C0E 472: DIGITAL SIGNAL PROCESSING

**Lab 09: Encoding and Decoding Touch-Tone Signals**

INDEX NUMBER: 3050420

NAME: HARDY HAWA TUNTEIYA

COURSE: BSc. Biomedical Engineering

Year 4

# INTRODUCTION

This lab teaches how to design and implement bandpass FIR filters in MATLAB to decode Touch-Tone (DTMF) signals automatically. The main lab file is dtmfrun.m

The functions used: firfilt(), and freqz().

In the subsequent sections of the lab, we made four functions in .m files

Dtmfdesign.m, dtmfscore.m and dtmfcut.m, dtfmdial.m (found in appendix)

# LAB EXERCISE - DTMF Decoding

**4.1 Simple Bandpass Filter Design: dtmfdesign.m**

dtmfdesign function was made.

function hh = dtmfdesign(fb, L, fs)

% dtmfdesign: Design DTMF bandpass filters

% hh = dtmfdesign(fb, L, fs) returns a matrix (L by length(fb)) where each column contains

% the impulse response of a BPF, one for each frequency in fb.

% fb = vector of center frequencies

% L = length of FIR bandpass filters

% fs = sampling frequency

% Each BPF is scaled so that its frequency response has a maximum magnitude equal to one.

hh = zeros(L, length(fb)); % Pre-allocate hh matrix

for k = 1:length(fb)

nn = 0:(L-1);

hh(:, k) = cos(2 \* pi \* fb(k) \* nn / fs);

ww = 0:pi/500:pi;

HH = freqz(hh(:, k), 1, ww);

zz = max(abs(HH));

HH1 = freqz((1 / zz) \* hh(:, k), 1, ww); % Scale to max value equal to one

plot(ww, abs(HH1));

xlim([0 3.35]);

ylim([0 1]);

grid on;

hold on;

end

hold off;

end

d) Use dtmfdesign when L = 40 and fs = 8000Hz, using the eight DTMF Frequencies

centre\_freq= [697 770 852 941 1209 1336 1477 1633];

a = dtmfdesign(centre\_freq, 40, 8000);

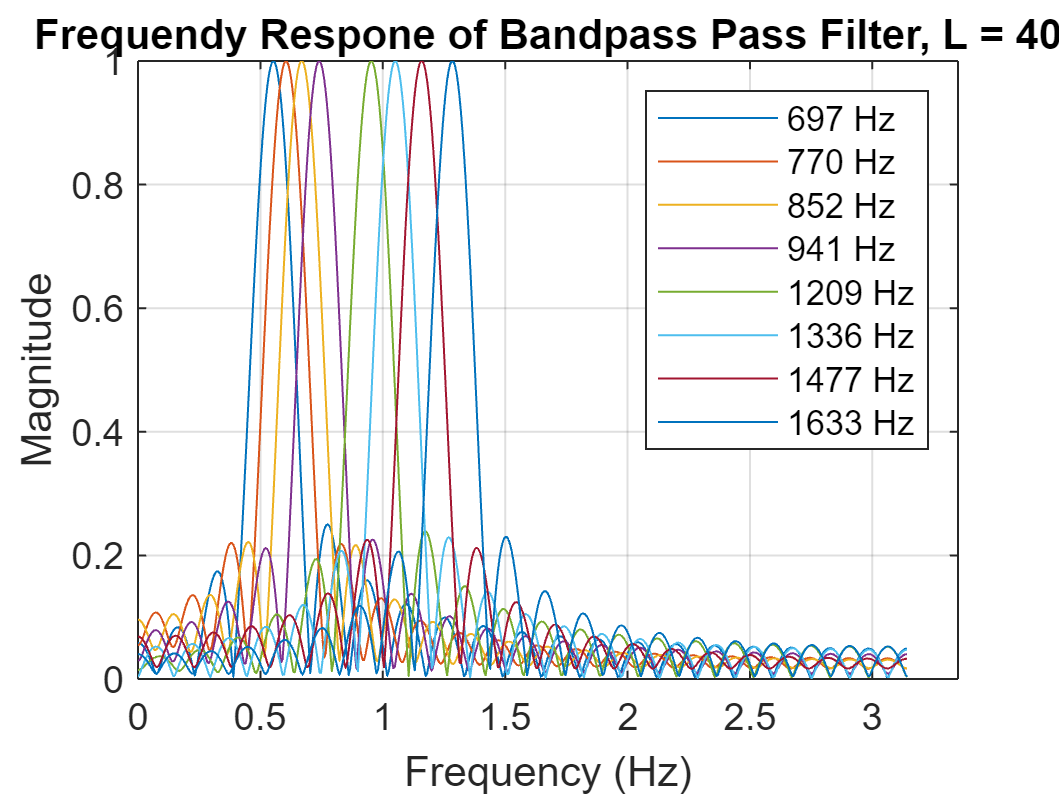
title('Frequendy Respone of Bandpass Pass Filter, L = 40');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

legend(arrayfun(@(f) sprintf('%d Hz', f), centre\_freq, 'UniformOutput', false));

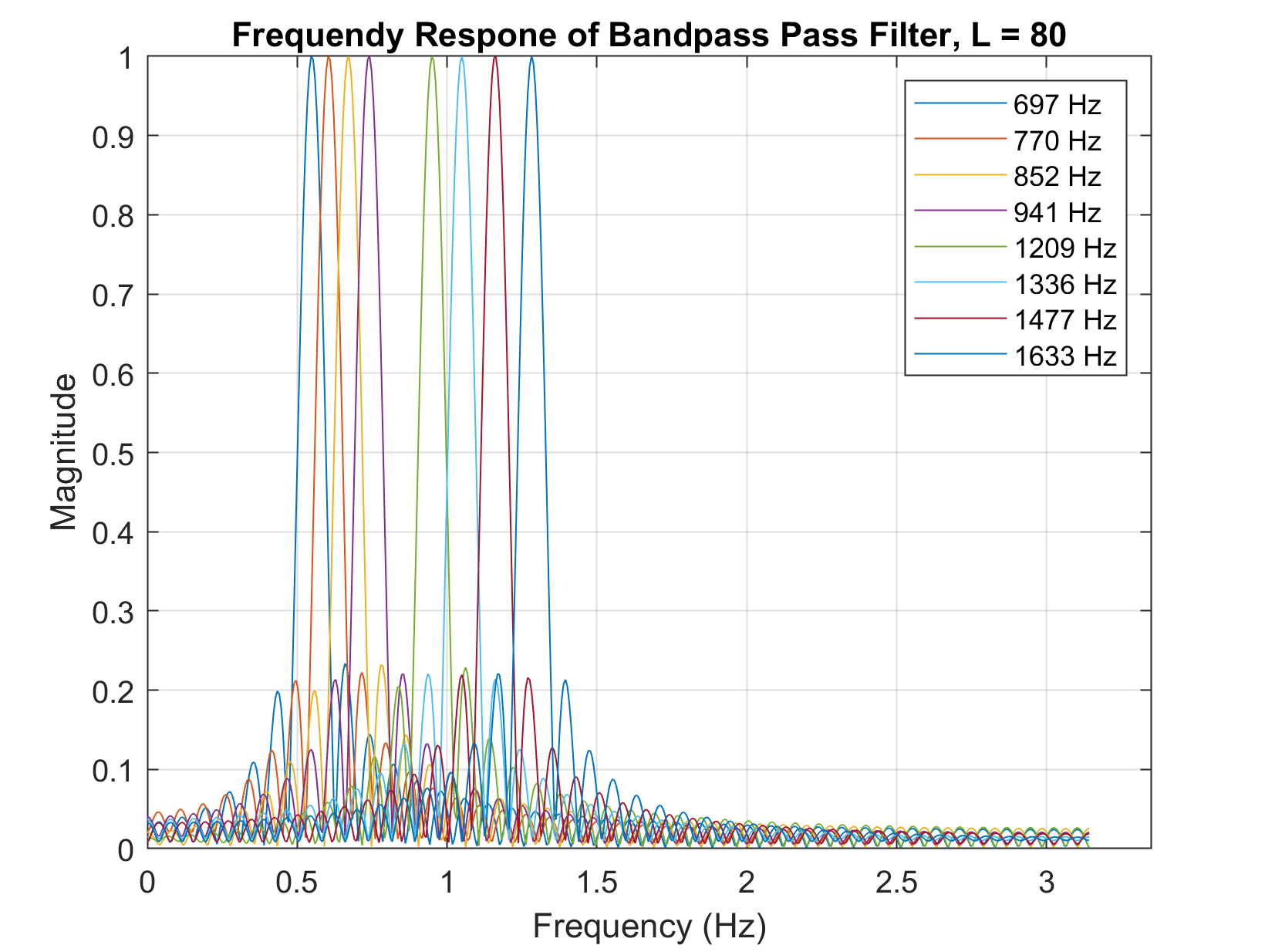
OUTPUT



e) Use dtmfdesign when L = 80 and fs = 8000Hz, using the eight DTMF Frequencies

Code is the same as in d, except for the second line  
a = dtmfdesign(centre\_freq, 80, 8000);

OUTPUT



f)i. Find Minimum Length L

Section f has two parts

%(f)Finding minimum length L (part 1)

centre\_freq = [697 770 852 941 1209 1336 1477 1633];

fs = 8000;

L = 2; % initial value for L

done = 0;

while (~done)

k = 0;

L = L + 1;

hh = []; % initiate

for i = 1:length(centre\_freq)

nn = 0:(L-1);

hh(:, i) = cos(2 \* pi \* centre\_freq(i) \* nn / fs); % filter impulse response

ww = 0:pi/1000:pi;

HH = freqz(hh(:, i), 1, ww);

HH1 = freqz((1 / max(abs(HH))) \* hh(:, i), 1, ww);

% frequency response of filter with maximum value equal to one

for j = 1:length(centre\_freq)

z = round(2 \* (centre\_freq(j) / fs) \* 1000);

if abs(HH1(z)) < 0.25 % stop-band proportion 25%

k = k + 1;

end

if k == 56

done = 1;

end

end

end

end

L = 88 % minimum L = 88

Thus L = 88

f)ii. Check if L satisfies the demand

f% (fii) Evaluation L whether satisfies the demand or not

L = 88;

centre\_freq = [697 770 852 941 1209 1336 1477 1633];

w\_loca = [1395 1541 1705 1883 2419 2673 2955 3267];

% the location for each eight frequencies’ peak value

hh = zeros(L, length(centre\_freq));

plot(0:0.001:pi, 0.25)

hold on

for i = 1:length(centre\_freq)

nn = 0:(L-1);

hh(:, i) = cos(2 \* pi \* centre\_freq(i) \* nn / fs); % filter impulse response

ww = 0:pi/8000:pi;

HH = freqz(hh(:, i), 1, ww);

HH1 = freqz((1 / max(abs(HH))) \* hh(:, i), 1, ww);

stem(ww(w\_loca), abs(HH1(w\_loca)));

% stem the specific point for each eight filters

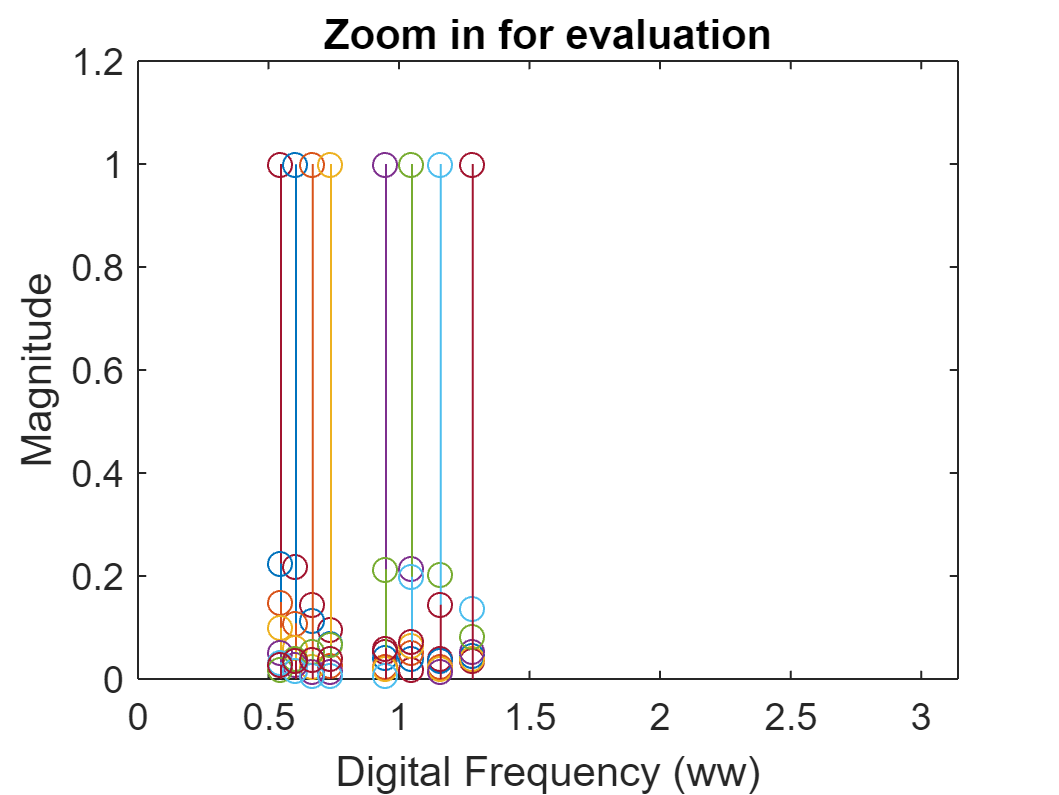
hold on;

title('Zoom in for evaluation')

xlabel('Digital Frequency (ww)'); ylabel('Magnitude');

end

OUTPUT



L = 88

**Comments on Selectivity**:

* For each filter, I observed the passband at points where the magnitude is close to 1, and the stopband when the magnitude is less than 0.25. This selectivity ensures only one frequency component lies within the passband and the others lie in the stopband.

**Analysis**:

* With L=40, the bandwidth is wider, making it more challenging to isolate specific frequencies.
* Increasing L to 80 narrows the bandwidth, improving selectivity, but also increases the computational complexity.

**Hardest Center Frequency to Meet Specifications**:

* Filters at lower frequencies (e.g., 697 Hz, 770 Hz) face more challenges due to the closeness of frequencies, requiring a higher L for better isolation.

**4.2 A Scoring Function**

Creating the dtmfscore.m function.

%DTMFSCORE

% Usage: sc = dtmfscore (xx, hh)

% returns a score based on the max amplitude of the filtered output

% xx = input DTMF tone

% hh = impulse response of ONE bandpass filter

%

% The signal detection is done by filtering xx with a length-L

% BPF, hh, and then finding the maximum amplitude of the output.

% The score is either 1 or 0.

% sc = 1 if max(|y[n]|) is greater than, or equal to, 0.59

% sc = 0 if max(|y[n]|) is less than 0.59

%

function sc = dtmfscore (xx, hh)

xx = xx\*(2/max(abs(xx))); %--Scale the input x[n] to the range [-2, +2]

yy=conv (hh, xx); % convolution of signal with BPF impulse response

if max(abs(yy))>=0.59 % binary output of signal presence in waveform

sc=1;

else

sc=0;

end

% figure; plot(abs(yy))

% Title ('Check for maximum amplitude function'), grid on

% xlabel('n'), ylabel('Magnitude')

end

Creating the dtmfcut.m function

function [nstart,nstop] = dtmfcut(xx,fs)

%DTMFCUT find the DTMF tones within x[n]

% usage:

% [nstart,nstop] = dtmfcut(xx,fs)

%

% length of nstart = M = number of tones found

% nstart is the set of STARTING indices

% nstop is the set of ENDING indices

% xx = input signal vector

% fs = sampling frequency

%

% Looks for silence regions which must at least 10 millisecs long.

% Also the tones must be longer than 100 msec

xx = xx(:)'/max(abs(xx)); %-- normalize xx

Lx = length(xx);

Lz = round(0.01\*fs);

setpoint = 0.02; %-- make everything below 2% zero

xx = filter( ones(1,Lz)/Lz, 1, abs(xx) );

xx = diff(xx>setpoint);

jkl = find(xx~=0)';

%%%xx(jkl);

if xx(jkl(1))<0, jkl = [1;jkl]; end

if xx(jkl(end))>0, jkl = [jkl;Lx]; end

%%%jkl';

indx = [];

while length(jkl)>1

if jkl(2)>(jkl(1)+10\*Lz)

indx = [indx, jkl(1:2)];

end

jkl(1:2) = [];

end

nstart = indx(1,:);

nstop = indx(2,:);

To check the code

xx=dtmfdial('1',8000); % for testing one filter at 697Hz

OUTPUT

A close-up of a graph

Description automatically generated

**d. Why the maximum value for H(ejω) must be one for each bandpass filter**

This guarantees predictable and consistent behavior, allowing accurate DTMF tone detection. This normalization ensures the filter neither amplifies nor attenuates the signal at the center frequency, facilitating reliable detection and consistent scoring.

**4.3 DTMF Decode Function**

Implement the dtmfrun.m function.

function keys = dtmfrun(xx, L, fs)

center\_freqs = [697 770 852 941 1209 1336 1477 1633];

hh = dtmfdesign(center\_freqs, L, fs);

[nstart, nstop] = dtmfcut(xx, fs); % Find the beginning and end of tone bursts

xx = xx \* (2 / max(abs(xx)));

keys = [];

dtmf.keys = ...

['1','2','3','A';

'4','5','6','B';

'7','8','9','C';

'\*','0','#','D'];

% Uncomment and use this section if needed

% for ii=1: length(nstart)

% x\_seg=xx(nstart(ii): nstop(ii)); % Extract one DTMF tone

% score= [];

% for i=1:8

% score(i) = dtmfscore(x\_seg, hh(:, i));

% end

% jkl=find(score==1);

% if length(jkl)~=2 % error indicator

% keys(ii)=-1;

% else

% yy=jkl(2)-4;

% keys(ii)=dtmf.keys(jkl(1), yy);

% end

% end

for kk = 1:length(nstart) % cycle through each tone

n = [];

x\_1 = xx(nstart(kk):nstop(kk)); % Extract one DTMF tone

for i = 1:length(center\_freqs) % cycle through each filter

zz = dtmfscore(x\_1, hh(:, i));

n = [n, zz]; % create a vector of ones and zeros representing where the frequency components lie

end

aa = find(n == 1); % create a vector of indices where ones occur

% Check for impossible scores and skip if they are found

if length(aa) ~= 2 || aa(1) > 4 || aa(2) < 5

keys = [keys, 'error'];

continue

end

row = aa(1); % decode row position from aa

col = aa(2) - 4; % decode column position from aa

keys = [keys, dtmf.keys(row, col)]; % set keys equal to the current keys and the key found in this iteration

end

end

**4.4 Telephone Numbers**

Test the complete system with a given phone number.

% Example usage:

fs = 8000;

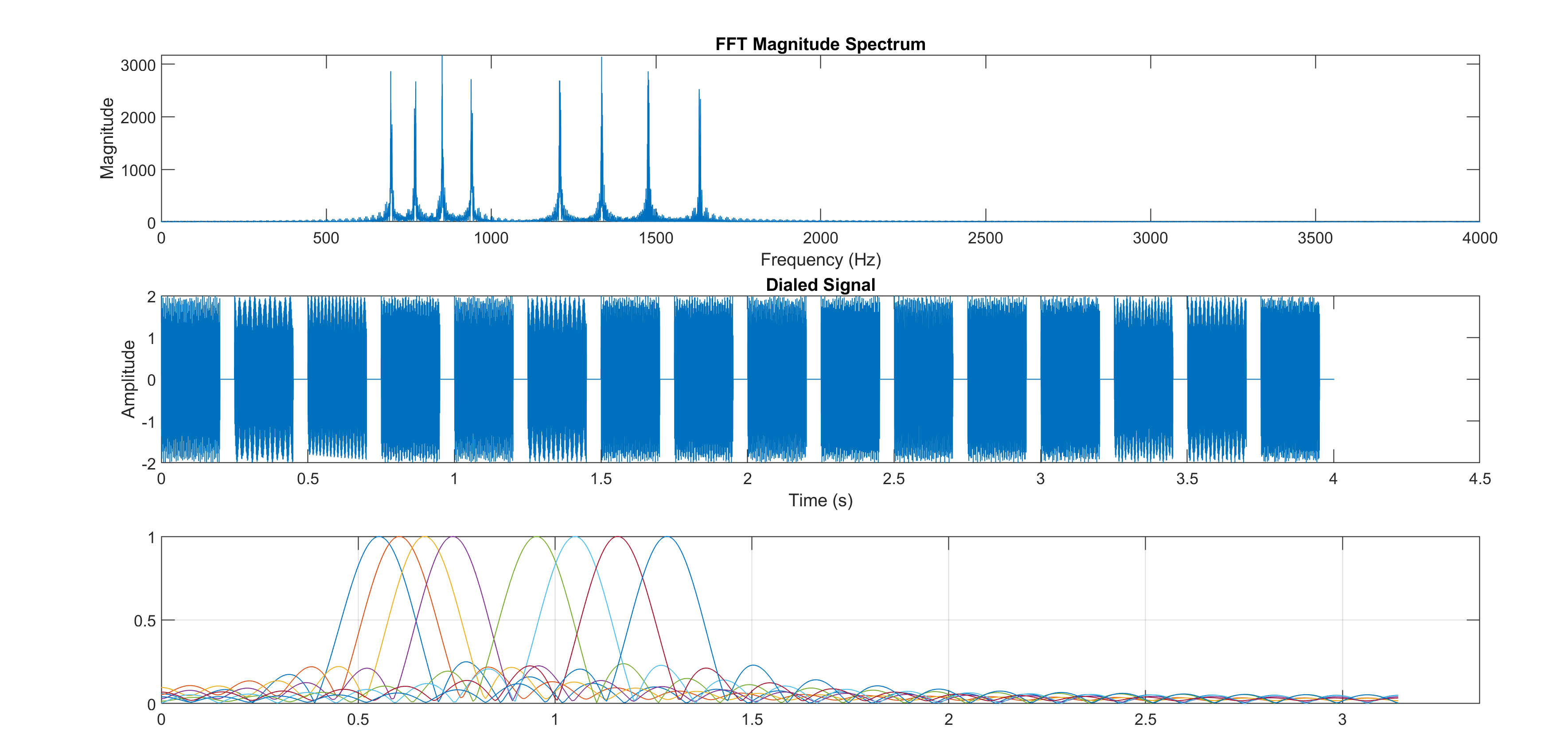
L = 40;

tk = '1234567890\*#ABCD';

xx = dtmfdial(tk, fs);

decoded\_keys = dtmfrun(xx, L, fs);

disp(decoded\_keys);



% Generate spectrogram for visualization

figure;

spectrogram(xx, 88, [], [], fs, 'yaxis');

title('Spectrogram of DTMF Signal');

OUTPUT

A diagram of a signal

Description automatically generated

# APPENDIX I: DTFMDIAL.M

function xx = dtmfdial(keyNames, fs)

%DTMFDIAL Create a signal vector of tones which will dial

% a DTMF (Touch Tone) telephone system.

%

% usage: xx = dtmfdial(keyNames, fs)

% keyNames = vector of characters containing valid key names

% fs = sampling frequency

% xx = signal vector that is the concatenation of DTMF tones.

dtmf.keys = ...

['1', '2', '3', 'A';

'4', '5', '6', 'B';

'7', '8', '9', 'C';

'\*', '0', '#', 'D'];

dtmf.colTones = ones(4, 1) \* [1209, 1336, 1477, 1633];

dtmf.rowTones = [697; 770; 852; 941] \* ones(1, 4);

% Initialize the signal vector

xx = [];

% Duration of each DTMF tone pair

toneDuration = 0.2; % 0.20 seconds

silenceDuration = 0.05; % 0.05 seconds of silence between tones

% Create each DTMF tone

for k = 1:length(keyNames)

key = keyNames(k);

% Find the row and column for the key

[ii, jj] = find(key == dtmf.keys);

% Get the row and column frequencies

f1 = dtmf.rowTones(ii, jj);

f2 = dtmf.colTones(ii, jj);

% Generate the time vector for the tone duration

tt = 0:(1/fs):toneDuration;

% Create the DTMF tone by summing two sinusoids

tone = cos(2 \* pi \* f1 \* tt) + cos(2 \* pi \* f2 \* tt);

% Concatenate the tone to the signal vector

xx = [xx tone];

% Add silence after the tone

silence = zeros(1, round(silenceDuration \* fs));

xx = [xx silence];

end

% Plot the FFT magnitude spectrum

N = length(xx);

X = fft(xx);

f = (0:N-1) \* (fs/N); % Frequency vector

figure;

subplot(3,1,1);

plot(f(1:N/2), abs(X(1:N/2)));

title('FFT Magnitude Spectrum');

xlabel('Frequency (Hz)');

ylabel('Magnitude');

% Plot the dialed signal

t = (0:N-1) / fs;

subplot(3,1,2);

plot(t, xx);

title('Dialed Signal');

xlabel('Time (s)');

ylabel('Amplitude');

% Plot the normalized frequency

normalized\_frequency = (0:N-1) \* (2\*pi/N); % Normalized frequency vector

subplot(3,1,3);

plot(normalized\_frequency(1:N/2)/pi, abs(X(1:N/2)));

title('Normalized Frequency');

xlabel('Frequency (\pi rad/sample)');

ylabel('Magnitude');

end