SivaDatta Budaraju,
17EC10010,
Group 31.

Assignment 3

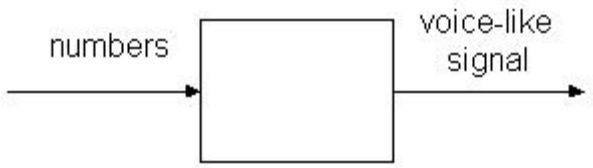
TITLE: DESIGN OF DUAL TONE MULTI-FREQUENCY (DTMF) CODER/DECODER

OBJECTIVE :-
Study and analysis of DTFM coder-decoder

DTFM -

The telephone network is designed to carry voice signals. Nonetheless, it is frequently asked to carry other types of signals. A simple and common example is telephone numbers. It has to do that over circuits designed to carry voice signals.

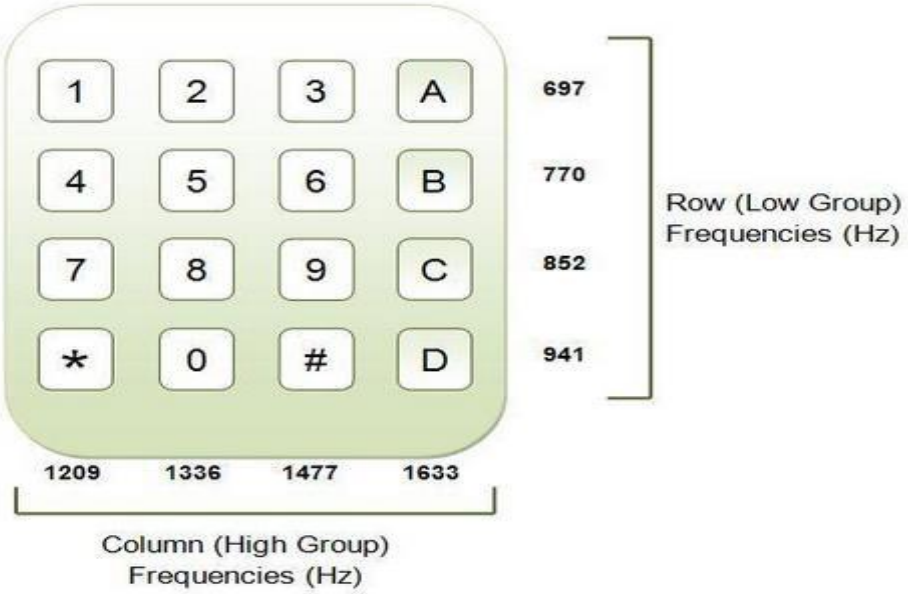
Therefore we need a system with following structure:



It converts sequences of numerical digits into signals that will easily traverse circuits designed for voice. The standard way to do that is DTMF Signaling. DTMF signaling or dual-tone, multi-frequency signaling. DTMF signaling converts decimal digits **{0,1,…,9}**, the symbols ***, #** and the alphabets **A,B,C,D** into sounds that share enough essential characteristics with voice to easily traverse circuits designed for voice.

The DTMF coder is thus a function that maps a number or some specific symbols into voice-like signal. The way the DTMF signaling works is that each digit is transformed into a pair of tones.

The tones are divided into two groups and each DTMF signals use one from each group. The tones are determined from the telephone keypads as follows:

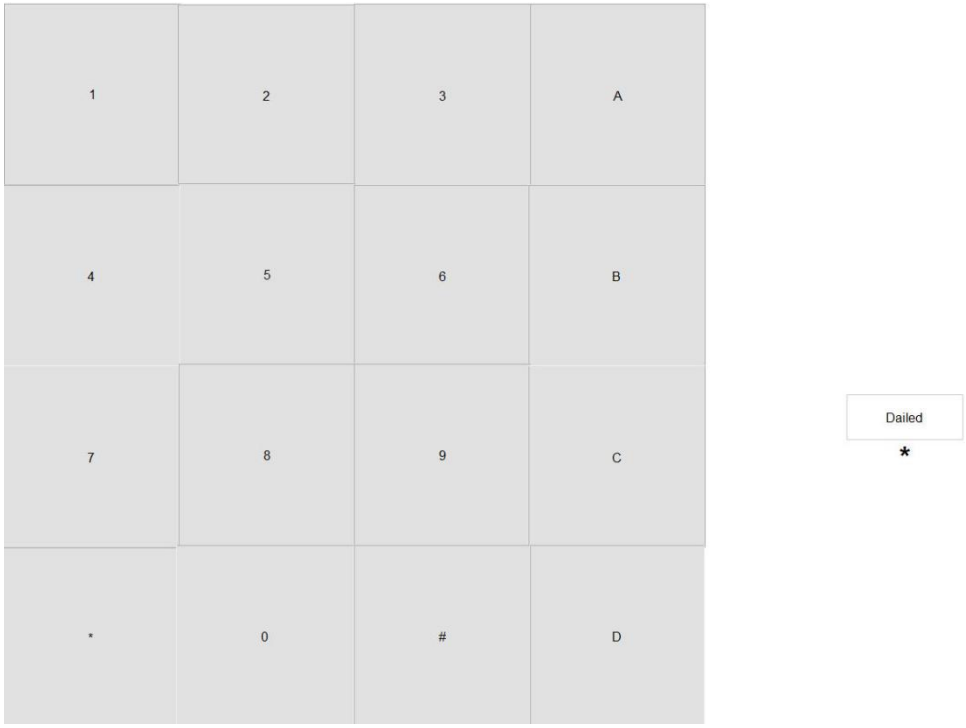


There is a frequency associated with each row and each column. Each key then specify two frequencies. When we press a key the telephone company knows what number we are dialing by observing the tones and they switch the call accordingly.

DTFM INTERFACE DESIGN

To achieve such an interface we used the below dial pad and gui code associated with it is shown below :-

The dialed box in the right shows the decoded output.




```
function varargout = gui_dial(varargin)
% GUI_DIAL MATLAB code for gui_dial.fig
%      GUI_DIAL, by itself, creates a new GUI_DIAL or raises the existing
%      singleton*.
%
%      H = GUI_DIAL returns the handle to a new GUI_DIAL or the handle to
%      the existing singleton*.
%
%      GUI_DIAL('CALLBACK',hObject,eventData,handles,...) calls the local
%      function named CALLBACK in GUI_DIAL.M with the given input arguments.
%
%      GUI_DIAL('Property','Value',...) creates a new GUI_DIAL or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before gui_dial_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to gui_dial_OpeningFcn via varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help gui_dial
```

```
% Last Modified by GUIDE v2.5 11-Feb-2020 14:54:47

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @gui_dial_OpeningFcn, ...
                   'gui_OutputFcn',  @gui_dial_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before gui_dial is made visible.
function gui_dial_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to gui_dial (see VARARGIN)

% Choose default command line output for gui_dial
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);


% UIWAIT makes gui_dial wait for user response (see UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.
function varargout = gui_dial_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

```
% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(1);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
%update_gui_bank


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(2);
dailed=subdecode(signal);
set(handles.out,'string',dailed);

% --- Executes on button press in pushbutton3.
function pushbutton3_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton3 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(3);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in A.
function A_Callback(hObject, eventdata, handles)
% hObject    handle to A (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad('A');
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(5);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(6);
dailed=subdecode(signal);
```

```
set(handles.out,'string',dailed);
% --- Executes on button press in B.
function B_Callback(hObject, eventdata, handles)
% hObject    handle to B (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad('B');
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(4);
dailed=subdecode(signal);
set(handles.out,'string',dailed);% --- Executes on button press in pushbutton7.
function pushbutton7_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(7);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(8);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in pushbutton9.
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(9);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in C.
function C_Callback(hObject, eventdata, handles)
% hObject    handle to C (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad('C');
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in Star.
function Star_Callback(hObject, eventdata, handles)
% hObject    handle to Star (see GCBO)
```

```
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad('*');
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in pushbutton0.
function pushbutton0_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton0 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(0);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in hash.
function hash_Callback(hObject, eventdata, handles)
% hObject    handle to hash (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad('#');
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in D.
function D_Callback(hObject, eventdata, handles)
% hObject    handle to D (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad('D');
dailed=subdecode(signal);
set(handles.out,'string',dailed);
% --- Executes on button press in Decode.
function edit4_Callback(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of edit4 as text
%        str2double(get(hObject,'String')) returns contents of edit4 as a double
% --- Executes during object creation, after setting all properties.
function edit4_CreateFcn(hObject, eventdata, handles)
% hObject    handle to edit4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

DTFM SIGNAL SYNTHESIS

To achieve the desired frequencies for each dial tone we used the code below :-
Observing a part of code for the button 5 used to make the gui,

```
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(5);
```

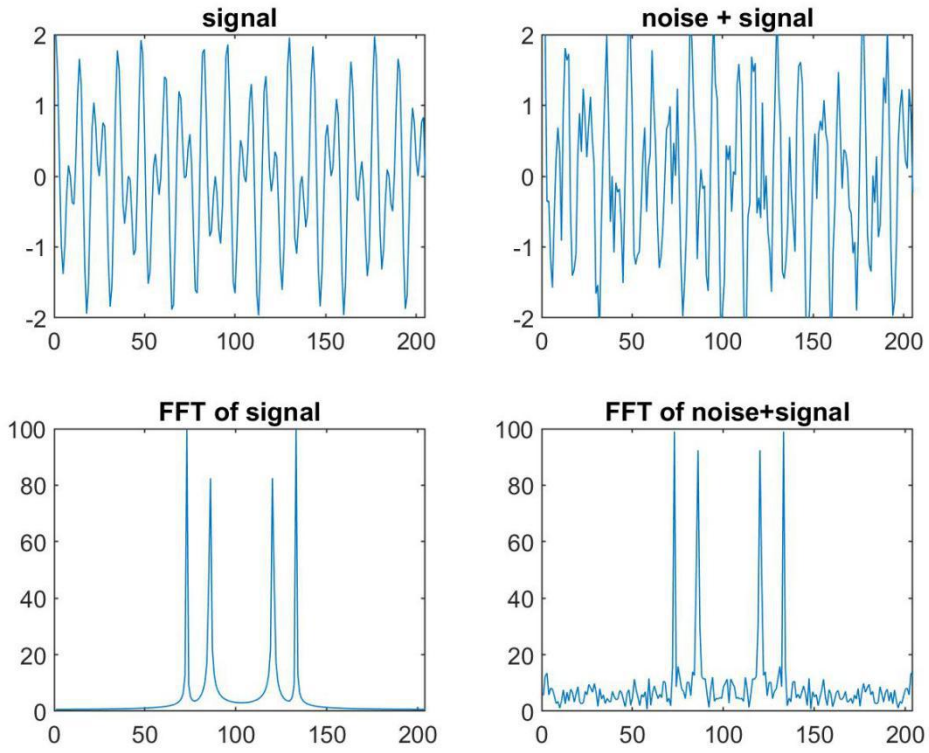
We see that we call the function phone_pad with 5 as input , which we can observe below that will produce a signal with frequencies 770Hz and 1336Hz which is exactly as given in question.

```
function [n,noise_signal]=phone_pad(k);

N=204;
n=0:N-1;
fs=8192;
F_c=[697;770;852;941]*ones(1,4);
F_c=F_c';F_c=F_c(:)';
F_r=[1209;1336;1477;1633]*ones(1,4);
F_r=F_r(:)';
F=2*pi/fs*[F_c;F_r];
switch k
case {1,2,3}
signal=cos(F(:,k)*n);
signal=sum(signal);
signal=[signal zeros(size(signal))];
case 'A'
signal=cos(F(:,4)*n);
signal=sum(signal);
signal=[signal zeros(size(signal))];
case {4,5,6}
signal=cos(F(:,k+1)*n);
signal=sum(signal);
signal=[signal zeros(size(signal))];
case {7,8,9}
signal=cos(F(:,k+2)*n);
signal=sum(signal);
signal=[signal zeros(size(signal))];

case 'B'
signal=cos(F(:,8)*n);
signal=sum(signal);
signal=[signal zeros(size(signal))];
case '*'
signal=cos(F(:,13)*n);
signal=sum(signal);
signal=[signal zeros(size(signal))];
```

```
case '#'
    signal=cos(F(:,15)*n);
    signal=sum(signal);
    signal=[signal zeros(size(signal))];
case 'D'
    signal=cos(F(:,16)*n);
    signal=sum(signal);
    signal=[signal zeros(size(signal))];
case 'C'
    signal=cos(F(:,12)*n);
    signal=sum(signal);
    signal=[signal zeros(size(signal))];
case 0
    signal=cos(F(:,14)*n);
    signal=sum(signal);
    signal=[signal zeros(size(signal))];

otherwise disp('Unknown digit')
end
signal=signal';

figure(2);
subplot(2,2,1)
plot(signal)
axis([0 205 -2 2]);
title(['signal']);
n=1:size(signal,1);
n1=204;
y1 = fft(signal,n1);         %taking fft of the signal
y1 = fftshift(y1);      %fft shift to get in -fs to +fs range
m1 = abs(y1);
subplot(2,2,3)
plot(m1)
%axis([0 205 -2 2]);
title(['FFT of signal']);
no=1.5*(rand(size(signal))-0.5);
noise_signal=signal+no;
subplot(2,2,2)
plot(noise_signal)
axis([0 205 -2 2]);
title(['noise + signal']);

y2 = fft(noise_signal,n1);         %taking fft of the signal
y2 = fftshift(y2);      %fft shift to get in -fs to +fs range
m2 = abs(y2);
subplot(2,2,4)
plot(m2)
%axis([0 205 -2 2]);
title(['FFT of noise+signal']);
disp(snr(noise_signal,noise_signal-signal));
```

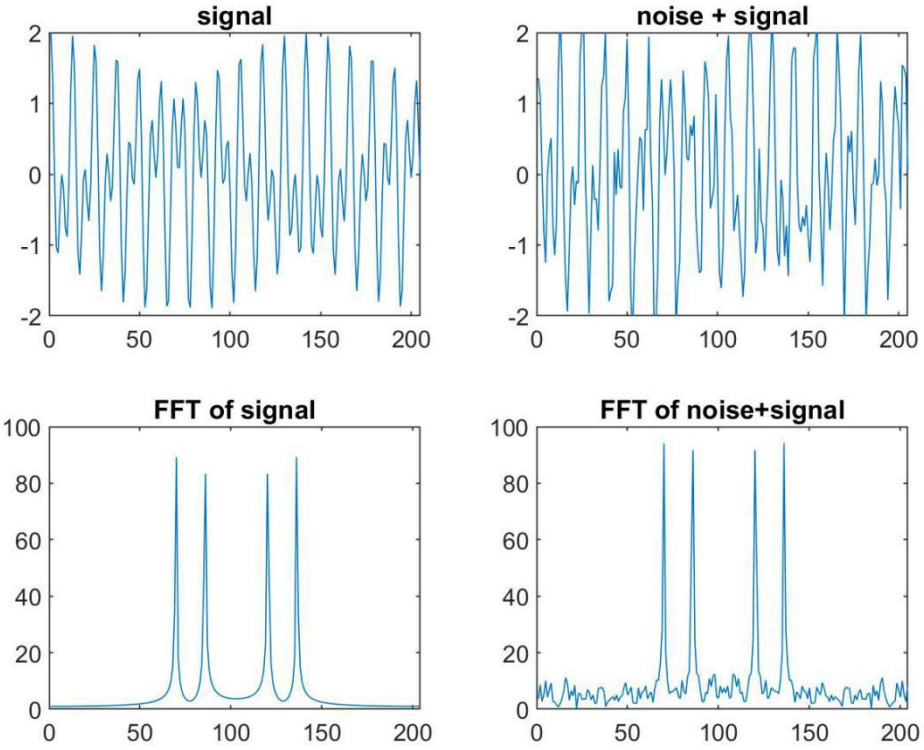
OUTPUT SIGNAL AND THEIR FFT TO EACH KEY PRESS WITHOUT AND WITH THE PRESENCE OF NOISE

1 is pressed

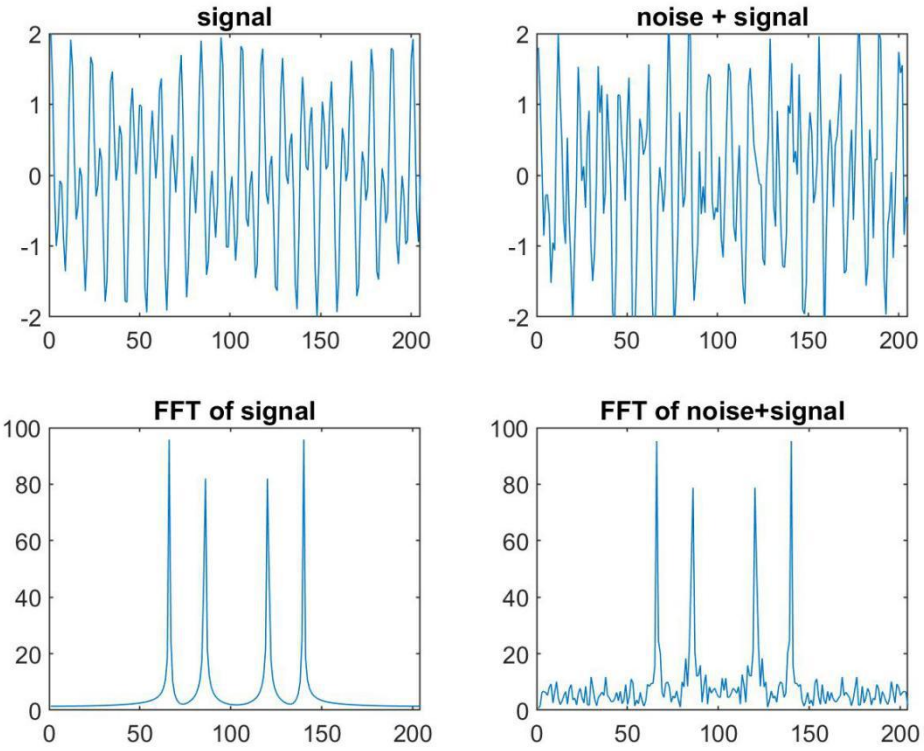


-We added noise to make it more robust since this is supposed to be implemented in a real life dial pad where noise might be added in the channel.
-We added noise that caused SNR to be about 5-6 dB.

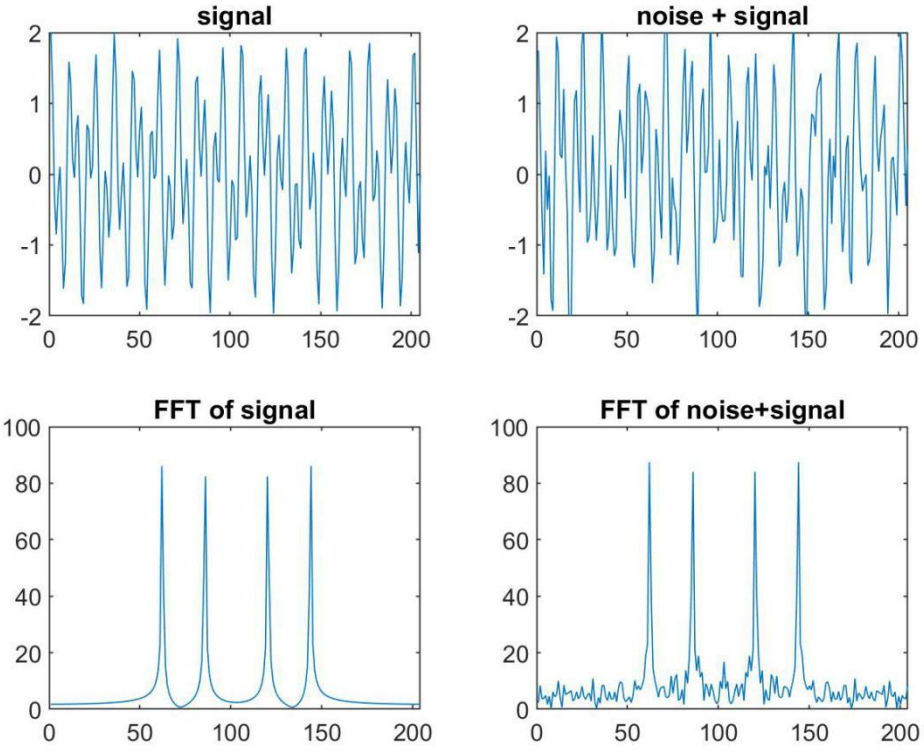
2 is pressed



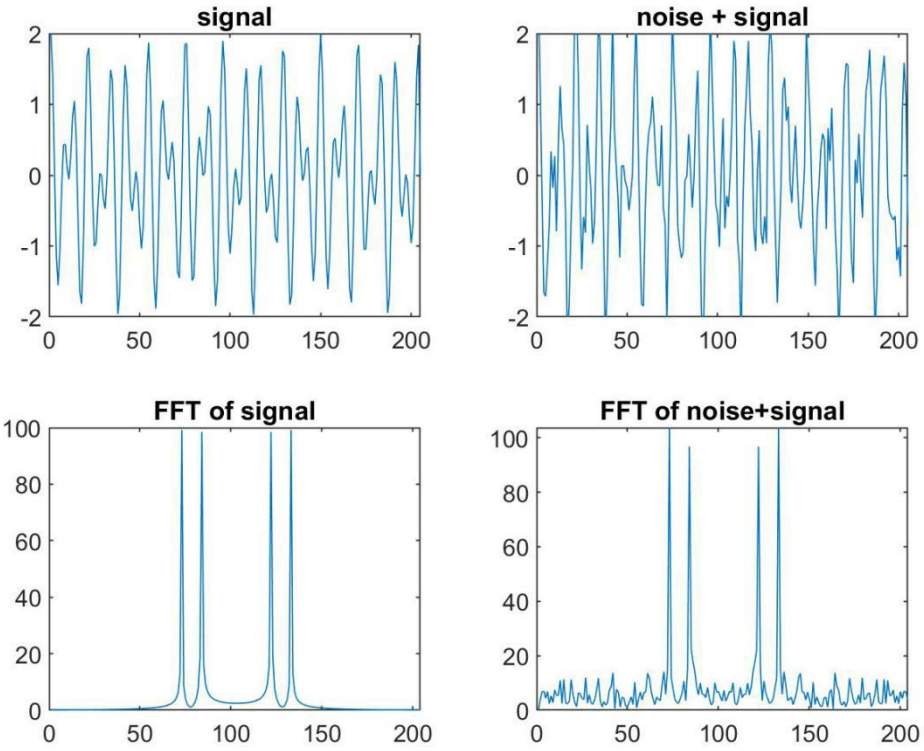
3 is pressed



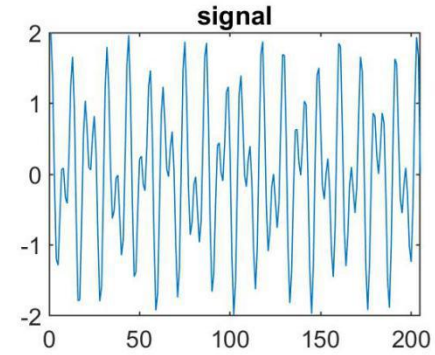
A is pressed

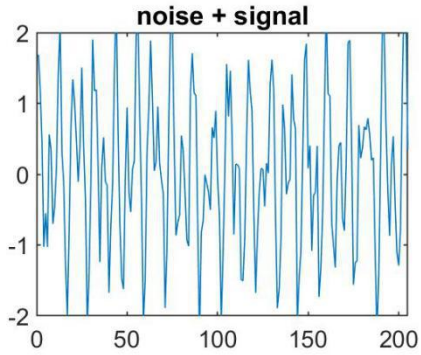


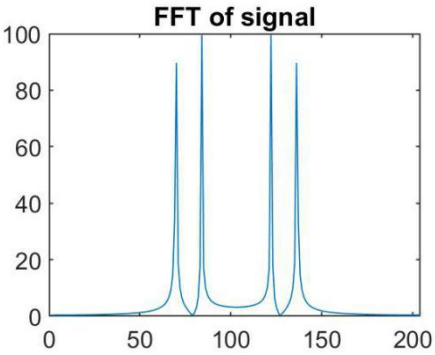
4 is pressed

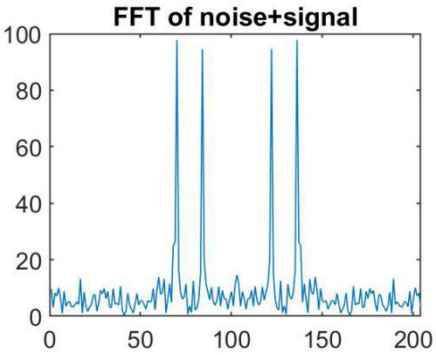


5 is pressed

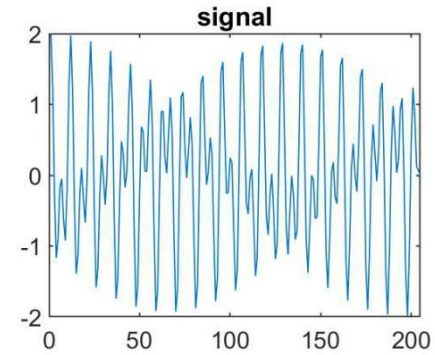


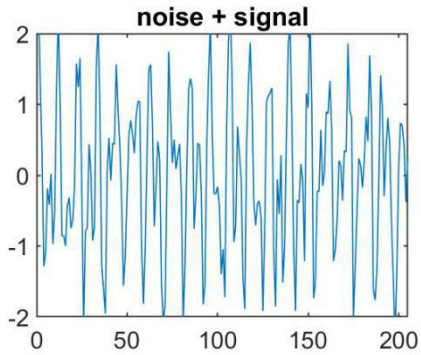


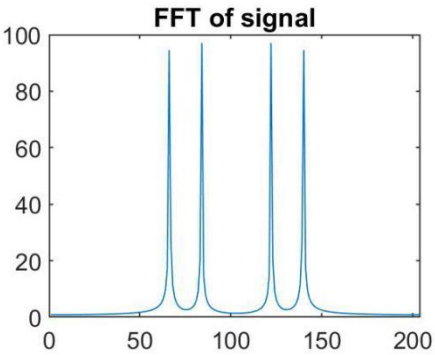


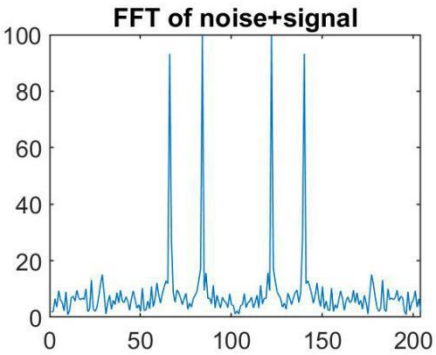


6 is pressed

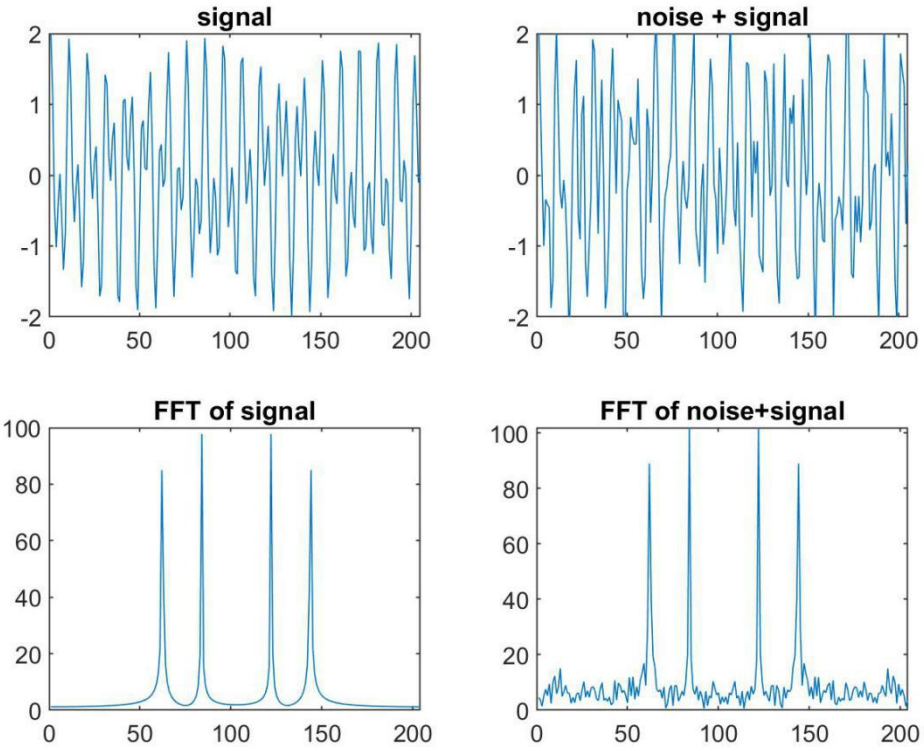




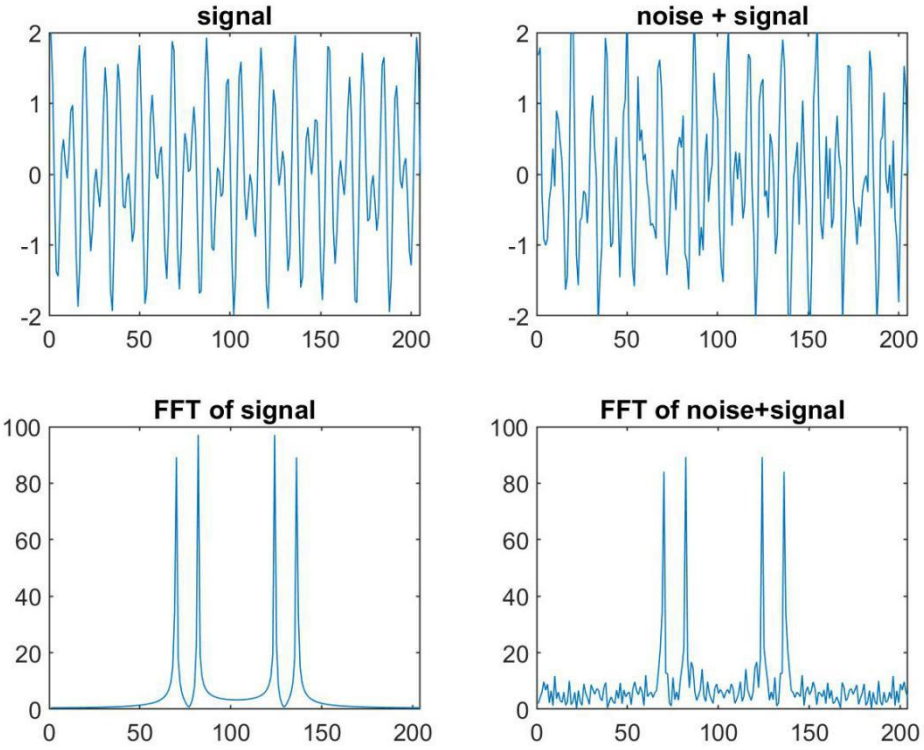




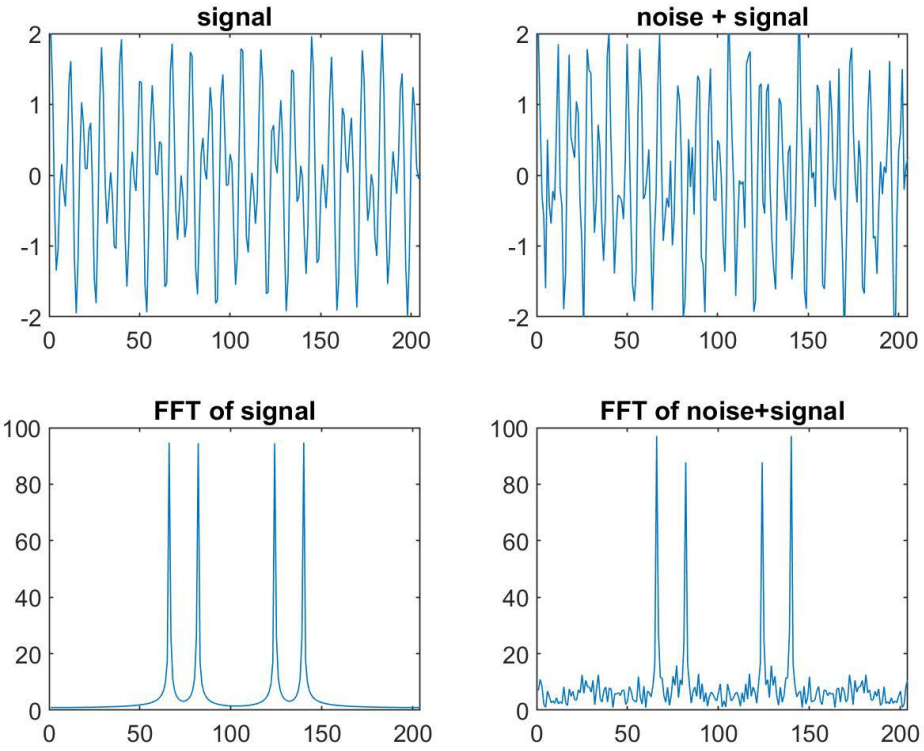
B is pressed



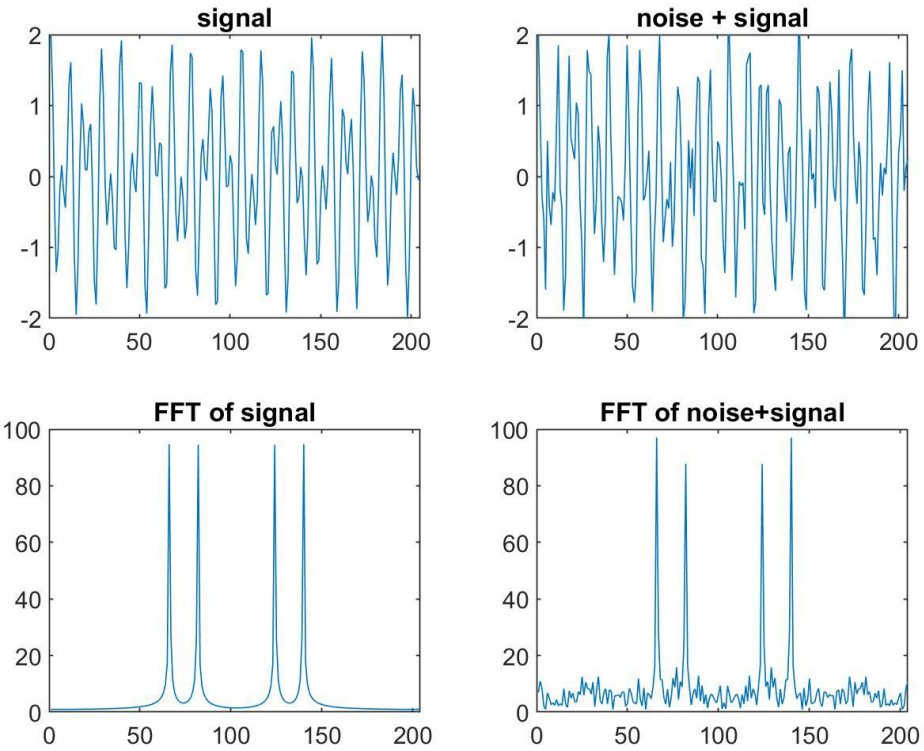
7 is pressed



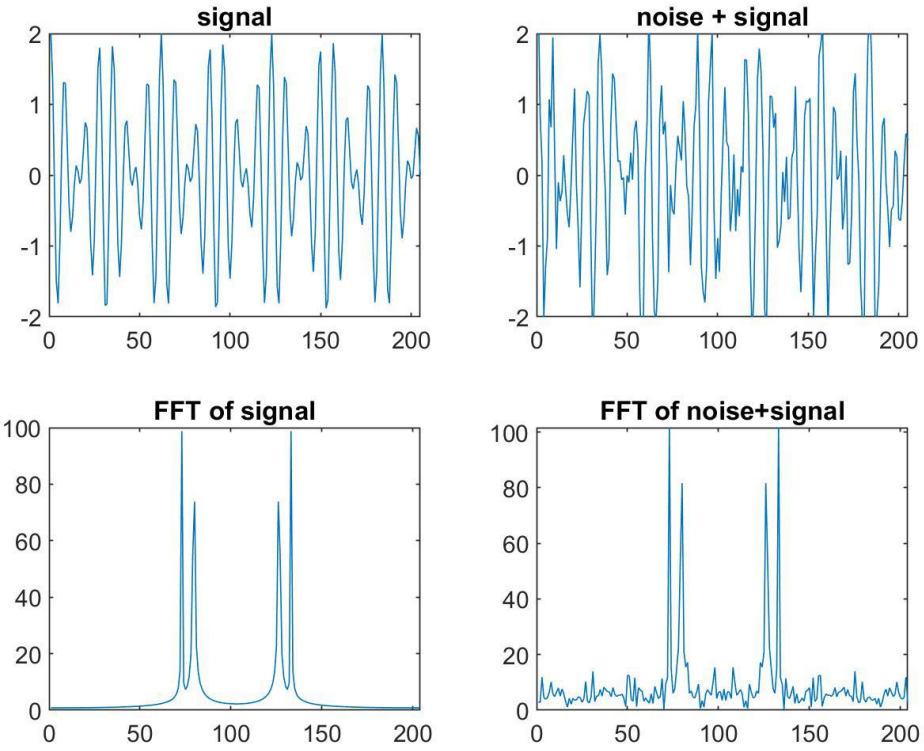
8 is pressed



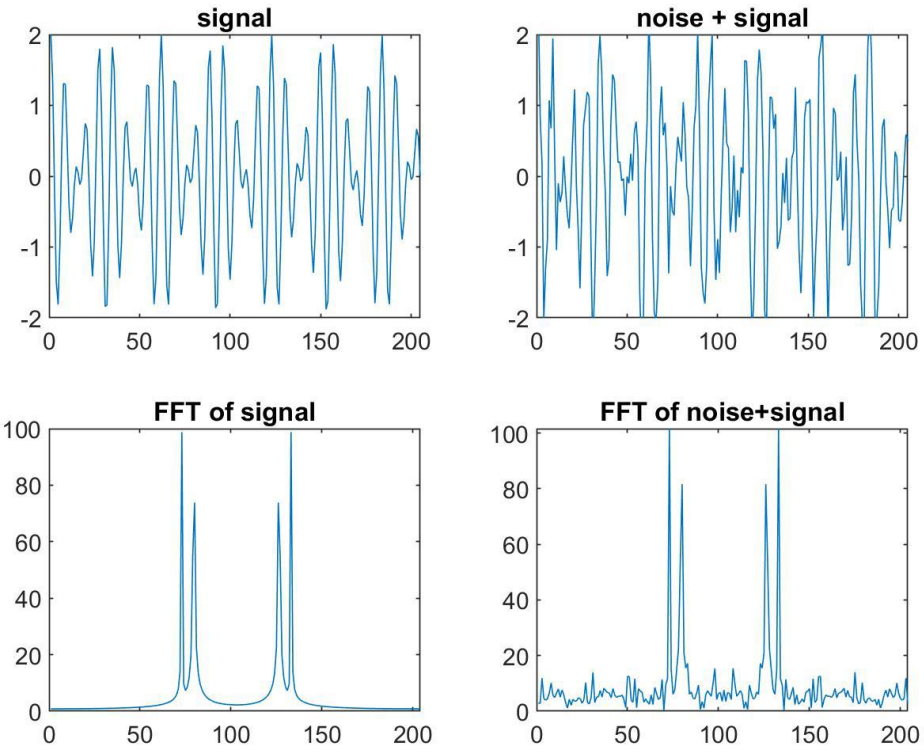
9 is pressed



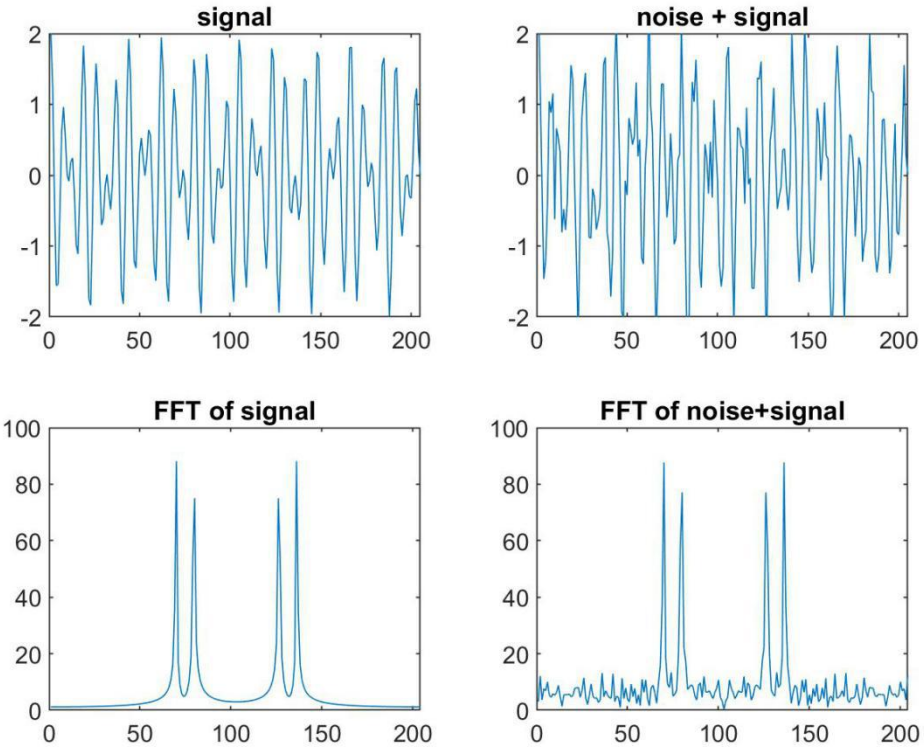
C is pressed



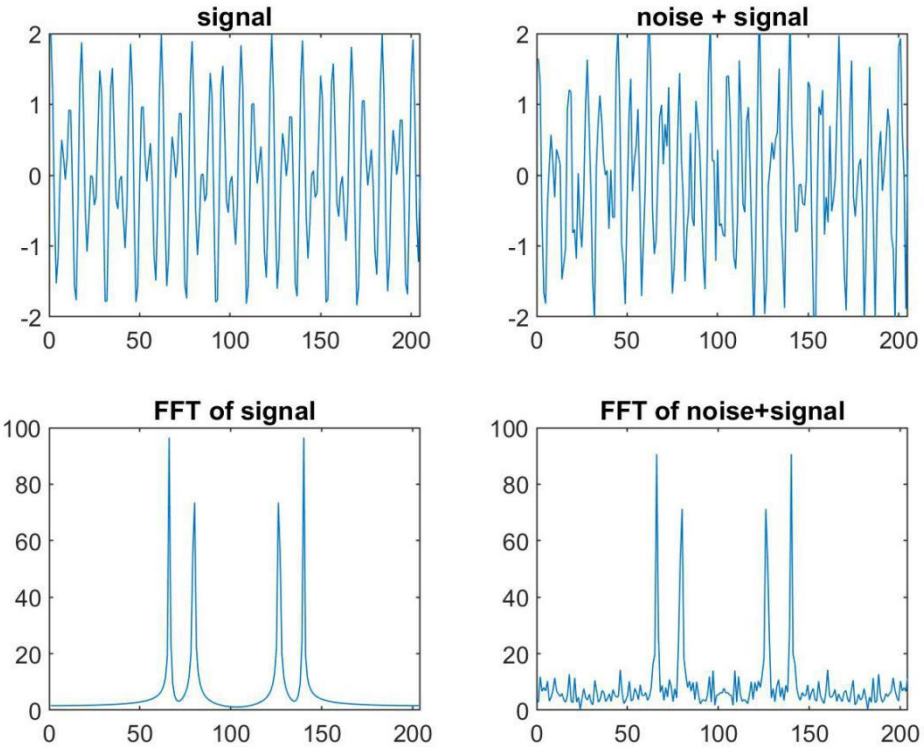
* is pressed



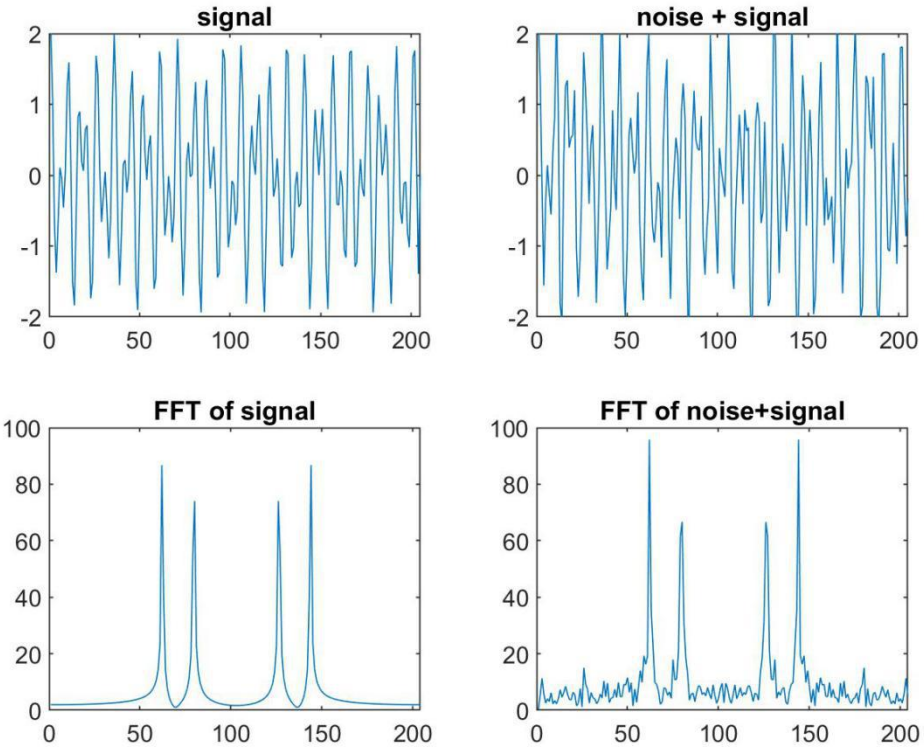
0 is pressed



is pressed



D is pressed



DTFM DECODER DESIGN

To decode the signal received that can also tolerate a bit of noise we call the subdecode function :-

Observing a part of code for the button 5 used to make the gui,

```
function pushbutton9_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
[n,signal]=phone_pad(9);
dailed=subdecode(signal);
set(handles.out,'string',dailed);
```

The signal generated before is now sent to subdecode and the output is displayed in dailed box of the gui.

SUBDECODE FUNCTION CODE:-

```
function dailed=subdecode(signal);

y=signal;
rmain=2048*2;rmag=1024*2;
cn=9;cr=0.5;
fs=8000;
cl=.25;ch=.28;
[b,a]=cheby1(cn,cr,cl);
yfilt1=filter(b,a,y);
h2=fft(yfilt1,rmain);
hmag2=abs(h2(1:rmag));
[b1,a1]=cheby1(cn,cr,ch,'high');
yfilt2=filter(b1,a1,y);
h3=fft(yfilt2,rmain);
hmag3=abs(h3(1:rmag));
hlow=fft(yfilt1,rmain);
hmaglow=abs(hlow);

hhigh=fft(yfilt2,rmain);
hmaghigh=abs(hhigh);

m=max(abs(hmag2));n=max(abs(hmag3));
o=find(m==hmag2);p=find(n==hmag3);
j=((o-1)*fs)/rmain;
k=((p-1)*fs)/rmain;
if j<=732.59 & k<=1270.91;
  %disp('---> Key Pressed is 1');
  dailed='1';
  elseif j<=732.59 & k<=1404.73;
   % disp('---> Key Pressed is 2');
    dailed='2';
  elseif j<=732.59 & k<=1553.04;
```

```
    % disp('---> Key Pressed is 3');
     dailed='3';
   elseif j<=732.59 & k>1553.05;
     % disp('---> Key Pressed is A');
    dailed='A';
elseif j<=809.96 & k<=1270.91;
     % disp('---> Key Pressed is 4');
dailed='4';
elseif j<=809.96 & k<=1404.73;
     % disp('---> Key Pressed is 5');
dailed='5';
elseif j<=809.96 & k<=1553.04;
     % disp('---> Key Pressed is 6');
     dailed='6';
   elseif j<=809.96 & k>1553.05;
     % disp('---> Key Pressed is B');
     dailed='B';
   elseif j<=895.39 & k<=1270.91;
     % disp('---> Key Pressed is 7');
dailed='7';
elseif j<=895.39 & k<=1404.73;
     % disp('---> Key Pressed is 8');
dailed='8';
elseif j<=895.39 & k<=1553.04;
     % disp('---> Key Pressed is 9');
     dailed='9';
   elseif j<=895.39 & k>1553.05;
     % disp('---> Key Pressed is C');
     dailed='C';
   elseif j>895.40 & k<=1270.91;
     % disp('---> Key Pressed is *');
     dailed='*';
   elseif j>895.40 & k<=1404.73;
     % disp('---> Key Pressed is 0');
     dailed='0';
   elseif j>895.40 & k<=1553.04;
    %  disp('---> Key Pressed is #');
    dailed='#';
   elseif j>895.40 & k>1553.05;
     % disp('---> Key Pressed is D');
    dailed='D';
end
```

We take the peaks of the ffts we have observed in the synthesis and differentiate each input using the if-else statements above.

DISCUSSION and OBSERVATIONS :-

1. A DTMF tone representing a single key selected consists of two summed frequencies that have been chosen such that no harmonics occur.

2. The sampling frequency is taken to be much higher than the maximum frequency to avoid aliasing.

3. No frequency is an integer multiple of one another so that no overlap occurs at receiver side.

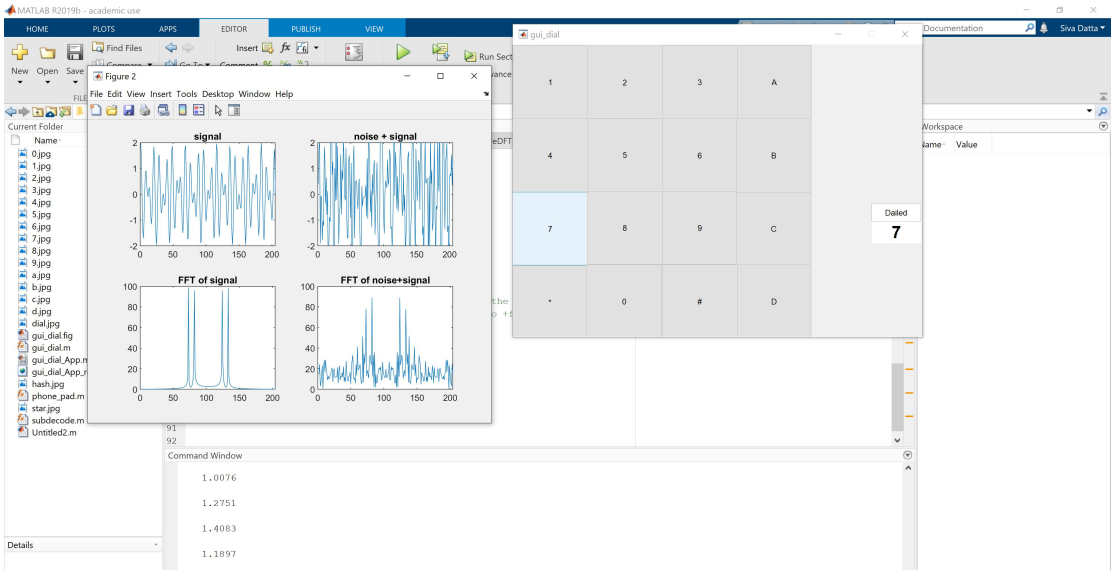
4. Difference or sum of any two frequencies does not equal any of the other defined frequencies.

5. We separated them based on the increasing order of frequencies using the if else statements.

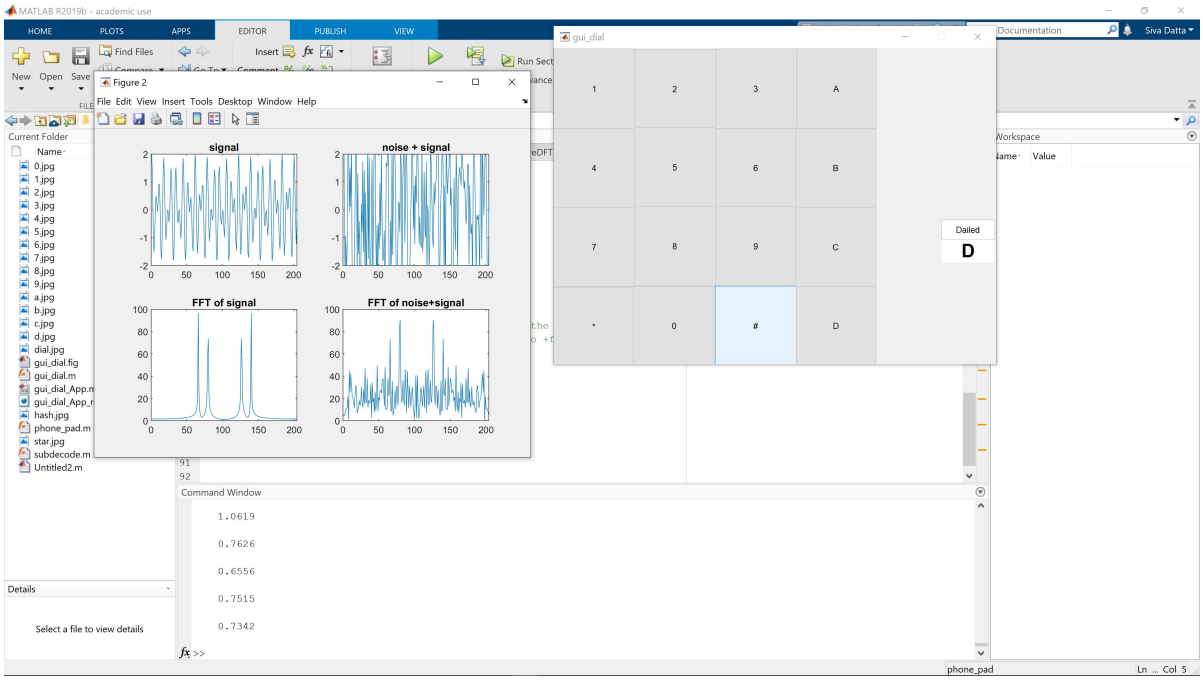
6. Though the examples shown above had an SNR of 5-6 dB , running our model through various possible noise channels has proven that it can withstand noise even through ranges of 1dB.

7. Below we see DTFM decoder working just fine around an SNR of 1dB.

The decoded output is shown in the dailed box on the right and The key pressed is highlighted in blue and we see that though we had large noise causing SNR=1.1897dB, the DTFM decoder managed to detect 7 every time we sent through a different random noise around the same SNR.



8. But as we went below 1dB SNR , we got wrong results as shown below, again the key highlighted in blue is the key pressed, and decoded output is shown in the box at the right.



Here SNR=0.7342dB