to the Heisenberg Uncertainty Principle used in quantum mechanics and has more variations, like for the Doppler effect. Hence, we should take a reasonable sample interval when transforming, giving us a fair approximation of time localization and providing meaningful data about the frequencies playing a role.

Correlation-

Another way to analyze a recording is by using the correlation between signals.
Generating a sine wave in the desired frequency makes it possible to check the correlation between the signal and the sine in different time sections to find how this frequency is expressed in the signal. The correlation can be calculated in the following expression:

$$2.3 \quad corr(x, y) = \rho(x, y) = \frac{cov(x, y)}{std(x)std(y)}$$

Consider a signal in which the frequency f is found in a phase shift $\pi/2$.
Taking the correlation with a sine wave of the same frequency gives zero since the inner product of orthogonal vectors gives zero. The inevitable conclusion is that a sine wave alone cannot judge frequency presence using correlation. However, two orthogonal vectors can be linearly combined to span the desired space, which leads to the following conclusion:

$$2.4 \quad Asin(\omega x) + Bcos(\omega x) = Csin(\omega x + D)$$

Taking the correlation of the signal with both a sine wave and a cosine wave, then taking the amplitude of the results and summing them up, should give us a way to find the presence of a specific frequency in the signal.
Timing-Frequency tradeoff exists here, so we should take appropriate sections.

Gating-

After obtaining frequency presence in different time sections, we can determine whether a key was pressed there. Assuming that random noises won't have a specially large magnitude in the precise frequencies we are interested in, we can determine a level of filtering that will serve as the difference between presses and random noise.
This approach I chose has several problems I will discuss later, but for a "beta version" algorithm, this should work for most good recordings with a low noise-to-signal ratio.

Assume that relevant frequencies in presses vary in normalized magnitude between 0.6-1, while noise varies in normalized magnitude between 0-0.4.
Taking the mean of the presses magnitude gives 0.8, and for noise gives 0.2.
In this sense, let's generalize this for different ratios and state that the mean magnitude for noise is 'A' while for pressing is 'B'.
The ratio of quiet time between presses to pressing time is typically 4.5, so taking the mean of all frequencies in all time intervals should give (I don't search for 1633 Hz - not a phone number):

$$2.5 \quad E(Amp) = \frac{(2B+5A)+4.5(7A)}{7\cdot 5.5} = \frac{2B+36.5A}{38.5} = 0.05B + 0.95A \approx A$$

Taking the mean of all frequencies gives a good approximation to the mean of the noise, with an offset to a slightly higher value.
In the example of a noise mean of 0.2 and a sound mean of 0.8, we obtain: 0.04+0.19 = 0.23.
Since the magnitude of the mean represents the middle in random noises, doubling it should be a good enough approximation for filtering. This approach gives the initialization for adapting bar that will be the threshold for what is known as 'Gating,' which is deleting everything below the threshold.

Leaks-

After gating, some noise might pass the threshold we set. In this case, there are three ways to treat it:

I. If three frequencies pass, the threshold is too low - incrementation.
II. If there is only one frequency, it's a reasonable leak that should be ignored.
III. If two frequencies don't make sense (e.g., 697, 770), they should be ignored.

These problems might escalate in noisy or dynamic recordings so the algorithm might fail.

Overlaps-

Consider a typing that occurred in between two times sections. The magnitude of the frequencies will be divided between the two sections. Moreover, two adjacent time sections will have the same dominant frequencies after gating, leading to duplicating the number in the result.
In cases of dynamic key pressing time, this might even make the number appear three or more times. For simplicity, I assume static pressing time with a short pressing time of fewer than 0.1 seconds and more than 0.2 seconds of silence between presses.
In this case, adjacent intervals of 0.1 seconds with the same gated frequencies will ignore the second appearance every time.

Decoding-

Having a conclusion expressed in a matrix of pairs of frequencies, we use the DTMF table provided and obtain the phone number most likely pressed.

3 Code

Focus on functions that are part of the main algorithm.

Signal Normalization:

```
function signal = sigNorm(signal)
    subplot(2,6,[1 2])
    plot(signal)
    title('Signal')
    xlabel('Sample')
    ylabel('Bits per sample')

    signal = signal/max(abs(signal));
    while nnz(abs(signal)>0.8)<length(signal)/1000
        signal(abs(signal)>0.8) = signal(abs(signal)>0.8)*0.7;
        signal = signal/max(abs(signal));
    end

    subplot(2,6,[7 8])
    plot(signal)
    title('Normalized Signal')
    xlabel('Sample')
    ylabel('Bits per sample - Normalized')
end
```

The signal is normalized and sometimes slightly distorted in cases where an instant unnatural peak prevents normalization.

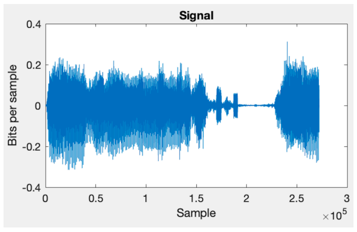


Figure 3.1

A signal before normalization.

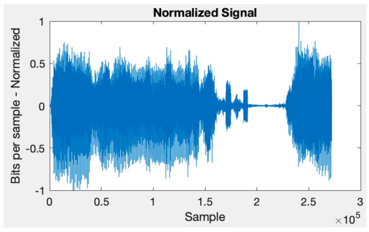


Figure 3.2

A signal after normalization.

Correlation Analysis:

```
function corr_mat = corrAnalyze(theory, time_wind, samp_wind, samp_mat)
    sines = sin(2*pi*theory.*linspace(0,time_wind,samp_wind)');
    cosines = cos(2*pi*theory.*linspace(0,time_wind,samp_wind)');

    corr_sin = zeros(7,size(samp_mat,2));
    corr_cos = corr_sin;
    for j = 1:7
        for k = 1:size(samp_mat,2)
            corr_sin(j,k) = corr(sines(:,j),samp_mat(:,k));
            corr_cos(j,k) = corr(cosines(:,j),samp_mat(:,k));
        end
    end
    corr_mat = abs(corr_sin)+abs(corr_cos);
    corr_mat(isnan(corr_mat)) = 0;
    corr_mat = corr_mat/max(corr_mat,[],"all");
end
```

Generating sines and cosines in the DTMF frequencies, we then take the correlation of each time interval with the frequencies.
The absolute values of the correlations are taken, and we sum up the two correlation matrices.
This gives a reliable amplitude for each of the frequencies.
NaNs at the end of the signal are being ignored, and we normalize everything.

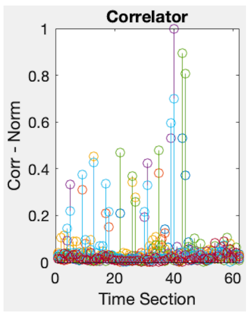


Figure 3.3

The graph shows the correlation of the signal with each of the frequencies. Each color represents one of the frequencies.
The x-axis is the time interval tested while the y-axis is the correlation strength.

Fourier Analysis:

```
function fourier_amp = fourAnalyze(fs, time_wind, samp_mat, theory)
    t = 0:1/fs:time_wind-1/fs;
    fourier_amp = zeros(7,size(samp_mat,2));
    for f = 1:7
        fourier_amp(f,:) = abs(trapz(samp_mat.*exp(-1i*2*pi*(theory(f))*t')));
    end
    fourier_amp = fourier_amp/max(fourier_amp,[],"all");
end
```

Generating a vector of time in which each data stored in 8-24 bits occurs, we take the Fourier transform by definition using trapz for trapezoidal integration, a good enough approximation.
The absolute value gives the amplitude, and eventually, everything is normalized.

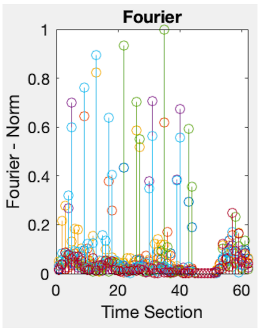


Figure 3.4

The graph shows the frequency-domain amplitude of the signal for each of the frequencies. Each color represents one of the frequencies.
The x-axis is the time interval tested while the y-axis is the amplitude.

Frequency Projection:

```
function freqs_proj = freqAnalyze(time_wind,samp_wind,samp_mat,fs)
    theory = [697 770 852 941 1209 1336 1477];
    corr_mat = corrAnalyze(theory, time_wind, samp_wind, samp_mat);
    subplot(2,6,3)
    stem(corr_mat')
    title('Correlator')
    xlabel('Time Section')
    ylabel('Corr - Norm')

    fourier_amp = fourAnalyze(fs, time_wind, samp_mat, theory);
    subplot(2,6,4)
    stem(fourier_amp')
    title('Fourier')
    xlabel('Time Section')
    ylabel('Fourier - Norm')

    freqs_proj = fourier_amp.*corr_mat;
    freqs_proj = freqs_proj/max(freqs_proj,[],"all");
    subplot(2,6,9)
    stem(freqs_proj')
    title('Correlator x Fourier')
    xlabel('Time Section')
    ylabel('Corr x Fourier')
end
```

Taking the DTMF frequencies, using corrAnalyze, we check correlation.
The exact process goes with fourAnalyze applying Fourier transform.
Taking the product of the two matrices with the Hadamard product, we obtain a matrix representing the frequencies' presence. The projection strength of noise is typically much lower since the result follows a behavior of squared numbers, and the matrix is normalized.

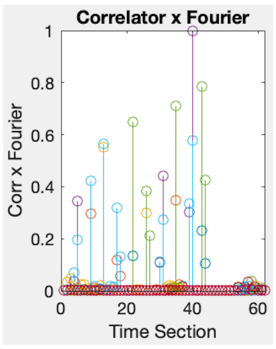


Figure 3.5

The graph shows the product of the correlation with the amplitude. Each color represents one of the frequencies.
The x-axis is the time interval tested while the y-axis is the product.

Gate Threshold Determination:

```
function level = levelDet(freqs_proj)
    noise_mean = mean(freqs_proj,"all","omitnan");
    level = 2.1*noise_mean;

    while level~=1
        peaks = freqs_proj>level;
        if any(sum(peaks)>2)
            level = level + 0.01;
            continue;
        end

        if any(sum(peaks(1:4,:)) > 1) || any(sum(peaks(5:7,:)) > 1)
            level = level + 0.01;
            continue;
        end
        break
    end
    subplot(2,6,10)
    stem(freqs_proj')
    yline(level, "Label","Gate")
    title('Gate Suggestion')
    xlabel('Time Section')
    ylabel('Corr x Fourier')
end
```

Given frequency projection, this function takes the mean of everything and doubles it to initialize the threshold.
I chose to multiply by 2.1 for a little more tolerance.
The threshold bar is raised as soon as more than two frequencies pass, or two that don't make sense together.

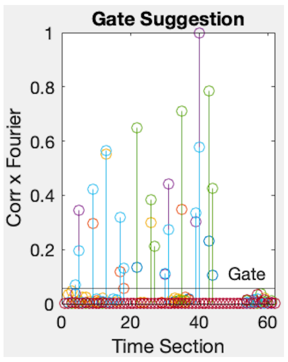


Figure 3.6

Gate suggestion is exampled here.
Everything under the bar will be a logical 0 while everything above will be a logical 1.

Fixing Time Overlaps:

```
function mat = fixTimeOv(mat)
    for i=1:size(mat,2)-1
        if all(mat(:,i+1) == mat(:,i))
            mat(:,i+1) = 0;
        end
    end
end
```

This function removes every second appearance of the sample pair of frequencies if it appears in adjacent time intervals.
This is made assuming that in 0.1 seconds, no one is pressing the same key twice.

The Main Process:

```
function Sound2Number(signal,fs)
    signal = sigNorm(signal);
    sound(signal,fs)

    time_wind = 0.1;
    samp_wind = time_wind*fs;
    signal = [signal; zeros(samp_wind-mod(length(signal),samp_wind),1)];
    samp_mat = reshape(signal,samp_wind,[]);

    freqs_proj = freqAnalyze(time_wind,samp_wind,samp_mat,fs);
    level = levelDet(freqs_proj);
    pairs = freqs_proj.*(freqs_proj>level);

    hits = pairs~=0;
    hits(:,sum(hits)==1) = 0;
    hits = fixTimeOv(hits);
    subplot(2,6,[5 6 11 12])
    stem((pairs.*hits)')
    title('Analysis Result')
    xlabel(['Time Section of ' num2str(time_wind) ' seconds'])
    ylabel('Frequency Gated Presence')
    hits = hits(:, any(hits ~= 0, 1));

    phone = decode_phone(hits);
    dispPhone(phone);
end
```

The signal is first normalized in sigNorm and compressed, slightly distorted in a few timings (If necessary due to not intended impulses) to obtain a naturally normalized signal.
Choosing a reasonable arbitrary time interval of 0.1 seconds, the signal is reshaped to a matrix when each column is a time interval, and each row is the bit number in each time interval.
Let the length of an interval be L and the number of time intervals be N. The matrix is L x N.
Using freqAnalyze, we obtain frequencies present in each time interval, so the matrix is 7 x N.
The threshold is then determined with levelDet, and the gated matrix is obtained.
The matrix is transformed into a logical matrix which is easier to treat, errors are deleted, and overlaps are fixed with fixTimeOv. If the phone number length is M, the result matrix is 7 x M, while each row is a frequency and each column is a number.
decode_phone treats this matrix and gives a number, while 100 is an asterisk and 102 is a hash.
dispPhone displays the result as an ordered string with a hyphen, etc.

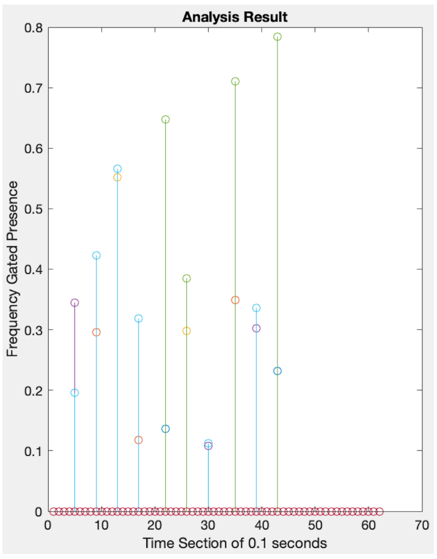


Figure 3.7

This is the final results considering amplitude.
Each time a key pressing was recognized implies a pair of frequencies in stems shown here.
This graph will be converted to logical values, times of silence are being cut out, and the number will be decoded from the final matrix.

Phone Decoding:

```
function phone = decode_phone(hits)
    hits = hits.*[1;4;7;100;0;1;2];
    phone = sum(hits);
    phone(phone == 101) = 0;
end
```

Multiplying a matrix of logical values by a vector of values gives a weighted vectors matrix.
Summing them up in a proper way gives the desired result.

4 Drawbacks

This algorithm is made with the underlying assumptions:

1. The pressing time is not dynamic, usually around 0.05-0.1 seconds.
2. The noise is not rich in the DTMF frequencies.
3. The noise is not too loud.
4. The dynamics of the noise and signal are not too big, s.t. amplitude separation is available.
5. The user doesn’t press the keys too fast.
6. Acoustics conditions don’t produce a significant echo.
7. The relative velocity between the phone and microphone is not too high, so the Doppler effect doesn’t make significant frequency shifts.

These might make the algorithm fail in some circumstances and should be considered.

5 Further Exploration

Here are a few suggestions for improving the algorithm:
Precise timing- As I chose the division for 0.1 seconds time intervals, this rough approximation is one of the weak spots. For example, taking cross-correlation with little incrementation might give more continuous, relatively more reliable results.
With a wiser time division and analysis, this approach might deal well with faster key presses and longer key holdings.

Noise Characterization- A dynamic adaptive code might get a better model for the noise characteristics, which can lead to a dynamic gate threshold, and even a noise reduction.
This approach will be a good solution for the problems that come from dynamic sounds.

Band Pass- Using BPFs, it is possible to have another view in the time domain of the suspicious peaks.

Analysis Integration- Using better integration techniques than just multiplying different results to obtain a more informed result of where key presses are.

Machine Learning- Given an extensive database of recordings paired with the actual numbers, it is possible to code training for this purpose. Better recognition of the pure signal can be achieved this way.