Introduction

Dual Tone Multi-Frequency (DTMF) system is a method used to dial phone numbers. Every digit is associated with two frequencies. DTMF keypads are laid out on an 4×4 matrix, each row represents low frequency and each column represents high frequency. Each key pressed on a phone generates two tones of specific frequencies. This is shown in Table 1.

The signal is a linear combination of two pure tone (pure sine waves) sounds with the frequencies as shown in Table 1. Furthermore, the signal is zero when no key is pressed and there is also noise in the signal.

The objective of this program is to find the phone number which is typed in the given sound fragment. To do this, Fast Fourier Transforms will be used, this is a method used in signal processing. The FFT algorithm is one of the most important mathematical algorithms. It is generally used to extract frequencies from a more or less periodic signal.[1]

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

Table 1: The Numeric Keypad with Correspinding DTMF Frequencies.

Syntax

This function uses five input arguments, a $n \times 1$ vector with the signal, the sampling rate which are required for the program to work. And three optional input variables, concerning sound playback, plotting output and the debug information. The sampling rate is the number of measurements per unit time. In the case of the given sound samples, this is either 8192 or 4096 samples per second. By giving a `true` argument to the `playSound` variable, the sound is played. By giving a `true` argument to the `showPlot` variable, a plot is displayed of the sound signal and FFT diagrams of every tone found. By giving a `true` argument to the `debug` variable, the debug information is shown. These parameters are optional and are off by default.

The output is a row vector with the digits of the phone number in `char` format. Furthermore, if asked the plots are shown as well as the debug information. An example of this debug information can be found in Appendix I.

[1] van den Berg, J.B., Planqué, B., 11 January 2017, Lecture Notes on Numerical Mathematics

Description of the Method

First of all, the optional input arguments are set to their default values if no argument is given to them. These are all set to off as default. Then, the locations of the tones are identified. This is done by looking at the top half of the sample data, these peaks in the sound fragment will most likely be the tones. This is followed by looking for the pauses between the tones, this is identified by looking for gaps the length of $\frac{1}{10}$ of the sampling rate or $\frac{1}{10}$ of a second. If there is a gap of such a length, the tones are considered to be different tones. This process is continued for all the tones.

As it is possible with this code to display multiple plots, this is described in lines 53-78, this is however self-explanatory. This also applies to the displaying of the debug information, this prints the length of the input, the amount of tones found and the percentage of data which was used. Furthermore, the DTMF data is saved with all the corresponding low and high frequencies as a row vector and a numeric keypad is created as a column vector.

A for-loop is created which continues for the length of the number of tones which were found. The beginning and the end of the tone are identified by using the location of the tone and location of the indices. FFT is applied to the found tone data. Because FFT is 'mirrored' only the first half of the created data is used.

Through the function `findpeaks` the peak frequencies in the FFT diagram and their amplitudes are found. You'd only want to find two frequencies that stand out. The minimal distance between the two peaks is the difference between the maximum of the low DTMF frequencies and the minimum of the high frequencies, 941 and 1209 Hz respectively. There is a possibility that the found frequency is not exactly the same as the DTMF frequency, that's why a margin of error of 50 Hz on both sides was applied. The final minimal peak distance is then $|941 - 1209| - 50 \cdot 2 = 168$ Hz.

The found frequencies are then compared to the DTMF frequencies, where the first peak will always correspond to the low frequency and the second peak to the high frequency. These frequencies are then mapped to a button on the numeric keypad by checking for a closest match on the defined frequencies in Table 1.

Example and Illustration

For the file `phonenumbers.mat`, with sampling rate 4096 Hz the function is called as follows: `extract_dtmf(number1,4096,false,true,true)`. Because both the plot variable and debug information variable are given a true argument, these are both displayed. The plots can be found in Figure 1 and Figure 2, furthermore, the debug information can be found in Appendix I. Figure 1 displays the whole sound. On the x-axis the time in seconds is displayed, on the y-axis the amplitude is given. The red transparent rectangles displays the used data in the original sample data. Furthermore, the red numbers define the number of found tones.

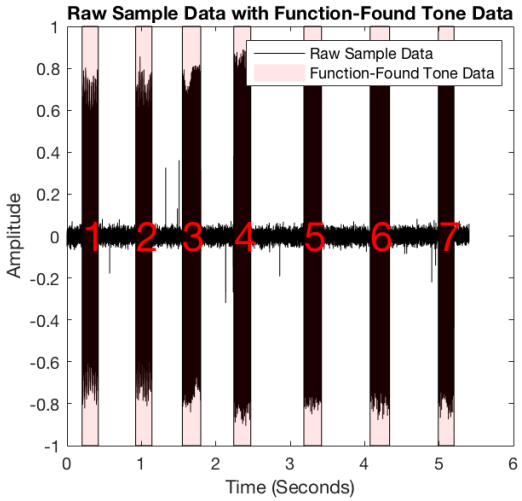

Figure 1: Raw Sample Data with Function-Found Tone Data of Sound Sample `number1`, with Sampling Rate 4096 Hz.

As this gives only a raw figure of the sound, a new plot is created which displays the results from the FFT of the found tones, in this case 7. This is shown in Figure 2. For each tone that has been found a subplot is created of the FFT peaks, these display both the frequency of the peaks as well as the amplitude. The frequencies are of most interest as these will give us the details about the dialed numbers.

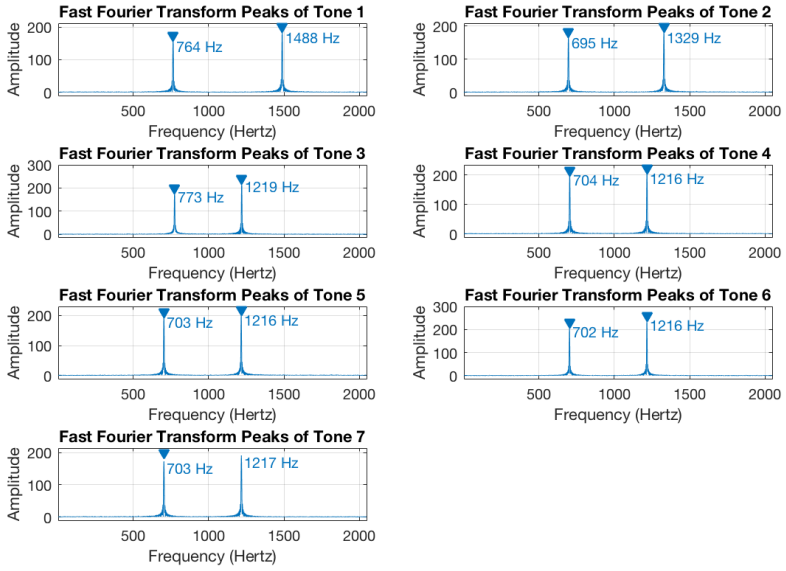

Figure 2: Fast Fourier Transform Peaks of the Function-Found Tone Data of Sound Sample `number1`, with Sampling Rate 4096 Hz.

Finally, the debug information, as can be found in Appendix I, shows us the detailed information and consequences of the FFT. For example, the first tone has FFT peak frequencies 764 Hz and 1488 Hz, these are closest to the DTMF frequencies of 770 Hz and 1477 Hz, therefore it can be concluded that at first number 6 has been pressed on the numeric keypad. In Table 2 all the extracted phone numbers can be found.

Sound File	Sampling Rate	Phone Number
`signal`	8,192 Hz	1 5 0 8 6 4 7 7 0 0 1
`number1`	4,096 Hz	6 2 4 1 1 1 1
`number2`	4,096 Hz	0 2 0 5 9 8 7 6 5 3
`number3`	4,096 Hz	0 6 4 4 2 4 4 9 0 1
`number4`	4,096 Hz	0 0 3 1 7 7 4 6 5 6 7 6 7
`number5`	4,096 Hz	0 0 1 2 0 2 4 5 6 1 4 1 4

Table 2: Function-Found Phone Numbers of All Given Samples.

Advantages and Drawbacks

The program is easy to understand and gives clear illustrations. Furthermore, it has a very low calculation time. It does, however, have some restrictions. First of all, if the wrong sampling rate is given to the function, the FFT calculations will be wrong. This will result in different peaks and therefore a different phone number. Also, if there is too much background noise, the program cannot distinguish the noise from the actual tones.

The program heavily relies on amplitude. If more than two high amplitude peaks are found, e.g. because of too much background noise, the program might not be able to determine the corresponding key correctly.

Performance

Like the previous functions, this one is also written with performance in mind. Showing figures and printing output in MATLAB is very heavy on the CPU, this is the reason it is off by default. The basic function without these options has a code-base of just 16 lines. It uses a minimal amount of loops and always vector operations when possible.

It is, however, possible to create a function without the use of any loops (see Appendix III). This is our alternative implementation. This function does not offer any nice graphs or debug information, and thus does not comply with the assignment. It is,however, very efficient code, and calculates the FFT by itself. We will not go in any detail on it works here, but you are free to try the code out yourself.

```
for i = 1:1000
    tic;
    extract_dtmf(signal,8192);extract_dtmf(number1,4096);
    extract_dtmf(number2,4096);extract_dtmf(number3,4096);
    extract_dtmf(number4,4096);extract_dtmf(number5,4096);
    timings(i) = toc;
end
mean(timings)
```

```
for i = 1:1000
    tic;
    extract_dtmf2(signal,8192);extract_dtmf2(number1,4096);
    extract_dtmf2(number2,4096);extract_dtmf2(number3,4096);
    extract_dtmf2(number4,4096);extract_dtmf2(number5,4096);
    timings(i) = toc;
end
mean(timings)
```

Figure 3: Scripts to Benchmark the Two Function Timings

The two functions were compared using the `tic` and `toc` command in MATLAB (see Figure 3). The test setup was a laptop with an Intel® Core™ i5-7360U@2.30 GHz processor running MATLAB R2019a on macOS 10.13.6 (17G7024). After running the above code the results were an average of 0.0421 seconds for our final code and a mere 0.0012 seconds of our alternative code. This a more than 35 times decrease in average function timings, while still yielding the same result. For a production environment this would be a more feasible solution, as debugging is not needed and function performance is more desired. However, conditions do of course apply, the same as our original implementation.

Appendix I: Example Function Output

```
The length of the input is 5.4041 seconds. Found 7 tones.
Using 10.21% of the input data.

Tone 1: Found FFT peak frequencies of 764 Hz and 1488 Hz.
These are closest to DTMF freqencies of 770 Hz and 1477 Hz.
This corresponds to value 6 on the numeric keypad.

Tone 2: Found FFT peak frequencies of 695 Hz and 1329 Hz.
These are closest to DTMF freqencies of 697 Hz and 1336 Hz.
This corresponds to value 2 on the numeric keypad.

Tone 3: Found FFT peak frequencies of 773 Hz and 1219 Hz.
These are closest to DTMF freqencies of 770 Hz and 1209 Hz.
This corresponds to value 4 on the numeric keypad.

Tone 4: Found FFT peak frequencies of 704 Hz and 1216 Hz.
These are closest to DTMF freqencies of 697 Hz and 1209 Hz.
This corresponds to value 1 on the numeric keypad.

Tone 5: Found FFT peak frequencies of 703 Hz and 1216 Hz.
These are closest to DTMF freqencies of 697 Hz and 1209 Hz.
This corresponds to value 1 on the numeric keypad.

Tone 6: Found FFT peak frequencies of 702 Hz and 1216 Hz.
These are closest to DTMF freqencies of 697 Hz and 1209 Hz.
This corresponds to value 1 on the numeric keypad.

Tone 7: Found FFT peak frequencies of 703 Hz and 1217 Hz.
These are closest to DTMF freqencies of 697 Hz and 1209 Hz.
This corresponds to value 1 on the numeric keypad.


ans =

    '6241111'
```

Appendix II: Full Function Implementation

```
function [numbers] = extract_dtmf(sampleData, fs, playSound, showPlot, debug)
%extract_dtmf Find DTMF signals and give corresponding numeric keypad
%values.
%   NUMBERS=EXTRACT_DTMF(SAMPLEDATA,FS,PLAYSOUND,SHOWPLOT,DEBUG) tries to
%   find DTMF signals in the input SAMPLEDATA, which is a vector consisting
%   of sampled data at sample rate FS, in Hertz.
%   The function uses Fast Fourier Transform to find the two peak
%   frequencies of DTMF in a window. A window is defined by a peak in
%   amplitude in the sound diagram. This function heavily relies on
%   amplitude.
%   The function will return a row vector consisting of all the digits as a
%   character.
%   It is optional to hear the sound by setting the input variable
%   PLAYSOUND to true. To show a plot of the input, set variable SHOWPLOT
%   to true. To show debug information, set DEBUG to true. All these
%   parameters are optional and off by default.
%   The plots will show a visualisation of the sound input and mark the
%   different windows it found, presumably containing DTMF signals. It will
%   then show the FFT frequencies found in another figure. To show the
%   mapping of these frequencies to numeric keypad digits, please enable
%   DEBUG.

if nargin < 5
    % Showing debug information is off by default.
    debug = false;
end
if nargin < 4
    % Showing plot is off by default.
    showPlot = false;
end
if nargin < 3
    % Playing sound is off by default.
    playSound = false;
end

if playSound
    % If wanted, the sound can be played.
    sound(sampleData,fs);
end

% Find location indices in the raw data where a peak is displayed, this is
% most probably a tone.
toneLocations = find(abs(sampleData)>.5*max(sampleData));

% Every tone has time between them. If between the found location indices
% above there is a gap of one tenth of the sampling frequency — or one
% tenth of a second — this is considered a different tone. The first tone
% starts at place 1 of the toneLocations. (See for-loop down below why it
% starts at 0 here). It ends at the first pause seen. Same with the last
```

```
50  % tone: it starts at the first jump and ends at the end of toneLocations.
51  toneIndices = [0; find(diff(toneLocations)>fs/10); length(toneLocations)];
52
53  if showPlot
54      % Close all current figures.
55      close all;
56      % Change figure sizes and position.
57      figure(1);
58      set(gcf,'Position',[0,100,600,600])
59      figure(2);
60      % Set the amount of subplots to fit the amount of tones.
61      amountOfTonePlots = floor(length(toneIndices)/2);
62      set(gcf,'Position',[600,100,600,600])
63
64      % Switch to figure 1.
65      figure(1);
66      % Convert the data to seconds and plot it.
67      dt = 1/fs;
68      t  = 0:dt:(length(sampleData)*dt)-dt;
69      plot(t,sampleData,'k');
70      % Add plot title and axis labels.
71      title('Raw Sample Data with Function-Found Tone Data');
72      xlabel('Time (Seconds)');
73      ylabel('Amplitude');
74      % Create hidden line for use in legend, as rectangle does not show.
75      rectangleLegend = line(NaN,NaN,'LineWidth',10,'Color',[1 0 0 .1]);
76      % Add the legend.
77      legend('Raw Sample Data','Function-Found Tone Data');
78  end
79
80  if debug
81      % Print the length of the input and the amount of tones found.
82      fprintf(['The length of the input is ' num2str(length(sampleData)/fs) ...
83          ' seconds. Found ' num2str(length(toneIndices)-1) ' tones.\n' ...
84          'Using ' num2str(round(100*length(toneLocations)/length(sampleData),2)) ...
85          '%% of the input data.\n\n']);
86  end
87
88  % Save the DTMF data and create a numeric keypad.
89  freqLow  = [ 697     770     852     941  ];
90  freqHigh = [ 1209    1336    1477    1633 ];
91  numpad   = ['123A'; '456B'; '789C'; '*0#D'];
92
93  % Loop over every tone.
94  for i=1:length(toneIndices)-1
95      % The toneIndices vector starts at location 0. The first location in
96      % the vector is 1. It ends at the next tone indice.
97      % The next tone will start at the previous tone location+1 and end at
98      % the next. This continues for all tones. Then extract the data from
99      % the original input.
100     currentToneBegin = toneLocations(toneIndices(i)+1);
```

```
101     currentToneEnd   = toneLocations(toneIndices(i+1));
102     currentToneData  = sampleData(currentToneBegin:currentToneEnd);
103 
104     if showPlot
105         % Switch to plot 1.
106         figure(1);
107         % Create a red transparent rectangle over the used data in the
108         % original sample.
109         rectangle('FaceColor',[1 0 0 .1], ...
110         'Position',[currentToneBegin/fs,-1,...
111         (currentToneEnd-currentToneBegin)/fs,2]);
112         text('Position',[currentToneBegin/fs,0],...
113             'string',i,'Color','red','FontSize',24);
114     end
115 
116     % Apply FFT to the found tone data and only use real values.
117     fftData = abs(fft(currentToneData,fs));
118     % FFT is 'mirrored,' so only use the first half.
119     fftData = fftData(1:end/2);
120 
121     % Find the two peak frequencies in the FFT diagram and their respective
122     % amplitudes.
123     [amplitude, freq] = findpeaks(fftData,'MinPeakHeight',.25*max(fftData),'
            MinPeakDistance',168);
124     % The first frequency will always be the lowest, and the second the
125     % highest. Find the closest match to DTMF frequencies.
126     [~, row]  = min(abs(freqLow-freq(1)));
127     [~, col]  = min(abs(freqHigh-freq(2)));
128     % Map the column and row to a button on the numeric keypad, create row
129     % vector.
130     numbers(i) = numpad(row,col);
131 
132     if showPlot
133         % Switch to figure 2 and subplot i.
134         figure(2);
135         subplot(amountOfTonePlots,2,i);
136         % Plot the peaks found above.
137         findpeaks(fftData,'MinPeakHeight',.25*max(fftData),'MinPeakDistance',168);
138         % Add text to the peaks with the corresponding frequency.
139         text('Position',[freq(1)+20,amplitude(1)-20],'string',[num2str(freq(1)) ' Hz'
                ],...
140             'Color',[0 0.4470 0.7410],'FontSize',10);
141         text('Position',[freq(2)+20,amplitude(2)-20],'string',[num2str(freq(2)) ' Hz'
                ],...
142             'Color',[0 0.4470 0.7410],'FontSize',10);
143         % Add plot title and axis labels.
144         title(['Fast Fourier Transform Peaks of Tone ' num2str(i)]);
145         xlabel('Frequency (Hertz)');
146         ylabel('Amplitude');
147     end
148 
```

```
149     if debug
150         % Print the found peak frequency and the closest DTMF frequency.
151         % Then print the corresponding button of the numeric keypad.
152         fprintf(['Tone ' num2str(i) ': Found FFT peak frequencies of ' num2str(freq
                (1)) ' Hz and ' num2str(freq(2)) ' Hz.\n']);
153         fprintf(['These are closest to DTMF freqencies of ' num2str(freqLow(row)) '
                Hz and ' num2str(freqHigh(col)) ' Hz.\n']);
154         fprintf(['This corresponds to value ' num2str(numbers(i)) ' on the numeric
                keypad.\n\n']);
155     end
156 end
```

Appendix III: Alternative Function Implementation

```
1  function [phone_number] = extract_dtmf2(data, sampling_rate)
2  sample = 0.03;
3  t = linspace(0,sample,sample*sampling_rate+1);
4  width = length(t);
5  height = floor(length(data)/width);
6  dataS = reshape(data(1:(height*width)),width,height);
7
8  LowF = [697 770 852 941];
9  HighF = [1209 1336 1477 1633];
10 w = exp(j*2*pi*LowF'*t);
11 dataLow = abs(w*dataS)/width;
12 w = exp(j*2*pi*HighF'*t);
13 dataHigh = abs(w*dataS)/width;
14
15 scale = 1:4;
16 threshold = 0.07;
17 low = scale*(dataLow>threshold);
18 high = scale*(dataHigh>threshold);
19
20 number = find(diff([0 low])>0);
21 row = low(number);
22 column = high(number);
23 numpad = ['123A'; '456B'; '789C'; '*0#D'];
24 diag(numpad(row,column))'
```