# SNS lab 11 part 1 and 2

Name:
Basil khowaja and Aqeel Mehdi

## Task 1:

```
ftable = [1;2;3;4;5]*[80,110]

fs = 8000; xx = [ ];

keys = rem(3:12,10) + 1

for ii = 1:length(keys)

kk = keys(ii);

xx = [xx,zeros(1,400)]; % adds silence

krow = ceil(kk/2);

kcol = rem(kk-1,2) + 1;

dur=0.15;

t=0:1/fs:dur;

xx = [xx, cos(2*pi*ftable(krow,kcol)*t)]; % adss the relevant tone

end

xx

soundsc(xx,fs)
```

a) The program takes the key one by one sequentially, and then generates a tone corresponding to the key by finding its row and column and passing it to a cosine function, and then finally concatentaing the tone of the key to the final signal xx. Before concateniting each key's tone, it also concatenates an array of 400 zeros which corresponds to 400/8000 = 0.05s duration of pause (silence).

b) The size of the table is 5x2. The player order of frequency is the order in which keys are given (equivalent to being pressed) which is given in keys vector (4 5 6 7 8 9 10 1 2 3)

c) SInce there is a 0.05 s delay and each tone plays for 0.2 seconds then each tone plays for 0.05 + 0.15 = 0.2

for 10 tones we have, t = (0.2)*10 = 2.0s total duration

No of samples = t * sampling_freq = 2.0 * 8000 = 16,000

**Output:**

```
ftable = 5×2
      80   110
     160   220
     240   330
     320   440
     400   550

keys = 1×10
       4     5     6     7     8     9    10     1     2     3
```

```
xx = 1×16010
       0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0     0 ...
```

## Task 2:

```
% Task 2

fs = 8000;

xx = dtmfdial(['A','3'], 8000);

soundsc(xx,8000)
```

## Output:

```
dtmf = struct with fields:
        keys: [4×4 char]
    colTones: [4×4 double]
dtmf = struct with fields:
        keys: [4×4 char]
    colTones: [4×4 double]
    rowTones: [4×4 double]
ii = 1
jj = 4
num_zeros = 400
ii = 1
jj = 3
num_zeros = 400
```

## Observation:

In this code we are actually configuring a sampling frequency of 8000 Hz and then generating a DTMF signa which is also known as dual tone multi frequency for the characters 'A' and '3' using the matlab dtmfdial() function then we play back the generated DTMF signal using the soundsc() function with the same sampling frequency of 8000 Hz. Here each key on a telephone keypad shows a unique frequency pair.

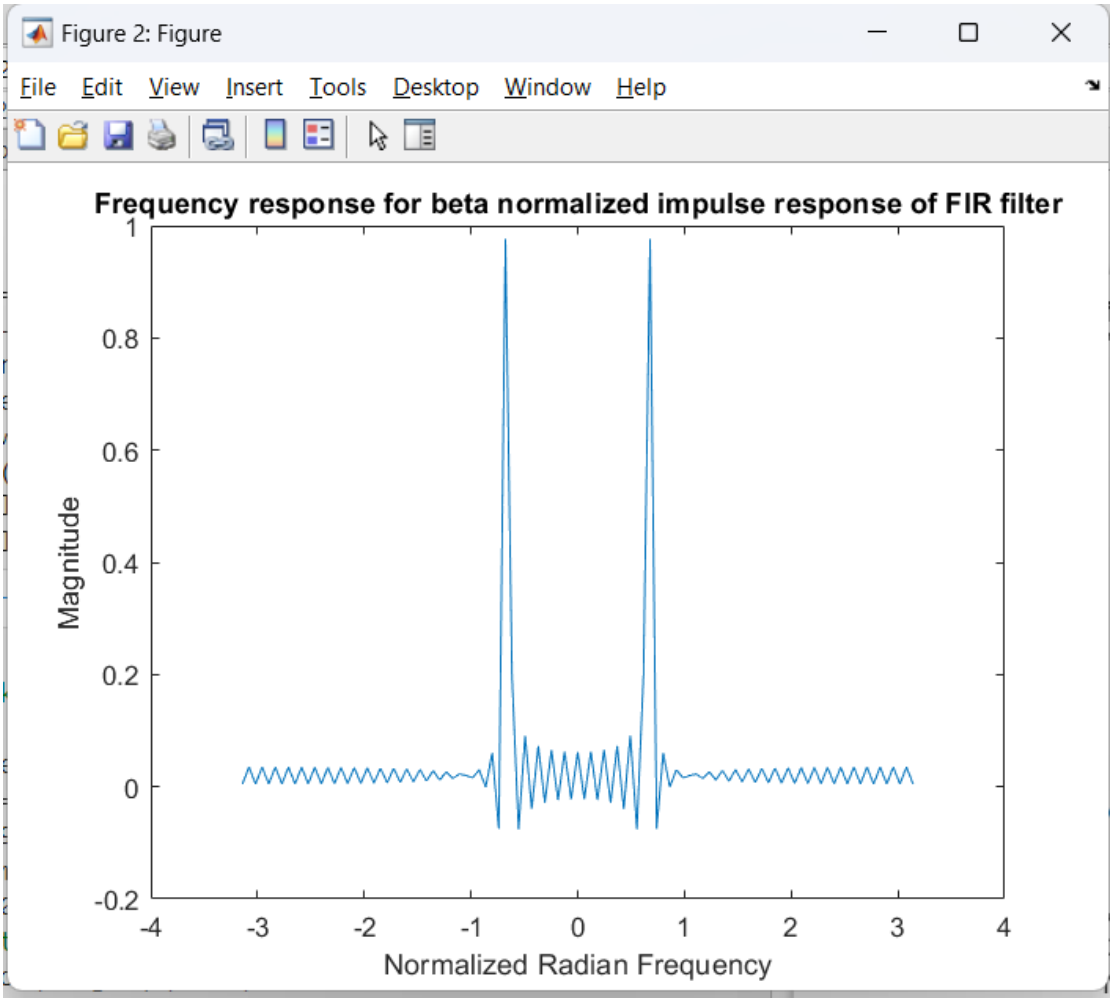
## Task 3:

```
% Task 3
fb = 852;
L = 51;
beta = maxbeta(L, fb, fs)
n = 0:L-1;
form = cos(2*pi*fb*n/fs);

% the value of beta is around 0.04.

form = beta*form;
ww = -pi:(pi/L):pi;
k = freqz(form, 1, ww);
figure;
plot(ww,k)
title('Frequency response for beta normalized impulse response of FIR filter')
xlabel("Normalized Radian Frequency");
ylabel("Magnitude");
```

**Output:**




**Explanation:**

We are making a Finite Impulse Response (FIR) filter in this code which has a cutoff frequency of 852 Hz and 51 samples in length. firstly we use the maxbeta() method to find the right scaling factor beta Then we make the impulse response of the filter by changing a cosine function with the cutoff frequency we want then we scale the impulse reaction with beta to make it normal then we use the freqz() tool to find the filter's frequency response and plot it against the normalized radian frequency axis.

## Task4:

```
count = 0;
L=[40 80 120 160 180 200];
figure;
sgtitle("Frequency Response for different Ls");
for i = L
   fb=[697 770 852 941 1209 1336 1477 1633];
   hh=dtmfdesign(fb,i,8000);

   ww = 0:(pi/i):pi;
   count = count + 1;

   subplot(length(L),1,count);
   title("Frequency Response for L = " +i);
   xlabel("Normalized Radian Frequency");
   ylabel("Amplitude");
   hold on;

   for j=1:8
       hn=hh(:,j);
       plot(ww,abs(hn))
   end
   legend('697 Hz' ,'770 Hz' ,'852 Hz' ,'941 Hz', '1209 Hz', '1336 Hz', '1477
Hz', '1633 Hz')
end
hold off;
figure;
title("Frequency Response for L = " +i);
xlabel("Normalized Radian Frequency");
```

```
ylabel("Amplitude");

hold on;
```

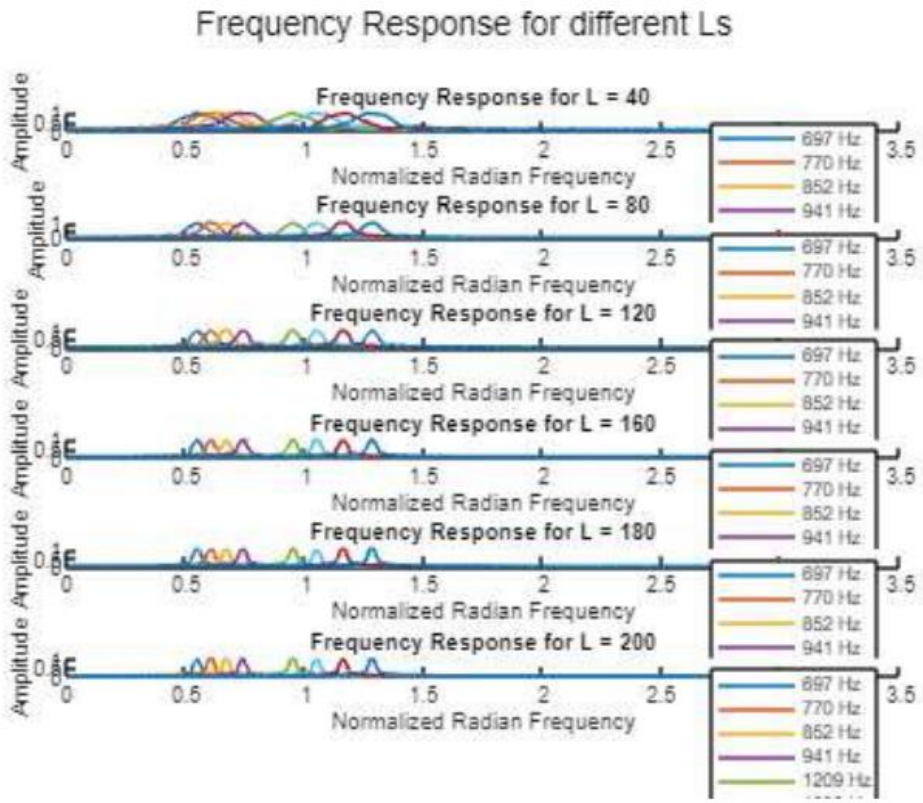
```
for j=1:8

   hn=hh(:,j);

   plot(ww,abs(hn))

end

legend('697 Hz' ,'770 Hz' ,'852 Hz' ,'941 Hz', '1209 Hz', '1336 Hz', '1477 Hz',
'1633 Hz')
```



## Explanation:

In this code the frequency response of Finite Impulse Response (FIR) filters made for DTMF signals is looked at for various filter lengths (L). It shows how changes in filter length affect the attenuation and passband features for different DTMF frequencies by plotting the response for each length separately. Then we put all the results in one plot for the last filter length that was given. We can observe that when we increase the value of L, the width of the passband decreases and we get a sharper peak. In the frequency response for different Ls we can see that we are not getting only one frequency but a bunch of frequencies in the pass band but the other frequencies are attenuated and we can easily determine which frquency is passed and which is not.

## Task 5, 6, 7:

```
fs = 8000;

L=200;

test =
['A','B','C','D','*','#','0','1','2','3','4','5','6','7','8','9'];

xx = dtmfdial(test , fs);

soundsc(xx, fs);
```

```
keys = dtmfrun(xx, L, fs)
```

**Task7:**

```
Number= ['4','0','7','*','8','9','1','3','2','B','A', 'D','C']
```

```
fs=8000;
```

```
xx = dtmfdial(Number, fs );

soundsc(xx, fs);

L = 200;

keys = dtmfrun(xx, L, fs)
```

Firstly we make DTMF signals for a sequence of test characters, such as letters, numbers, and special characters, then play them back then we analyze these signals to find the corresponding keys pressed by using the dtmfrun() function with a specific FIR filter length L set to 200 samples. then we repeat this process for a sequence representing a phone number. We can observe from output that If the magnitude of H(e^jω) exceeds 1, the filter can amplify the input signal, causing

distortion and loss of information. Conversely, if the magnitude is less than 1, the filter may attenuate the input signal, resulting in a lower amplitude output with potential loss of important information.

## Function:
## (Dtmfdial):

```
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);


xx = [];
for keynum = keyNames
    dur = 0.2;
    [ii, jj] = find(keynum == dtmf.keys); % find corresponding row and column
    t = 0:1/fs:dur;
    num_zeros = fs * 0.05; % to introduce 0.05 seconds of silence
    Row = dtmf.colTones(ii, jj); % determine the frequency from row and column
    Col = dtmf.rowTones(ii, jj);
    xx = [xx, zeros(1, num_zeros)]; % introduce silence
    xx = [xx, cos(2*pi*Row*t) + cos(2*pi*Col*t)]; % include the corresponding
tone
end


end
```

**Dtmfrun():**

```
function keys = dtmfrun(xx, L, fs)
% DTMFRUN keys = dtmfrun(xx, L, fs)
% returns the list of key names found in xx.
% keys = array of characters, i.e., the decoded key names
% xx = DTMF waveform
% L = filter length
```

```
% fs = sampling freq
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);


hh = dtmfdesign(dtmf.colTones(1,:), L, fs);
[nstart, nstop] = dtmfcut(xx, fs);


keys = []; % Initialize keys


for kk = 1:length(nstart)
    x_seg = xx(nstart(kk):nstop(kk)); % Extract one DTMF tone
    freq_arr = [];

    for freq = 1:length(dtmf.colTones)
        hn = hh(:, freq);
        sc = dtmfscore(x_seg, hn);
        if sc == 1
            freq_arr = [freq_arr , freq];
        end
    end


    if length(freq_arr) == 2
        if freq_arr(1) ~= freq_arr(2) && freq_arr(2) > 4
            key = dtmf.keys(freq_arr(1), freq_arr(2) - 4);
            keys = [keys, key];
        end
    end
end
```

**Task5(Dtmfscore()):**

```
function score = dtmfscore(xx, hh)
% 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(abs(y[n])) is greater than or equal to 0.59
% sc = 0 if max(abs(y[n])) is less than 0.59
%
xx = xx * (2/max(abs(xx))); % Scale the input x[n] to the range [-2, +2]
xc = conv(xx, hh);
max_val = max(abs(xc));


if max_val >= 0.59
    score = 1;
else
    score = 0;
end
end
```

## Dtmfdesign():

```
function hh = dtmfdesign(fb, L, fs)
% DTMFDESIGN
% 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 freq
%
% Each BPF must be scaled so that its frequency response has a
% maximum magnitude equal to one.

hh = [];

for k = 1:length(fb)
        freq = fb(k);
        hn = cos(2*pi*freq*(0:L-1)/fs);
        norm = maxbeta(L, freq, fs);
        hn = hn * norm;
        hh = [hh; hn];
end

end
```

## MaxBeta():

```
function norm = maxbeta(L, fb, fs)
% MAXBETA


n = 0:L-1;
```

```
hn = cos(2*pi*fb*n/fs);
ww = 0:(pi/L):pi;
freq_response = freqz(hn, 1, ww);
maxi = max(abs(freq_response));
norm = 1/maxi;
end
```