EE 419 - Project 4
Discrete Fourier Transform and FFT Signal Processing 2

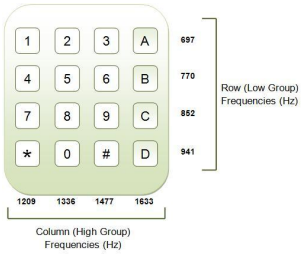
Names: Chris Adams & Aiku Shintani **Lab Date: 1/29/19**

Bench #: 9 **Section: 2**

1) Telephone Touch Tone (Dual Tone Multi Frequency) Decoder

Design Specifications: DTMF signal decoder

- Accept an audio signal sample array name
- Correctly determine the touchtone key that was pressed
- Return a text variable with the key value
- Display the spectrum of the signal you are decoding
- Display the decoded key character somewhere on the same DFT spectrum plot.
- Operate properly for both uncorrupted and noisy signals



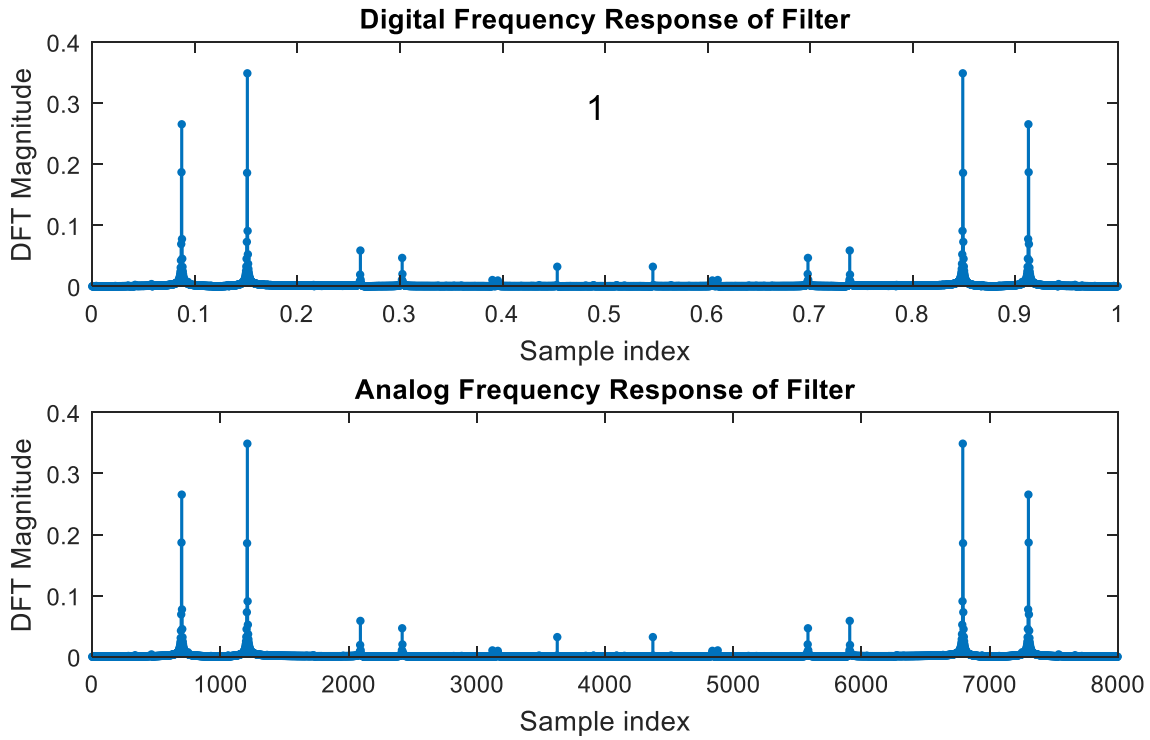
Describe the signal processing algorithm(s) used to determine the two tones present in the sound sample:

Test Case File: *touchtones.mat*

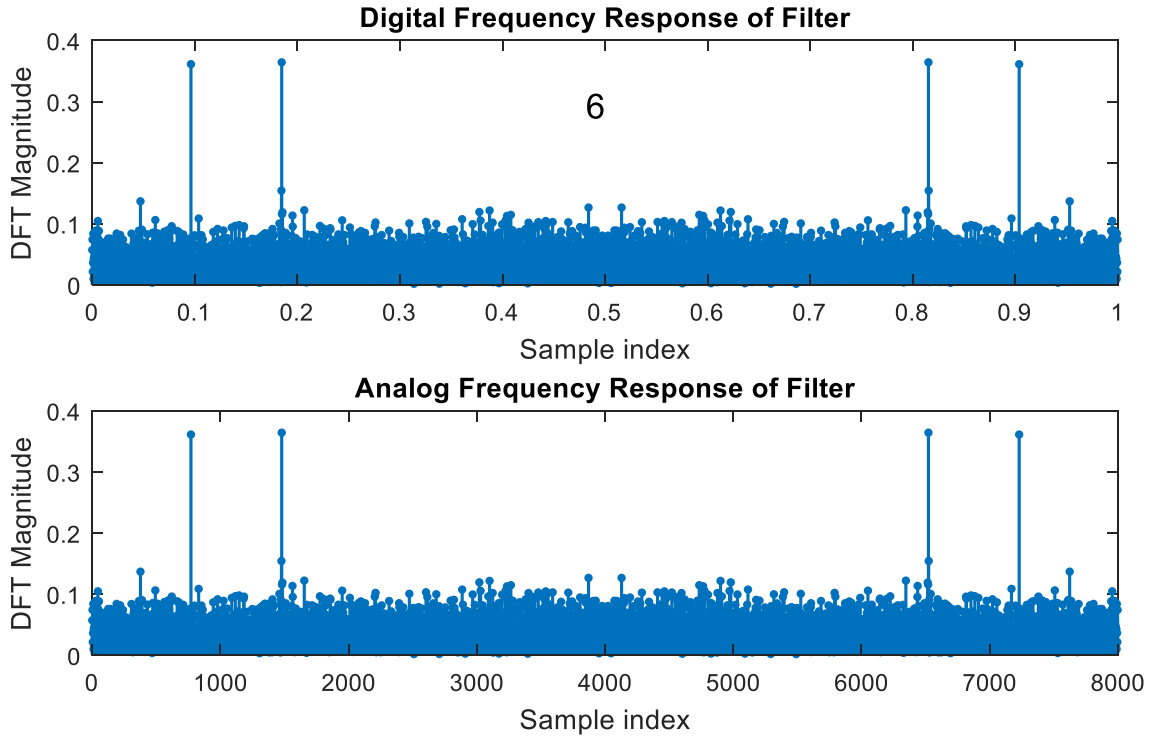
Test Results:

Tone	Audio Array	Decoded Key	Audio Array	Decoded Key
None	no_tone_quiet	**N/A**	no_tone_noisy	**N/A**
0	tone_0	**0**	tone_0_noisy	**0**
1	tone_1	**1**	tone_1_noisy	**1**
2	tone_2	**2**	tone_2_noisy	**2**
3	tone_3	**3**	tone_3_noisy	**3**
4	tone_4	**4**	tone_4_noisy	**4**
5	tone_5	**5**	tone_5_noisy	**5**
6	tone_6	**7**	tone_6_noisy	**6**
7	tone_7	**8**	tone_7_noisy	**7**
8	tone_8	**9**	tone_8_noisy	**8**
9	tone_9	**10**	tone_9_noisy	**9**

DFT Plot – Uncorrupted Tone (no noise): Test Tone Audio Array: **tone_1**



DFT Plot – Noisy Tone: Test Tone Audio Array: **tone_6_noisy**



Matlab Code:

```
function [key_num] = DTMF_signal_decoder(test_tone, fs, figure_num)
%This function takes a time domain signal in '.mat' format and the sampling
%frequency as inputs. The function takes the FFT of the signal and creates
```

```
%a plot of the magnitude response. The function finds the presence of two
%particular frequencies correponding to the row and col. of a keypad press.
%The function ultimately returns the which key was pressed on the keypad.

str = 0;

xn = test_tone;                                    %time domain test tone
[DFTx, Fd] = plot_DFT_mag(xn, fs, figure_num);    %plot the DFT magn.

%find the col peak
Tacq = 0.5;
Mx = length(xn);
DFTx_mag  = abs(DFTx);
fa_max_1_index = find(DFTx_mag == max(DFTx_mag)); %retruns indexes for 2 pks
                                                   % (as an array) of col
DFTx_max_1 = DFTx_mag(fa_max_1_index(1))/(fs*Tacq); %mag. divided by M
fa_val(1) = fa_max_1_index(1)/Mx*fs;% index doesn't provide enough info
                                    % to get fa value, quantify the index #

%find the row peak
DFTx_mag([fa_max_1_index(1) - 5: fa_max_1_index(1) + 5]) = 0; %set max of col to 0
DFTx_mag([fa_max_1_index(2) - 5: fa_max_1_index(2) + 5]) = 0; %set max of col to 0

fa_max_2_index = find(DFTx_mag == max(DFTx_mag)); %returns indexes for 2 pks
                                                   %(as an array) of row
DFTx_max_2 = DFTx_mag(fa_max_2_index(1))/(fs*Tacq); %mag. divided by M
fa_val(2) = fa_max_2_index(1)/Mx*fs;% index doesn't provide enough info
                                    % to get fa value, quantify the index #

fa_val = sort(fa_val, 'descend');   % in case the row and col peak height
                                    % is not consistent

%determine which key was pressed
fc1 = 1209; fc2 = 1336; fc3 = 1477; fc4 = 1633;
fr1 = 697; fr2 = 770; fr3 = 852; fr4 = 941;
thresh = 30;

%row 1
if ((fc1 - thresh < fa_val(1) && fa_val(1) < fc1 + thresh) && (fr1 - thresh <
fa_val(2) && fa_val(2) < fr1 + thresh))
    key_num = 1;
elseif ((fc2 - thresh < fa_val(1) && fa_val(1) < fc2 + thresh) && (fr1 - thresh <
fa_val(2) && fa_val(2) < fr1 + thresh))
   key_num = 2;
elseif ((fc3 - thresh < fa_val(1) && fa_val(1) < fc3 + thresh) && (fr1 - thresh <
fa_val(2) && fa_val(2) < fr1 + thresh))
   key_num = 3;
elseif ((fc4 - thresh < fa_val(1) && fa_val(1) < fc4 + thresh) && (fr1 - thresh <
fa_val(2) && fa_val(2) < fr1 + thresh))
   key_num = 'A';
   str = 1;
%row 2
elseif ((fc1 - thresh < fa_val(1) && fa_val(1) < fc1 + thresh) && (fr2 - thresh <
fa_val(2) && fa_val(2) < fr2 + thresh))
    key_num = 4;
elseif ((fc2 - thresh < fa_val(1) && fa_val(1) < fc2 + thresh) && (fr2 - thresh <
fa_val(2) && fa_val(2) < fr2 + thresh))
   key_num = 5;
```

```
elseif ((fc3 - thresh < fa_val(1) && fa_val(1) < fc3 + thresh) && (fr2 - thresh <
fa_val(2) && fa_val(2) < fr2 + thresh))
   key_num = 6;
elseif ((fc4 - thresh < fa_val(1) && fa_val(1) < fc4 + thresh) && (fr2 - thresh <
fa_val(2) && fa_val(2) < fr2 + thresh))
   key_num = 'B';
   str = 1;
%row 3
elseif ((fc1 - thresh < fa_val(1) && fa_val(1) < fc1 + thresh) && (fr3 - thresh <
fa_val(2) && fa_val(2) < fr3 + thresh))
    key_num = 7;
elseif ((fc2 - thresh < fa_val(1) && fa_val(1) < fc2 + thresh) && (fr3 - thresh <
fa_val(2) && fa_val(2) < fr3 + thresh))
   key_num = 8;
elseif ((fc3 - thresh < fa_val(1) && fa_val(1) < fc3 + thresh) && (fr3 - thresh <
fa_val(2) && fa_val(2) < fr3 + thresh))
   key_num = 9;
elseif ((fc4 - thresh < fa_val(1) && fa_val(1) < fc4 + thresh) && (fr3 - thresh <
fa_val(2) && fa_val(2) < fr3 + thresh))
   key_num = 'C';
   str = 1;
%row 4
elseif((fc1 - thresh < fa_val(1) && fa_val(1) < fc1 + thresh) && (fr4 - thresh <
fa_val(2) && fa_val(2) < fr4 + thresh))
    key_num = '*';
    str = 1;
elseif ((fc2 - thresh < fa_val(1) && fa_val(1) < fc2 + thresh) && (fr4 - thresh <
fa_val(2) && fa_val(2) < fr4 + thresh))
   key_num = 0;
elseif ((fc3 - thresh < fa_val(1) && fa_val(1) < fc3 + thresh) && (fr4 - thresh <
fa_val(2) && fa_val(2) < fr4 + thresh))
   key_num = '#';
   str = 1;
elseif ((fc4 - thresh < fa_val(1) && fa_val(1) < fc4 + thresh) && (fr4 - thresh <
fa_val(2) && fa_val(2) < fr4 + thresh))
   key_num = 'D';
   str = 1;
else
    key_num = 'N/A';
    str = 1;

end

%display pressed key on figure
if (str == 1)
    text(3850, 0.9, key_num, 'FontSize',14)
else
    text(3850, 0.9, num2str(key_num), 'FontSize',14)
end

end
```

2) Touchtone Sequence Decoder

Functional Requirements:

- Your program should properly handle either **7-digit** (no area code) **or 10-digit** (with area code) touchtone sequences. (No need to handle country codes or long distance prefixes).
- Once 10 digits are detected, your program should stop looking for tones. (**Extra tones should be ignored**.) Once finished, you should verify that you ended up with either 7 or 10 digits; and indicate an "Error – Invalid Sequence" if it did not. (Extra tones beyond 10 should NOT generate an error.]
- The decoding should work properly even when there are **harmonics** of the touchtones from clipping or other distortions, or in the presence of **noise**.
- Each individual tone will be **separated by a pause** (no tone) that also can last from 0.2 – 10 seconds. (Note: There may still be noise during the pause.)
- Your design should be able to function with **different time durations** for each of the individual touch tones (with each varying in length from 0.2 – 3.0 seconds)

Output:

- Your Matlab program should display the final decoded 7 or 10 digit button sequence in the Matlab Command Window, or an error message if an invalid sequence was provided.
- Your program should also display a **spectrogram**( ) plot (using the built-in Matlab function to plot it) of nearly all of the original tone samples. (You may have to truncate some samples from the end to use a fixed spectrogram segment block size.) From this, you should be able to visually pick out where each different tone starts and ends, and where the frequency peaks are in each segment (to help you check your results).
- Since you are probably using your tone decoder from part 1 above, your program will also produce spectrum plots of each sub-segment processed. These should also be helpful for debugging and verifying your program. You do not need to include more examples of these in this part of your project report.

Design:

1. **Descriptions of the algorithms** you used for
   a. Separating the individual tones out from the multiple tone/pause sequence. (Finding the beginning and ends of different tones/pauses, or removing redundant results)

      The program loops through 0.1 sec chunks of the audio file at a time. There are 2 counters (key_hold, and invalid_count), that are used to count consecutive 0.1sec pauses and same key strokes. The function confirms that a key has been pressed once key_hold reaches 2 (0.2 sec). Key_hold will keep counting until a pause (noise) is detected. Once the pause count (invalid_count) reaches 0.2 sec, the function confirms that a pause has occurred and denotes that a key can now be pressed. The invalid_count keeps incrementing until a new key has been decoded.

   b. Limiting the decoding to 10 keystrokes.

      There is a counter called "total_keys" that increments every time the key_hold count gets to 2 (0.2 sec). After the counter reaches 10, the function exits the loop that cycles through data and outputs the 10 keystrokes.

2. **Explain your selection for the FFT size** used in your FFT signal processing: What value did you choose? How did you arrive at this particular value as your choice?

   The FFT size selected was 800 samples (would be taking the 1024 point fft). This was done because the sampling rate is 8kHz and therefore 800 points would account for 0.1 sec in the audio file. Given that the minimum key/pause time is 0.2 sec, 0.1 sec blocks allowed for decent time and frequency resolution after multiple trials.

3. **Matlab code listing** (with appropriate comments)

```
function [sequence] = DTMF_sequence_decoder(audio_file, fs)
% This function takes two inputs: audio_file, fs. Based on the provided
% sampling frequency of the file (fs), the function chops up the file into
% 0.1sec chuncks and procedes to decode the key sequence. The funtion will
% display the key times and pause times as well as determine if the
% sequence is an invalid phone number
```

```
    chop = fs/10;                                   %chunk of data (0.1 sec of data)
    interval = chop/fs;
    chop_num = floor(length(audio_file)/chop);  %number of total chunks of data
    total_keys = 0;                                 %counter for keys received
    ready = 1;                                      %variable to receive new key
    invalid_count = 0;                              %counter for number of pause chunks
    n = 0;                                          %loop counter
    key_hold = 0;                                   %key counter

    %cycle through data
    while(n <= chop_num-1)

        section = audio_file((n*chop +1):(n+1)*chop);   %chunk of data to look at

        [key] = DTMF_signal_decoder(section, fs, 1);    %get key
        if (key ~= 'N/A')                               %if key received
            key_hold = key_hold +1;                     %increment key hold time
            if (ready == 1)                             %ready to receive new key
                if (total_keys == 9)
                    n = chop_num-1;                     %end loop if 10 keys
                    sequence(10) = key;                 %final key
                    total_keys = 10;
                end
                if (key_hold > 1)                           %key held for 0.2 sec
                    total_keys = total_keys + 1;            %increment key counter
                    sequence(total_keys) = key;             %add to key sequence
                    ready = 0;                              %not ready to receive

                    %print pause time
                    disp('pause time:')
                    disp(invalid_count*interval)

                    invalid_count = 0;                      %reset pause count
                end
            end

        else                                            %pause received
            if (invalid_count > 0)
                ready = 1;                              %ready to receive new count
                if (key_hold ~= 0)
                    %print key time
                    disp('key time:')
                    disp(key_hold*interval)
                    key_hold = 0;
                end
            end
            invalid_count = invalid_count + 1;          %increment pause count
        end
        if (invalid_count == 10*fs/chop + 1)            %if pauses equal 10s
            n = chop_num-1;                             %end loop
        end
        n = n + 1;
    end

    disp(sequence)                                      %print key sequence
    if (total_keys ~=7 && total_keys ~=10)
        disp('Error - Invalid Sequence')                %invalid bc not 10 or 7 keys
    end

    %plot spectrogram
    figure
    window = 128; nonoverlap = 120;
    spectrogram(audio_file, window, nonoverlap, 1024, fs)
end
```

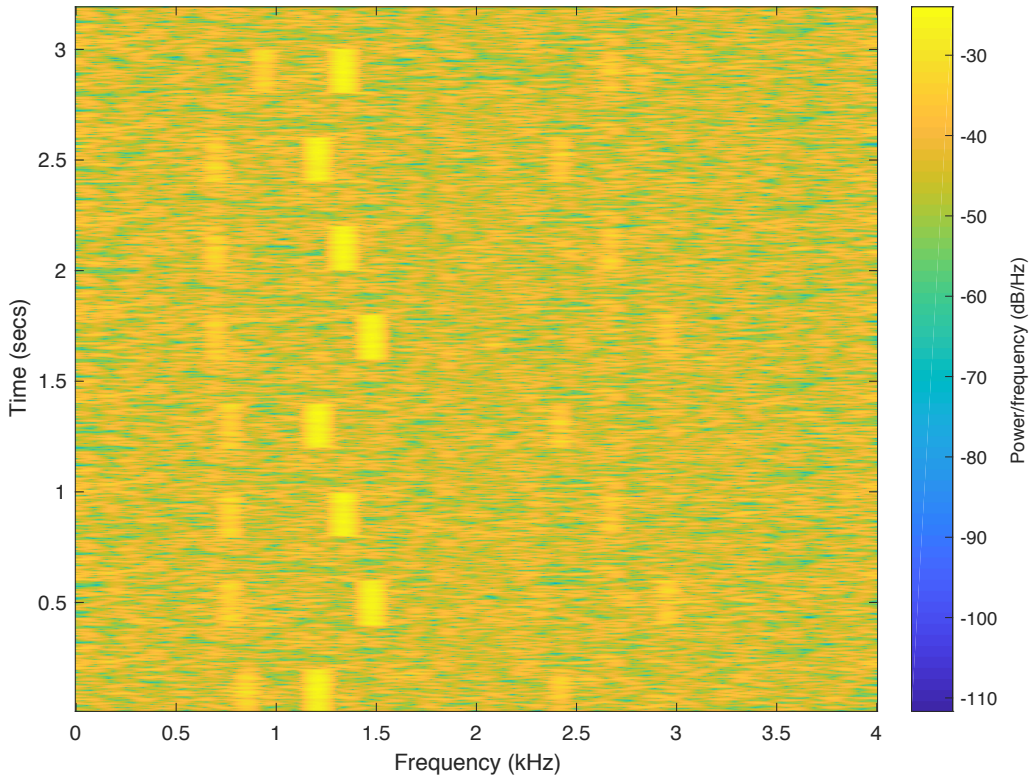
Test Cases:

1. An **invalid length** sequence (invalid # of key tones)
2. A sequence with **> 10 individual tones**.
3. A 10-tone sequence with different, valid lengths for all tones and pauses, with **no noise**. Include at least one example of each of the following in the sequence:
    a. A minimum time length tone.
    b. A maximum time length tone.
    c. A minimum time length pause.
    d. A maximum time length pause.
4. A 10-tone sequence with different, valid lengths for all tones and pauses, **with noise** in all tones and pauses. Include at least one example of each of the following in the sequence:
    a. A minimum time length tone.
    b. A minimum time length pause.
    c. Three redundant key tones in a row (like: …8-8-8…)
5. A **7-tone**, noisy sequence that **begins with a no-tone (noisy) pause**.

Test Results:

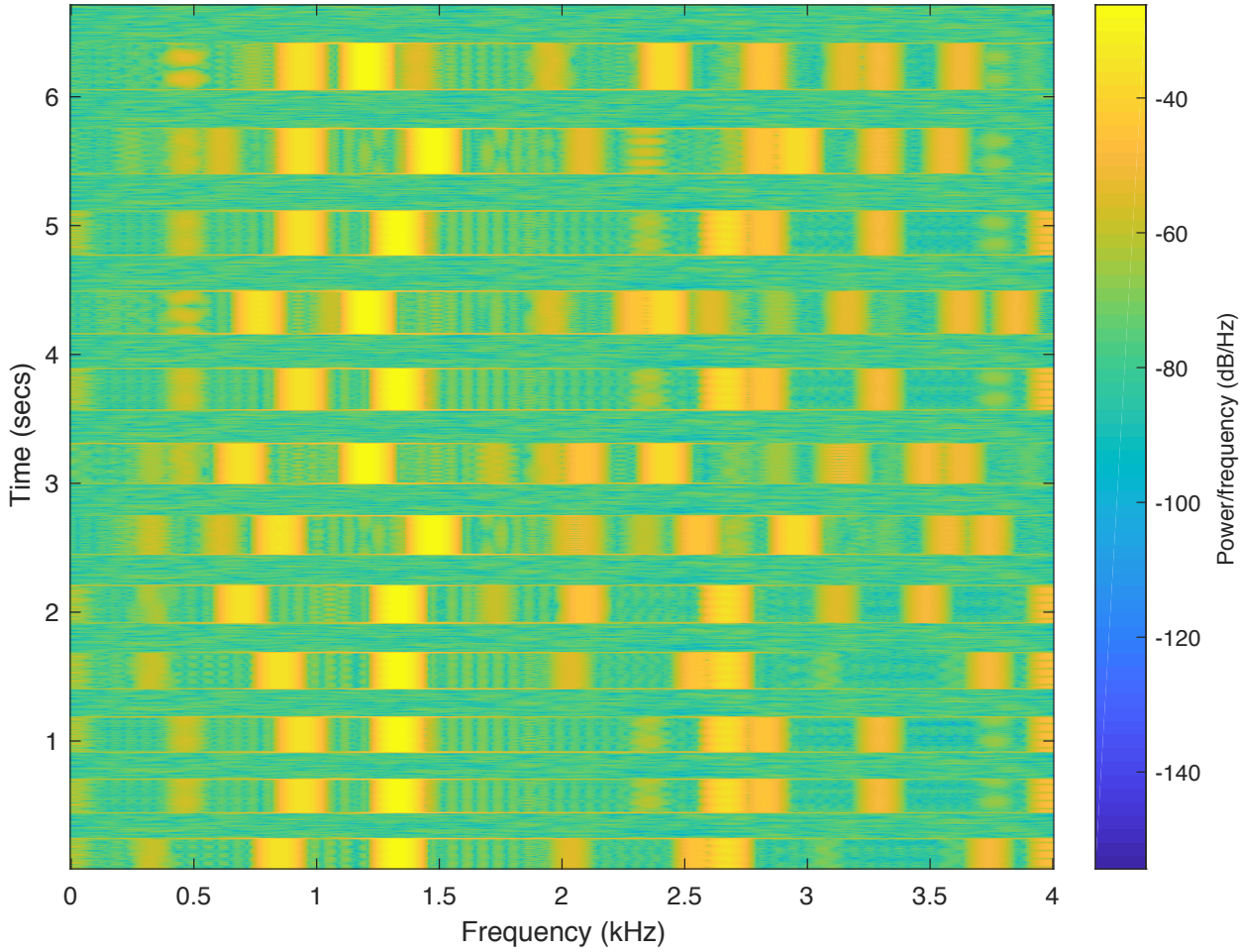
Test Case #:	**1**	**Test Case Description:**			**Invalid length sequence**				**Test Array Name:**		**TestCase1**	
Sequence:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**(11)**	**Noisy?**
Key:	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**				Tones Noisy? ↓
Key Tone Duration:	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**				**Yes**
Pause Duration:	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**	**0.2**					**Yes**
Decoded Key/Error	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**				Pauses ⇑ Noisy?

Spectrogram Plot – Test Case 1:



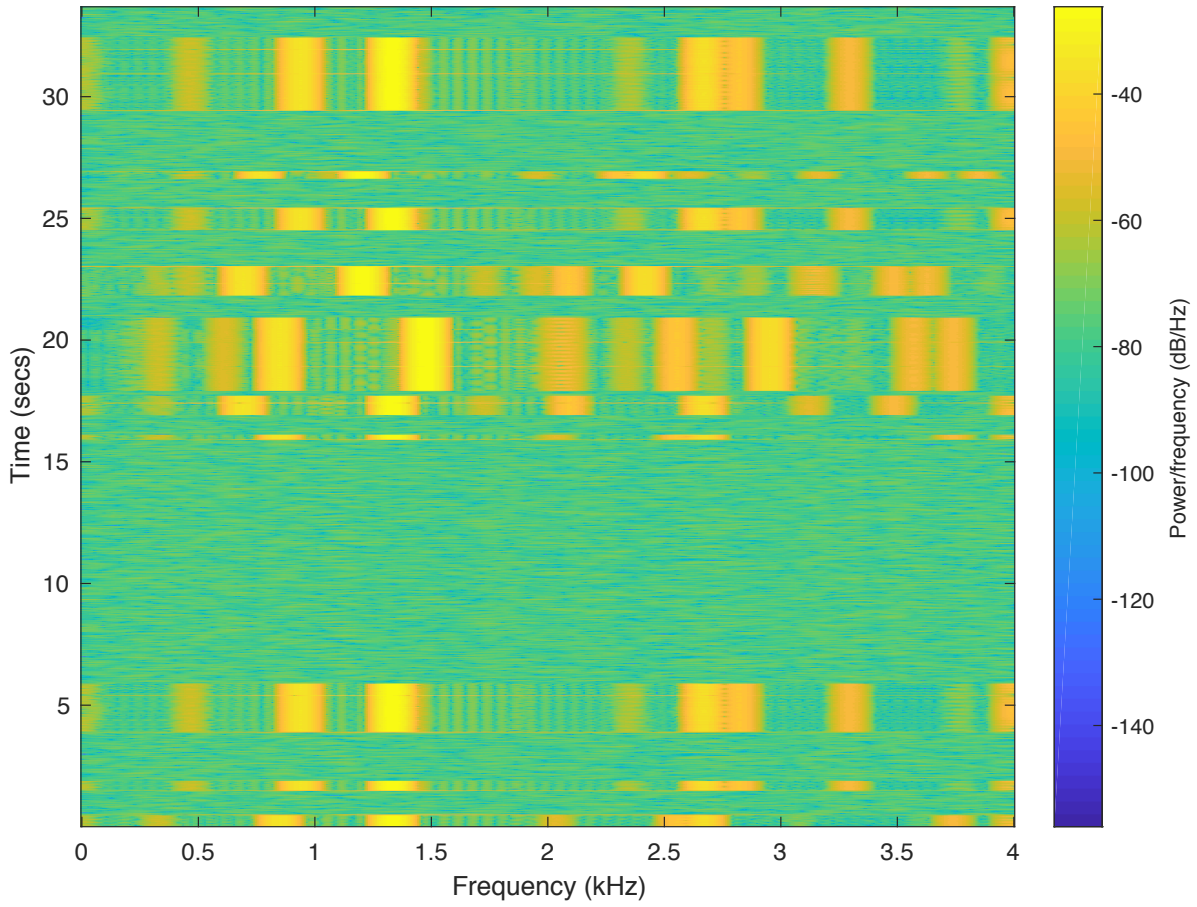
Test Case #:	2	Test Case Description:			Sequence with > 10 tones				Test Array Name:		TestCase2	
Sequence:	1	2	3	4	5	6	7	8	9	10	11	Noisy?
Key:	8	0	0	8	2	9	1	0	4	0	#	Tones Noisy? ↓
Key Tone Duration:	0.6	0.3	0.3	0.3	0.4	0.3	0.4	0.3	0.4	0.4		No
Pause Duration:	0.3	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2			No
Decoded Key/Error	8	0	0	8	2	9	1	0	4	0		Pauses ⇑ Noisy?

Spectrogram Plot – Test Case 2:



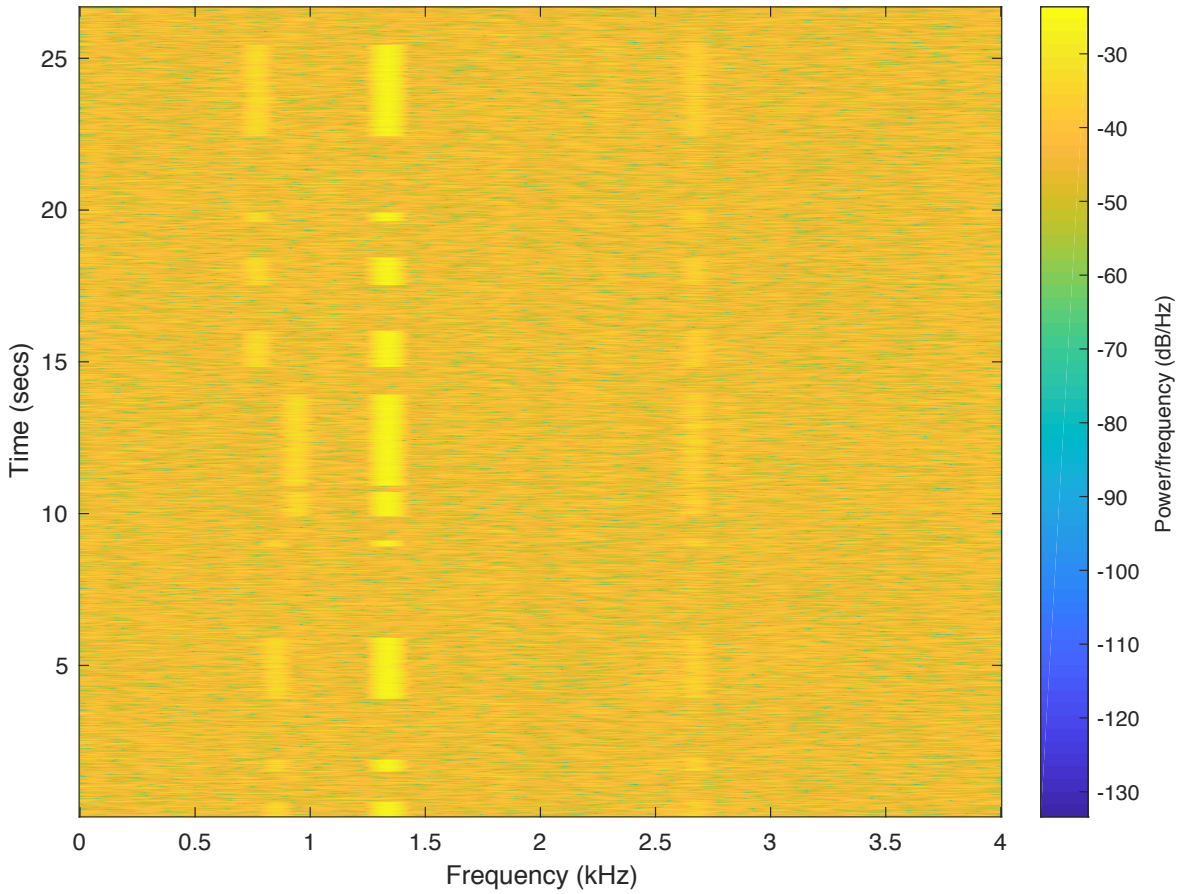
Test Case #:	3	Test Case Description:			10 valid tones - no noise				Test Array Name:		TestCase3	
Sequence:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**(11)**	**Noisy?**
Key:	**8**	**0**	**0**	**8**	**2**	**9**	**1**	**0**	**4**	**0**		Tones Noisy? ↓
Key Tone Duration:	**0.5**	**0.4**	**2**	**0.2**	**0.8**	**3**	**1.2**	**0.9**	**0.3**	**3**		**No**
Pause Duration:	**1**	**2**	**10**	**0.8**	**0.2**	**0.9**	**1.5**	**1.2**	**2.6**			**No**
Decoded Key/Error	**8**	**0**	**0**	**8**	**2**	**9**	**1**	**0**	**4**	**0**		Pauses ⇑ Noisy?

Spectrogram Plot – Test Case 3:



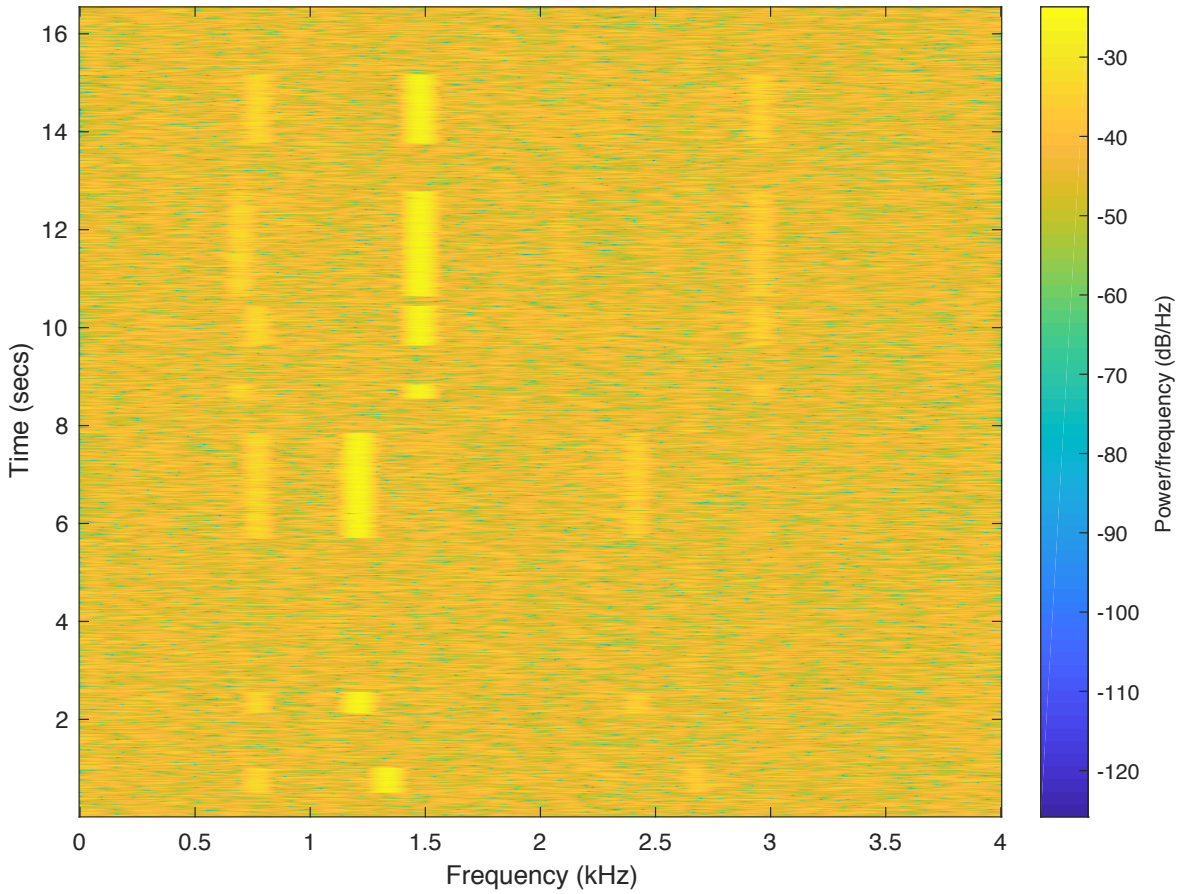
Test Case #:	4	Test Case Description:			10 valid tones - with noise (with 3^{+} redundant keys)				Test Array Name:		TestCase4	
Sequence:	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**(11)**	**Noisy?**
Key:	**8**	**8**	**8**	**8**	**0**	**0**	**5**	**5**	**5**	**5**		Tones Noisy? ↓
Key Tone Duration:	**0.5**	**0.4**	**2**	**0.2**	**0.8**	**3**	**1.2**	**0.9**	**0.3**	**3**		**Yes**
Pause Duration:	**1**	**2**	**3**	**0.8**	**0.2**	**0.9**	**1.5**	**1.2**	**3**			**Yes**
Decoded Key/Error	**8**	**8**	**8**	**8**	**0**	**0**	**5**	**5**	**5**	**5**		Pauses ⇑ Noisy?

Spectrogram Plot – Test Case 4:



Test Case #:	5	Test Case Description:			7-tone, noisy sequence; begins with a noisy pause				Test Array Name:		TestCase5	
Sequence:	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**(8)**	**(9)**	**(10)**	**Noisy?**
Key:		**5**	**4**	**4**	**3**	**6**	**3**	**6**				Tones Noisy? ↓
Key Tone Duration:		**0.6**	**0.5**	**2.3**	**0.4**	**0.8**	**2.2**	**1.5**				**Yes**
Pause Duration:	**0.5**	**1**	**3**	**0.6**	**0.7**	**0.2**	**0.9**					**Yes**
Decoded Key/Error		**5**	**4**	**4**	**3**	**6**	**3**	**6**				Pauses ⇑ Noisy?

Spectrogram Plot – Test Case 5:



Conclusions:

Summarize one or two learnings about spectral analysis using the Discrete Fourier Transform or Fast Fourier Transform that this project helped you understand better. Also describe any particular challenges that you had to overcome, and at least one suggestion for improvement of this lab in the future.

Name: Aiku Shintani

Conclusions: Throughout the course of this project, the Discrete Fourier Transform was utilized to implement a signal and sequence decoder for touch tones. In the first part of this project, a time domain signal, of a touchtone signal, was set as input to a custom-made function which would take the FFT of the signal and plot its spectral content. It was particularly challenging to find a logical method to extract the second highest peak in the

magnitude response. The second highest peak of the touchtone signal corresponded to a particular row of the keypad while the highest peak of the touchtone signal corresponded to a particular column of the keypad. This process involved development of an understanding of how the Matlab user interface works in terms of the execution of a function/script and how workspace variables are affected.

In the second part of the project, a function was created for decoding sequences of touch tones. Once again, this was a time domain signal with spectral content which was interpreted using DFT/FFT methods. In order to successfully receive every touchtone in the sequence correctly, numerous constraints for timing were utilized. The most difficult challenge of this project was to satisfy these constraints for all of the provided test cases since the test cases were designed to find bugs in the program. To implement a function such as this in the future, I would suggest that the programmer draw out all of the scenarios (of different types of sequences) prior to writing their code. Once this is done, the code will be less prone to bugs.

Name: Chris Adams

Conclusions: In this experiment, I learned the importance of considering the tradeoff between time and frequency resolution. Prior to this experiment, I expected that more samples meant better data. However, when dealing with time like in the DTMF sequence decoder, time is relatively important. This caused a lot of the issues when deciding the FFT size for decoding. The FFT size was chosen at 800 samples to get a time resolution of 0.1 sec. Another difficult aspect was that there is a large range associated with the expected timing durations of keys and pauses. For shorter times, finer time resolution is desired and vice versa. That is why 0.1 sec was chosen as an accurate medium that worked for every test case. If the audio files were sampled at a higher rate, frequency resolution would improve which would allow for more time resolution.

Another key takeaway from this project is how to use the FFT of a signal to decode something such as a keystroke. It was amazing to apply our knowledge of frequency domain plots to determine real keystrokes associated with certain tones. In our program, we looked at the two highest peaks while using some conditionals to discard bad results. For the purposes of this project, that was sufficient. However, I am excited to use correlation in future projects to get more definite results.