Mark:	/ 60
TA:	

University of Auckland

Laboratory Instructions and Work Sheet

2024

Course Number(s):	**MECHENG705**
Laboratory Number	**1**
Laboratory Title:	**Digital filter design**

Family name: Koh

Given name: Andreea

Student ID: 59569338

Group number: Stream 3 Group 1

1.1 Introduction

Digital signal processing (DSP) has been widely used in various applications. One of the key roles of digital signal processing is *filtering* for extracting particular information from a given signal (i.e. signal detection), which is often contaminated by noise. In order to effectively achieve desired performance, digital filters should be properly designed to meet their specifications. In this lab, we learn a basic design method of FIR filters, namely the window-based method, to extract the key codes from telephone touch-tone signals.

The goal of this lab is to learn how to design FIR filters in MATLAB, and implement the designed FIR filters to real world applications. The **objectives** of this lab are:

- To learn how to use MATLAB for digital signal processing.
- To learn how to design a simple FIR filter.
- To learn the role of FIR filters for signal detection in real world applications.

1.2 Assessment

Answer all questions given in this lab sheet by using your results obtained in the lab sessions. Submit **this lab work sheet** (i.e. Don't write up your own report! It will not be marked.) along with MATLAB plots and/or printouts as required by the announced due date/time. You can attach extra pages if each answer box does not have enough space, however make sure you indicate which question the answers on the extra pages are discussing. **Attach copies of all your Matlab codes** at the end of this lab sheet upon submission.

1.2.1 Marking rubrics

In total this lab assignment carries 60 marks. Tasks allocated marks are clearly stated with the amount of the marks they carry by boldface font. Your answer will be assessed based on various factors including but not limited to:

- Completeness - if you have completed the required tasks
- Correctness - if you demonstrate that you have gained correct understanding of the relevant concepts
- Accuracy - if your answer is accurate

Editorial errors such as missing labels on graph axis may also affect the marks you receive.

1.3 Acknowledgement

This laboratory is designed based on the Lab # 9 included in: McClellan, Schafer, and Yoder, *Signal Processing First*, Prentice Hall, 2003. Some of the MATLAB commands and GUIs are developed from the material provided by this textbook. Various lab problems are available in the CD-ROM of the textbook for students who are interested in learning other topics in signal processing.

1.4 MATLAB Setup

Before you start this lab, you need to do some settings with MATLAB by following steps.

1. Download `ME705_LabSet2024.zip` from Canvas and expand the data to any directory.
2. On MATLAB, set `Current Directory` as the directory where you expanded the files stored in `ME705_LabSet2024.zip`. You will not change the `Current Directory` throughout this lab.

1.4.1 Useful Matlab commands

There are various built in commands in MATLAB which are useful to implement digital signal processing algorithms. The following commands may be useful to work on this lab. For more details refer to the MATLAB online help or type `help xxx` on the command window where `xxx` is a command name.

freqz

`freqz` is a function that returns the frequency response of a digital filter. The basic syntax is

```
[H, W] = freqz(b, a, N),
```

which returns the `N`-point frequency response vector, `H`, and the corresponding normalised angular frequency vector, `W`, for the digital filter with numerator and denominator polynomial filter coefficients stored in `b` and `a`, respectively. When the filter is an FIR filter, `a` should always take 1 since there is no denominator in its transfer function. For the frequency resolution `N`, we use a fixed value 4096 in this lab. There are several other useful syntaxes which can be found in Matlab help. For example providing the sampling frequency of the system as the fourth input argument `Fs`, i.e. `[H, W] = freqz(b, a, N, Fs)`, would change the unit of `W` from the normalised angular frequency to the analogue frequency in Hz.

spectrogram

`spectrogram` is a function that returns a spectrogram calculated using short-time Fourier transform (STFT)[1]. Basic syntax to draw a spectrogram is

```
spectrogram(x,window,noverlap,nfft),
```

where `x` is the input signal `window` and `noverlap` are a vector of window function and the overlapped samples between adjacent frames, respectively. `nfft` is the number of DFT points which should be the same or longer than the length of `window`.

find

`find(x)` returns a vector containing the indices of each non-zero element in a vector `x`.

1.5 Overview of FIR Filtering

We will define an FIR filter as a discrete-time system that converts an input signal $x[n]$ into an output signal $y[n]$ by means of the weighted summation:

$$y[n] = \sum_{k=0}^{M} b_k x[n-k]. \tag{1.1}$$

Equation (1.1) gives a rule for computing the n-th value of the output sequence from certain values of the input sequence. The filter coefficients $\{b_k\}$ are constants that define the filter's behaviour. As an example, consider the system for which the output values are given by

$$\begin{aligned} y[n] &= \frac{1}{3}x[n] + \frac{1}{3}x[n-1] + \frac{1}{3}x[n-2] \\ &= \frac{1}{3}\left(x[n] + x[n-1] + x[n-2]\right). \end{aligned} \tag{1.2}$$

This equation states that the n-th value of the output sequence is the average of the n-th value of the input sequence $x[n]$ and the two preceding values, $x[n-1]$ and $x[n-2]$. For this example the b_k's are $b_0 = \frac{1}{3}$,$b_1 = \frac{1}{3}$, and $b_2 = \frac{1}{3}$.

MATLAB has a built-in function, `filter( )`, for implementing the operation in (1.1), but we can also use another command `conv( )` for the case of FIR filtering since FIR filter implementation is equivalent to the process of convolution sum. The function `filter` implements a wider class of filters than just the FIR. In the rest of this lab, we use only `conv`. The following MATLAB statements implement the three-point averaging system of (1.2):

```
nn = 0:99;                <- Time indices
xx = cos(2*pi*0.04*nn);   <- Input Signal
bb = [1/3 1/3 1/3];       <- Filter coefficients
yy = conv(bb, xx);        <- Compute the output
```

[1] See MECHENG 370 lecture slides available on Canvas for details of STFT.

In this case, the input signal `xx` is a vector containing a cosine function whose **normalised angular** frequency is 0.08π. In general, the vector `bb` contains the filter coefficients $\{b_k\}$ needed in (1.1). These are loaded into the `bb` vector in the following way:

$$\texttt{bb} = [\texttt{b0, b1, b2,} \cdots \texttt{,bM}];$$

In MATLAB, all sequences have finite length because they are stored in vectors. If the input signal has, for example, L samples, we would normally only store the L samples in a vector, and would assume that $x[n] = 0$ for n outside the interval of L samples; i.e. we do not have to store any zero samples unless it suits our purposes.

If we process a finite-length signal through (1.1), then the output sequence $y[n]$ will be longer than $x[n]$ by M samples. Whenever `conv( )` implements (1.1), we will find that

```
length(yy) = length(xx)+length(bb)-1
```

In this lab, you will use `conv( )` to implement FIR filters and begin to understand how the filter coefficients define a digital filtering algorithm. In addition, this lab will introduce examples to show how a filter reacts to different frequency components of the input signal.

1.6 Telephone Touch Tone Dialling

As a real world application of digital filters, in this lab we look at signals used for dialling a telephone. Dual-tone multi-frequency (DTMF) signalling is a telecommunication signalling system using the voice-frequency band over telephone lines between telephone equipment and other communications devices and switching centres. DTMF is standardised by ITU-T Recommendation Q.23.

Telephone keypads are used to generate DTMF signals to make a call. When any key is pressed, the sinusoids of the corresponding row and column frequencies in Table 1.1[2] are generated and summed, hence it is named "dual tone". As an example, pressing the "5" key generates a signal containing the sum of the two tones at 770 Hz and 1336 Hz together. The frequencies in Table 1.1 were chosen to avoid harmonics to occur by the distortion caused by transmission channels. No frequency is an integer multiple of another, the difference between any two frequencies does not equal any of the frequencies, and the sum of any two frequencies does not equal any of the frequencies. This makes it easier to detect exactly which tones are present in the dialled signal in the presence of line noise and non-linear distortions.

On the receiver side there are several steps to decode a DTMF signal:

1. Divide the received signal into short time segments representing individual key presses.

2. Filter the individual segments to extract the possible frequency components. **Bandpass filters** can be used to isolate the sinusoidal components.

[2]Normally we do not see the keys "A" to "D" on a telephone device.

Table 1.1: Extended DTMF encoding table for Touch Tone dialing. When any key is pressed the tones of the corresponding column and row are generated and summed. Keys A-D (in the fourth column) are not implemented on commercial and household telephone sets, but are used in some military and other signalling applications.

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

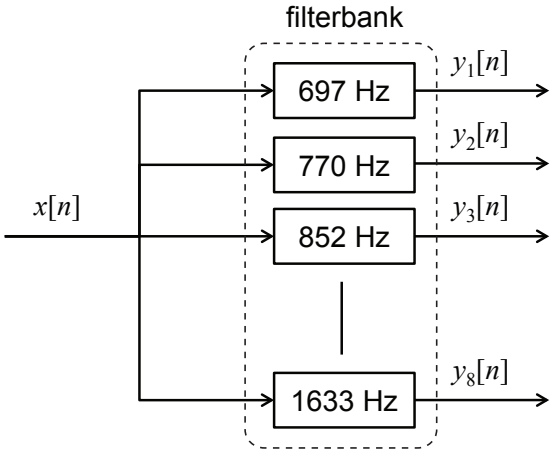


Figure 1.1: Filter bank consisting of PFs which pass frequencies corresponding to the eight DTMF component frequencies listed in Table 1.1. The number is each box is the center frequency of the BPF.

3. Determine which two frequency components are present in each time segment by measuring the size of the output signal from all of the bandpass filters.

4. Determine which key was pressed, 0-9, A-D, *, or # by converting frequency pairs back into key names according to Table 1.1.

It is possible to decode DTMF signals using a simple FIR *filter bank*. A *filter bank* is an array of digital filters that have different passband. The filter bank in Fig. 1.1 consists of eight bandpass filters each of which passes only one of the eight possible DTMF frequencies. The input signal for all the filters is the same DTMF signal $x[n]$.

Here is how the system should work: When the input to the filter bank is a DTMF signal, the outputs from two of the bandpass filters (BPFs) should be larger than the rest. If we can detect (or measure) which two outputs are the large ones, then we know the two corresponding frequencies. These frequencies are then used as row and column pointers to determine the key from the DTMF code. A good measure of the output levels is the peak value at the filter outputs, because when the BPF is working properly it should pass only one sinusoidal signal and the peak value would be the amplitude of the sinusoid passed by the filter. More discussion of the detection problem can be found later in this lab.

1.7 Preparation

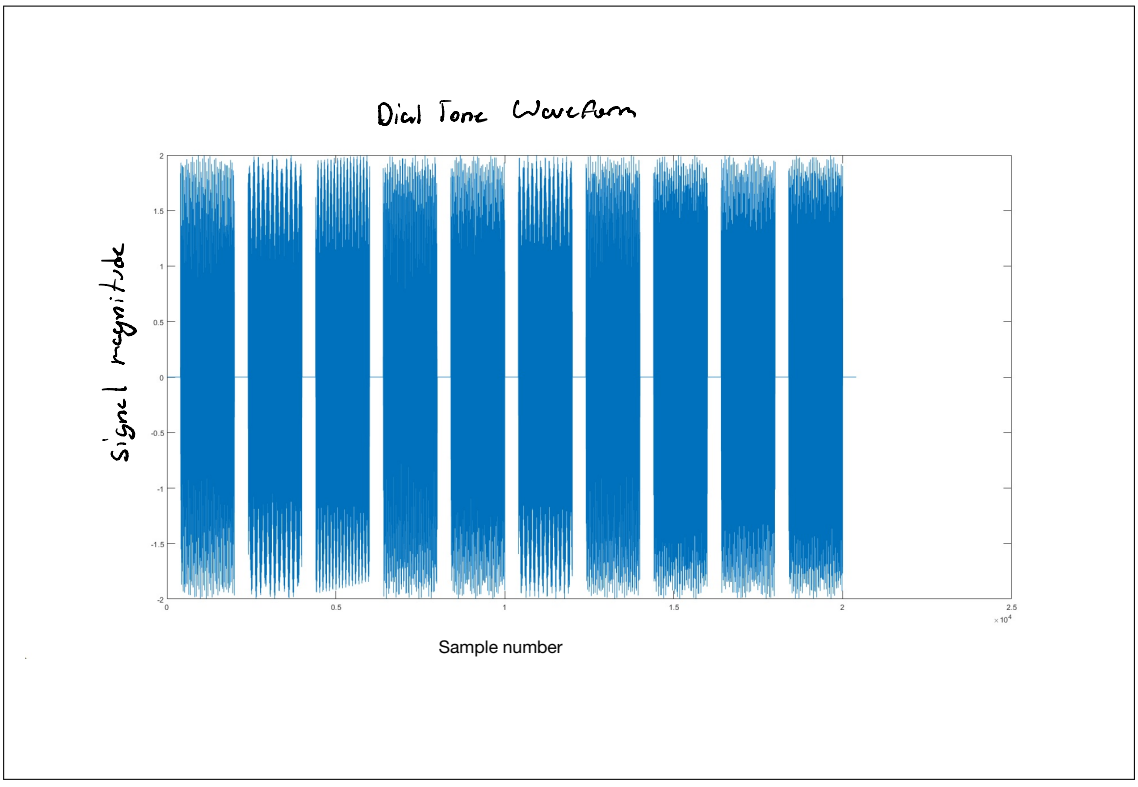
1.7.1 DTMF Encoder

In the `ME705_LabSet2024.zip` there is a MATLAB script named `dtmfdial.m`. This code outputs a vector of samples with sampling rate $f_s = 8000$ Hz containing a DTMF tones, one tone pair per key, that corresponds to a dialled key given by the first argument of the function `keyNames`. The duration of each tone pair is 200 msec and is separated from the tones of adjacent keys by a silence for 50 msec. The second argument of the code is the SNR of the generated DTMF tone assuming that some white (Gaussian) noise is added to the signal while it's transmitted. If no second argument is given, no noise (i.e. SNR: ∞ dB) is added to the DTMF signal.

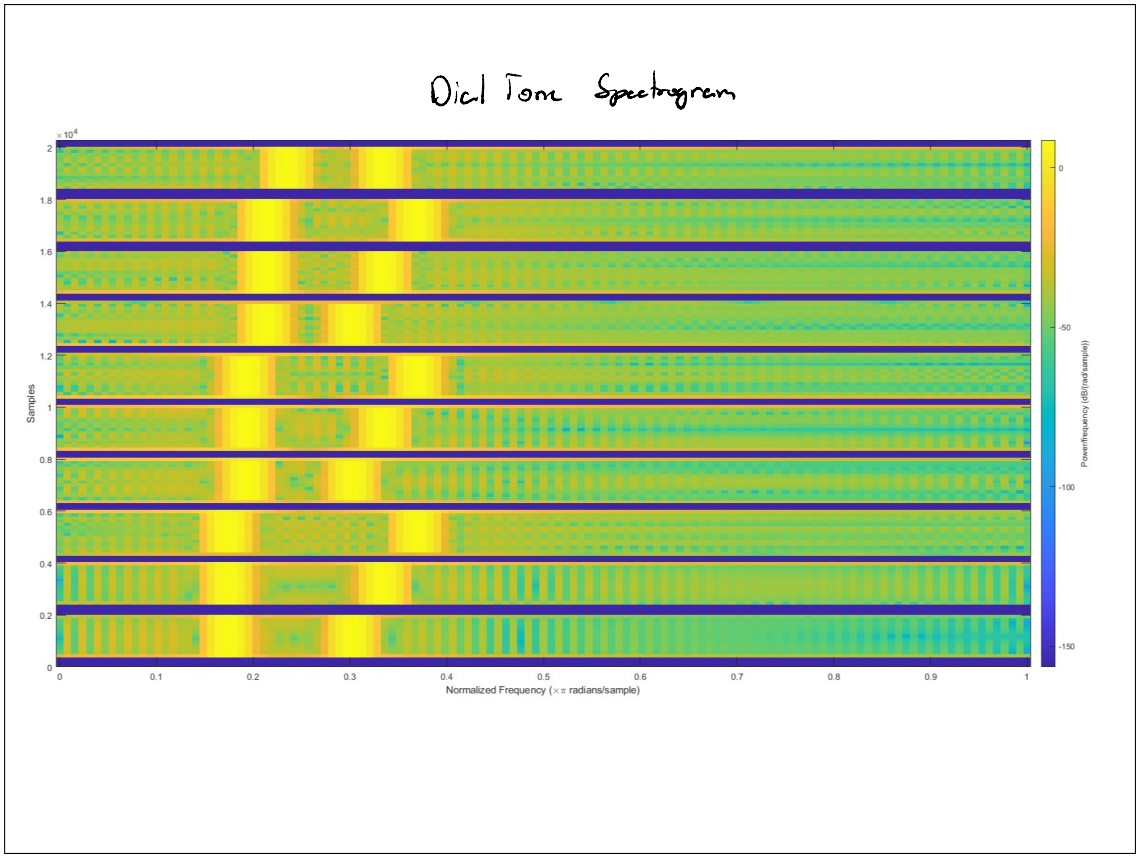
Tasks

1. Generate a DTMF tone without noise (SNR: ∞ dB) for an arbitrary 10 digits phone number consisting of the keys that may be used in a DTMF signal (i.e. 0-9, A-E, * or #). Listen to the generated signal using `sound` command.
2. Draw the waveform and spectrogram of the generated tone and attach them in the box below. For drawing the spectrogram use a hamming window for extracting each frame. Choose the length of the window carefully so that the DTMF tones and the silence are clearly seen in the spectrogram. **[4 marks]**

Waveform:



Spectrogram:



1.7.2 Simple Bandpass FIR Filter Design by window-based method

There are various ways to design an FIR filter. One relatively simple way is to use the window-based method. With the window-based method the filter coefficients (i.e. impulse response) of the FIR filter are derived from an ideal frequency response of the filter to be designed. For bandpass filters, the ideal filter response would consist of unit impulse whose peak is located at the centre frequency Ω_c of the bandpass filter, namely

$$H(\Omega) = \delta(\Omega - \Omega_c) + \delta(\Omega + \Omega_c). \tag{1.3}$$

For example, we pick $\Omega_c = 0.2\pi$ if we want the peak of the filter's passband to be centred at 0.2π. Note that Ω is a **normalised** frequency.

The filter coefficients are then calculated by applying the inverse discrete-time Fourier transform (IDTFT) to the ideal filter response, which is then truncated by multiplying a window to make the filter length be finite. The inverse DTFT of the ideal BPF in (1.3) is given by

$$h[n] = \beta \cos[\Omega_c n]. \tag{1.4}$$

Thus the filter coefficients of the designed bandpass FIR filter will be

$$b[n] = \beta \cos[\Omega_c n] w[n], \tag{1.5}$$

where $w[n]$ denotes the finite-length $(L+1)$-sample window. Here L corresponds to the order of the FIR filter. For example a rectangular window is defined by

$$w[n] = \begin{cases} 1 & n = 0, 1, \cdots, L \\ 0 & \text{else} \end{cases} \tag{1.6}$$

Also it is possible to choose β so that the maximum value of the frequency response magnitude will be one (i.e. normalised).

Now the bandwidth of the bandpass filter is controlled by the window size, which is determined by L; the larger the value of L, the narrower the bandwidth. Follow the instructions below to see the effect of window size to the frequency response of the filter.

Tasks

1. Generate a bandpass filter that will pass a frequency component at $\Omega_c = 0.2\pi$ with the filter order (L) equal to 50 using a rectangular window. Numerically find the value of β so that the maximum value of the frequency response magnitude will be one. To this end let MATLAB measure the peak value of the unscaled frequency response, and then have MATLAB compute β to scale the peak to be one which is normally the inverse of the peak value you have found. Answer the value of β you found in the answer box below.
   Hint: Use `freqz` to calculate the frequency response of the designed filter. **[3 marks]**

   $\beta = 0.04$ thus $H = 1$

2. The passband of the BPF is defined by the region of the frequency response where $|H(e^{j\Omega})|$ is close to its maximum value of one. Typically, the passband width is defined as the length of the frequency region where $|H(e^{j\Omega})|$ is greater than $\frac{1}{\sqrt{2}}$. Find the cut-off frequencies and bandwidth of the BPF in normalised frequency. **[3 marks]**
   Hint: Use `find` to locate those frequencies from the frequency response calculated above. Remember `freqz` outputs the angular frequency of each frequency bin of the frequency response in its second output argument `W`.

   Lower cut-off: 0.5775 rad
   Higher cut-off: 0.6842 rad
   Bandwidth = 0.1066 rad

3. If the sampling rate is 8000 Hz, determine the analogue frequency of the passband width and the cut-off frequency of the designed BPF. **[3 marks]**

$\theta = \frac{f_s \cdot \omega}{2\pi}$

Lower cut-off = 735.4 Hz

Higher cut-off = 871.1 Hz

Bandwidth = 135.7 Hz

1.8 Lab exercises: DTMF Decoding

A DTMF decoding system needs two pieces of functions: a set of BPFs (i.e. filterbank) to isolate individual frequency components; and a detector to determine whether or not a given component is present. The detector must "score" each BPF output and determine which two frequencies are most likely to be contained in the given DTMF signal. In a practical system where noise and interference are also present, this scoring process is a crucial part of the system design.

To make the whole system work, you will have to write three M-files: `dtmfrun.m`, `dtmfscore.m` and `dtmfdesign.m`. Skeletons of these codes are provided in `ME705_LabSet2024.zip`. An additional M-file `dtmfcut.m` is also provided in `ME705_LabSet2024.zip`. `dtmfrun` is the main function which will call `dtmfdesign`, `dtmfcut`, and `dtmfscore`. The following sections discuss your tasks to complete these functions. You are expected to add your own lines to complete the skeleton codes to implement required functions of each code discussed below.

1.8.1 Bandpass Filter Design: `dtmfdesign.m`

The FIR filters that will be used in the filter bank (Fig. 1.1) are a simple type constructed with window-based method, as already discussed in the Preparation. According to the discussion the impulse response of the FIR filter is simply a sampled cosine truncated by a window

$$h[n] = \beta \cos\left[2\pi \frac{f_b}{f_s} n\right] w[n], \tag{1.7}$$

where f_s is the sampling frequency. The parameter f_b defines the centre frequency of the passband, e.g. we pick $f_b = 852$ if we want to isolate the 852 Hz component from a DTMF signal.

Following is your tasks to complete the skeleton code `dtmfdesign.m` to produce all eight bandpass filters needed for the DTMF decoder.

Tasks (N.B. Answer boxes will be found in the following pages)

1. Write lines that will generate the eight bandpass FIR filters specified in (1.7). Like you did in Section 1.7.2, devise a strategy for picking the constant β so that the maximum value of the frequency response of the filters will be equal to one.

2. Once the filters are designed, you have to make sure that you have designed a correct set of BPFs. In particular, you should justify how to choose L, the order of the filters. Through the following tasks, you are running the $L = 40$ and $L = 80$ cases, and then determine empirically the minimum order L so that the frequency response will satisfy the specifications on passband width and stopband rejection given in the Task 6.

3. Generate the eight (normalised) bandpass filters with $L = 40$ and $f_s = 8000$. Plot the magnitude of the frequency responses all together on one plot (the range $0 \leq \Omega \leq \pi$ is sufficient because $|H(e^{j\Omega})|$ is symmetric) and attach it to the box below. Find the width of passband in analogue frequency by looking for the cut-off frequencies of each BPF (answer the values in the box). Hint: use the `hold` command to keep a plot on the same window. **[4 marks]**

4. Repeat the previous task with $L = 80$ and $f_s = 8000$. **[4 marks]**

5. Comment on the width of the passband of the BPFs by comparing the two sets of BPFs using different L. Why do they vary when different L is given? **[4 marks]**

6. Find minimum value of L that satisfies the following filter design specifications.
   Filter Design Specifications: For each of the eight BPFs, only one frequency of the DTMF frequencies should lie within the *passband* of the BPF and all other DTMF frequencies lie in the *stopband*. The *passband* of the BPF is defined by the region of Ω where $|H(e^{j\Omega})| \geq \frac{1}{\sqrt{2}}$ (i.e. -3 dB). The *stopband* of the BPF is defined by the region of Ω where $|H(e^{j\Omega})| < \frac{1}{4}$ (i.e. -12 dB). **[8 marks]**

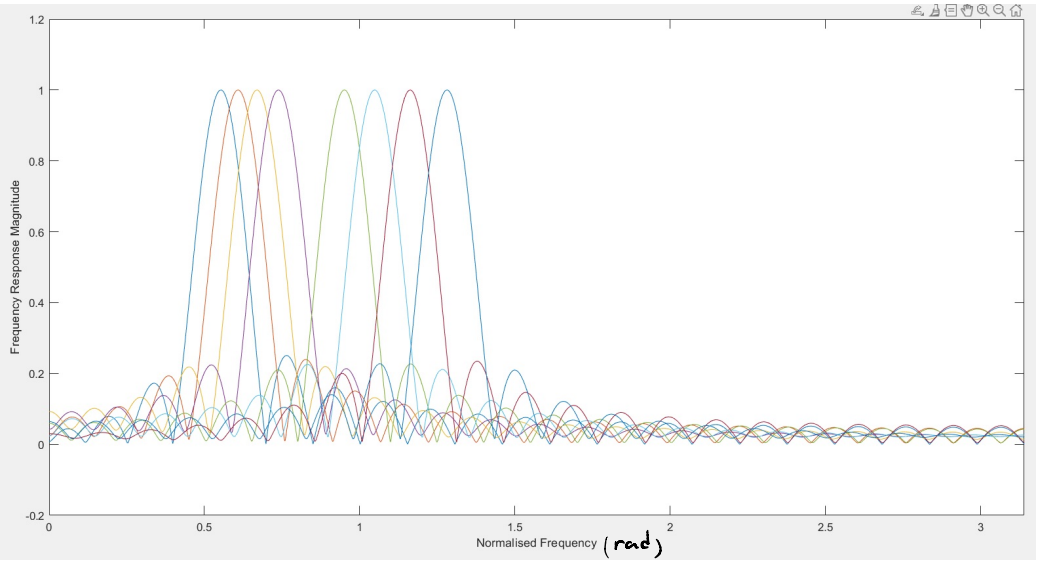
7. Comment on the selectivity of the bandpass filters. Is each filter's passband narrow enough so that only one frequency component lies in the passband and the others are in the stopband? Since the same value of L is used for all the filters, which filter drives the problem? In other words, which centre frequency is the hardest to meet the specifications for the chosen value of L? **[2 marks]**

For the rest of this lab, unless otherwise specified, use L that you determined in Task6.

Answer box for Task3

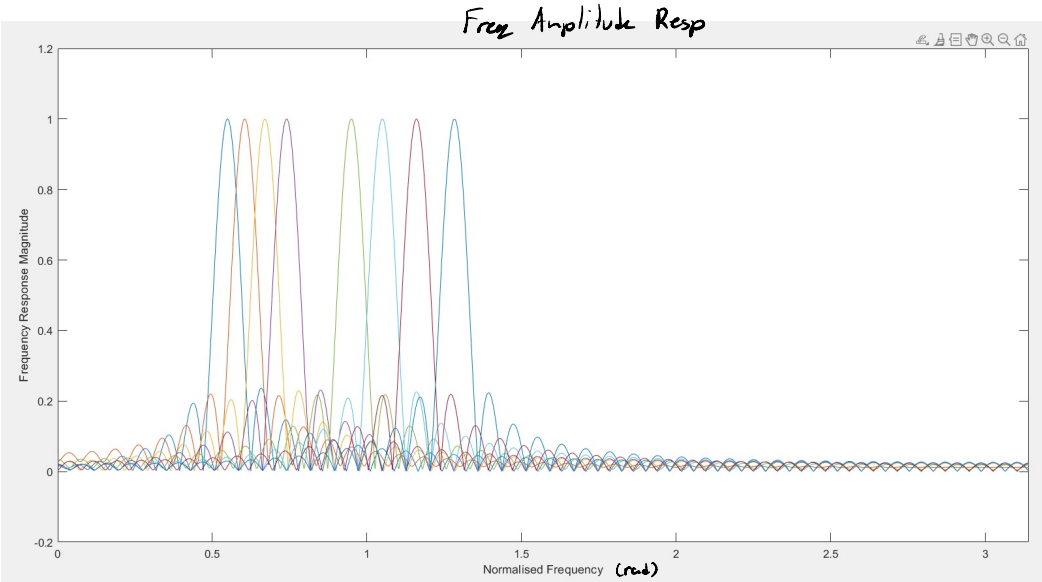
f(Hz)	697	770	852	941	1209	1336	1477	1633
Bandwidth (Hz)	168	173.8	173.8	170.9	168	170.9	171.9	170.9

Freq Amplitude Resp



Answer box for Task4

f(Hz)	697	770	852	941	1209	1336	1477	1633
Bandwidth (Hz)	86	87	86	87	87	87	87.9	87.9



Answer box for Task5

Task 4 uses a filter length double that of task 3. This results in passband width halving and becoming much more consistent.

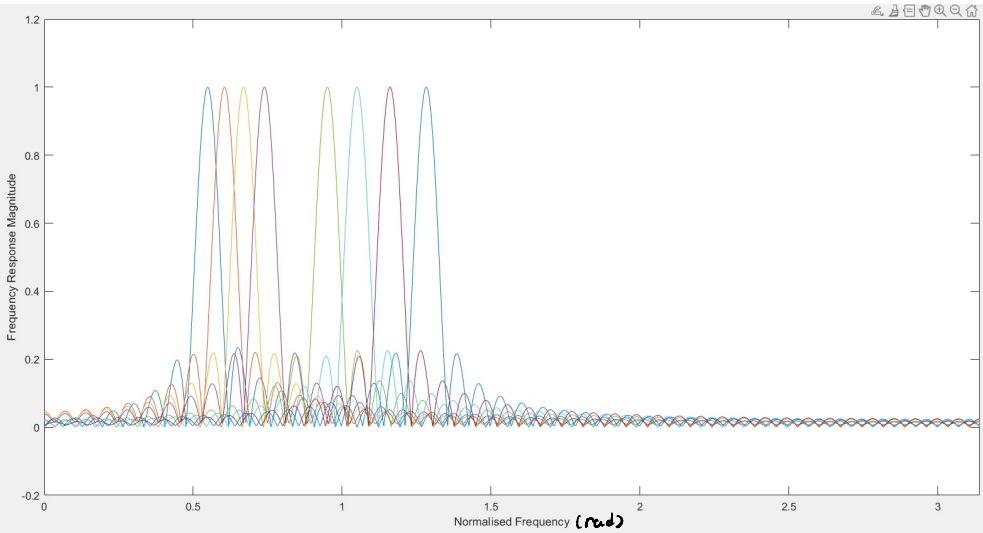
This is because, by increasing filter length, we are increasing the filter order leading to better attenuation and therefore sharper cutoff which decreased bandwidth.

A decrease in filter order leads to worse attenuation, larger bandwidths and less consistent bandwidths.

Answer box for Task6

Lmin = 86

Freq Amplitude Resp



Answer box for Task7

The 2nd BPF at center freq 770 Hz is the hardest to meet specifications as it has the worst overlaps with the first and third BPFs.

Due to the first 2 BPFs being so close to each other, a higher value of L was required so their center frequencies didn't overlap with eeeh other passbands.

The other BPFs would not need such a high value of L as their passbands are already narrow enough vs their adjacent BPFs

1.8.2 A Scoring Function: `dtmfscore.m`

The second objective is developing a detector; a process that requires a binary decision on the presence or absence of the individual tones. In order to make the signal detection an automated process, we need a score function that rates the different possibilities.

Tasks

1. Complete the skeleton of the `dtmfscore.m` function. The input signal `xx` to the `dtmfscore` must be a short segment of the DTMF signal that corresponds to a single key. The task of breaking up the original DTMF signal into such short segment is done by calling the prescribed function `dtmfcut` prior to running `dtmfscore`, which provides the start and end sample indices of each segment. Use the following rule for scoring: the score equals one when $\max_n |y_i[n]| \geq 0.59$; otherwise, it is zero. The signal $y_i[n]$ is the output of the i-th BPF.

   Prior to filtering and scoring, make sure that the input signal $x[n]$ is normalised to the range $[-2, +2]$. With this scaling the two sinusoids that make up $x[n]$ should each have amplitudes of approximately 1.0. Therefore the scoring threshold of 0.59 corresponds to a 59% level for detecting the presence of one sinusoid.

2. The scoring rule above depends on proper scaling of the frequency response of the bandpass filters. Explain why the maximum value of the magnitude for $H(e^{j\Omega})$ must be equal to one for each filter. Consider the fact that both sinusoids in the DTMF tone will experience a known gain (or attenuation) through the bandpass filter, so the amplitude of the output can be predicted if we control both the frequency response and the amplitude of the input. The score returned from `dtmfscore` must be either a 1 or a 0 for each tone. **[3 marks]**

We don't want the filter to act as an amplifier when a signal is passed through. Thus the max magnitude should be normalised to 1. This allows the output signal to be much more predictable as its bandwidth is unchanged. Thus, the scoring system can be correctly designed

1.8.3 DTMF Decode Function: `dtmfrun.m`

Finally the *main* function of DTMF decoding `dtmfrun`, which calls the codes you have already developed, needs to be written to have the whole decoding system to run. You are expected to add lines to the skeleton code provided so that he function `dtmfrun` works as follows:

First, it designs the eight bandpass filters using `dtmfdesign`, then it breaks the input signal down into individual segments using the prescribed `dtmfcut`. For each segment, it will have to call the `dtmfscore` to score the different BPF outputs and then determine the key for that segment. The final output is the list of decoded keys. You must add the logic to decide which key is present. Summarise your decoding logic in the following answer box. **[6 marks]**

Because each key has a unique freq combination, the sum of a key's freqs will be unique.

1. Get freqs of key input from DTMFscore
2. Sum freqs
3. Check sum along a switch of known sums to find what key it is.

1.8.4 Decoding DTMF signal with unknown telephone number

Once you are confident that your codes are running correctly, load the file

`unknown_number_xx.mat`

which stores a DTMF signal carrying 9-digits unknown telephone number. Note that `xx` is your team number that you signed up on Canvas's People page. Find the telephone number using your decoder. **[6 marks]**

Team 1

Number: 099234158

1.8.5 Effect of noise to the decoder

As discussed earlier, in practice the DTMF signal received from a transmission line is usually contaminated by noise. Although efforts should be made to minimise the amount of noise added while the signal travels through the transmission line, you could also make the DTMF decoder more robust[3] to noise.

In the first place, we should investigate how noise would affect your decoder. Generate a DTMF tone WITH noise at various SNR (follow the same procedures as you did in Section 1.7.1 but this time you need to have the second argument, i.e. SNR) and evaluate how robust your decoder is to the noise. Use "01205978436" as your number. Find the minimum SNR (as the closest integer) that your decoder is able to detect the number encoded in the DTMF signal correctly (success rate 90% or higher). **[4 marks]**

Hint: Since the noise added by `dtmfdial` is generated by a random process (Gaussian distribution)[4], the detected number may vary at certain SNRs.

$SNR_{min} = 8$

[3] This means the decoder maintains its performance under low SNR environment.

[4] See lecture slide Chapter 4

1.8.6 Noise robust decoder

Identify the causes of the decoder failing to detect a signal correctly when the SNR is below the value you found in 1.8.5. Discuss what changes should be made to your decoder in order to increase its robustness if some noise is expected to be added while a DTMF signal is transmitted. **[6 marks]**

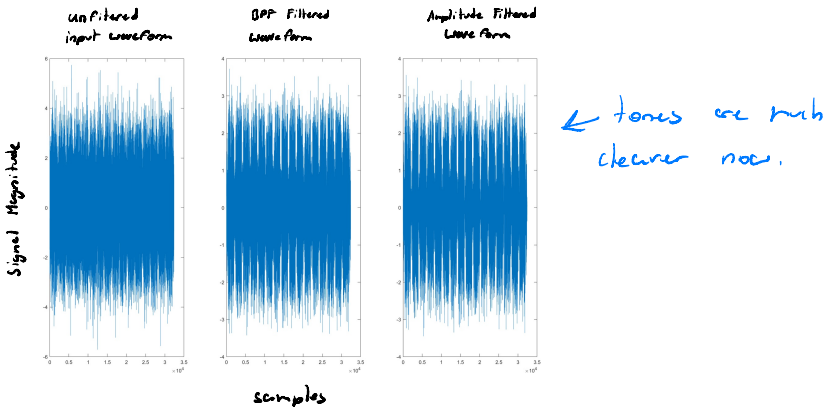
Bonus point You may support your answer by implementing your ideas and showing results using a noisy DTMF signal generated in Section 1.8.5 with various SNR. Up to **5 bonus marks**[5] may be given if your answer is supported by results as an evidence.

DTMFCut is unable to distinguish the start and stop index between tones when a signal is noisy. As a result, DTMFCut will not cut the signal at the right places, and multiple signals may be passed through to DTMFScore, thus returning error.

To improve performance and robustness against noise, we applied a Bandpass filter and a noise gate. First, we passed the input signal through a BPF to eliminate any noise outside of the 8 BPFs centre frequencies. Then we passed that signal through a noise gate to eliminate the amplitude of already low amplitude samples. This is because we noticed that the noise that occurred between tones (in the region where there is 50ms of silence) was ~20% of the amplitude of the key tones.
Finally, we passed the signal through the same BPF one more time to get rid of any added noise to clean the signal.

Additionally, we made adjustments to the DTMFScore function to only take the top 2 frequencies by amplitude rather than checking for any frequency above 59% of the max amplitude. This is because noise above 59% of the max amplitude may accidentally be taken instead. Since the key tones are the loudest, we know for sure the loudest 2 frequencies are the tones.

Our results from this implementation are shown in the figure below. Our algorithm was able to decode a signal with SNR = -4 with an accuracy of > 90%.



This is the end of this lab. **Remember you need to attach copies of all your Matlab codes at the end of this lab work sheet upon submission**.

[5] Note that the full mark of this lab is capped at 60 regardless of the bonus point added.

```
%% task 171
clear;
clc;

% Create Signal
a = dtmfdial(['1','2','3','4', '5', '6', '7', '8', '9', '0']);

% Play tone generated
sound(a);

% Plot waveform and spectrograph
plot(a);
spectrogram(a,hamming(126), 0, 256);
```

```
%% task 172
clear;
clc;

% Values for Filter
L = 50;
N = 4096;
a = 1;
wn = a;
b = zeros(1,L);
omegaC = 0.2*pi;

% Calculating b with beta being zero
j = 1;
for i = 0:L
        b(j) = cos(omegaC * i)*wn;

        j = j + 1;
end

% Calc Frequency Responce
[H, W] = freqz(b, a, N);

% Find Beta that equals magnitude responce of 1 by dividing 1 by max
% magnitude
beta = 1 / max(abs(H));

% ReCalc Frequency Responce
[H, W] = freqz(beta.*b, a, N);

% Plot Responce
plot(abs(H));

% Find range that is above 1/sqrt(2)
result = find(abs(H) > (1/sqrt(2)));

% Get the first and last Normalised Frequency
ftOmeg = W(result(1));
ltOmeg = W(result(size(result,1)));

ftOmeg - ltOmeg;

ftf = ftOmeg * (8000 / (2*pi));
ltf = ltOmeg * (8000 / (2*pi));

bandwidth = ltf - ftf;
```

```
%%% Task 181

%% task 5
clear;
clc;

%initialise variables
fs = 8000;
fcent = [697;770;852;941;1209;1336;1477;1633];

%loop until min L is found
for L = 80:100
    l = findMinval(L, fs, fcent);

    if l
        disp(L)
        break;
    end
end

%% Basic Filter Test Task 3 and 4
clear;
clc;

%initialise variables
L = 80;
fs = 8000;
fcent = [697;770;852;941;1209;1336;1477;1633];

%get signals
[bb, H, W] = dtmfdesign(fcent, L, fs);

%plot
t = linspace(0,pi,4096);
plot(t, abs(H))


%% Function for Task 5



function l = findMinval(L, fs, fcent)
% System parameters

% gets the responce
[bb, H, W] = dtmfdesign(fcent, L, fs);
l = 1;

%loop for all BPFs
```

```
for j = 1:(size(fcent, 1) - 1)
    %find start-stop bandi ndexes
    index = find(abs(H(:,j)) >= 0.25);
    start = W(max(index),j);

    %check if they overlap
    if (start > ((fcent(j + 1) / fs) * 2 * pi))
        l = 0;
        break;
    end
end
end
```

```
%% task 182
clear;
clc;

% Initialise variables
fs = 8000;
L = 86;
fcent = [697;770;852;941;1209;1336;1477;1633];

% Dial tones
a = dtmfdial(['1','2','3','4', '5', '6', '7', '8', '9', '0']);

% Cut up signal
[nstart,nstop] = dtmfcut(a,fs);

k = 1;

% get bpf signals
[bb, H, W] = dtmfdesign(fcent, L, fs);

% get the frequencies of the dial tones
for i = 1:size(nstart, 2)

    freq_section = a(nstart(i):nstop(i));
    sc = dtmfscore(freq_section, bb);

    freq_components(:,i) = nonzeros(sc .* fcent);
end
```

```
%% task 183
clear;
clc;

%initialise variables
fs = 8000;
L = 86;
fcent = [697;770;852;941;1209;1336;1477;1633];

%check if the keys returned is the same as number
number = '01205978436';
a = dtmfdial(number, 7);
keys = dtmfrun(a,L,fs);
```

```
%% task 184
clear;
clc;

%initialise variables
fs = 8000;
L = 86;
fcent = [697;770;852;941;1209;1336;1477;1633];

%get our groups dial tone
b = load('unknown_number_1.mat');
a = b.xx;

%get the keys pressed
keys = dtmfrun(a,L,fs)
```

```
%% task 185
clear;
clc;

%initialise vars
fs = 8000;
L = 86;
fcent = [697;770;852;941;1209;1336;1477;1633];

%arbitrary number
number = '01205978436*';

%initialise trial no.
k = 500;
f = 0;

sum = 0;

% loop for number of trials
for i = 1:k
    a = dtmfdial(number, 8);

    %get the keys
    keys = dtmfrun(a,L,fs);

    %for the keys that were returned, compare them to the arbitrary number
    for j = 1:size(keys,2)
        if keys(j) == number(j)
            %If matching, increment sum
            sum = sum + 1;
        end
        % Increment total number of trials done
        f = f + 1;
    end

end

%calculate percentage success
disp(sum / f);
```

```
%% task 186
clear;
clc;

%initialise variables
fs = 8000;
L = 86;
fcent = [697;770;852;941;1209;1336;1477;1633];
number = '01205978436';
k = 500;
f = 0;

sum = 0;

for i = 1:k
    a = dtmfdial(number, -4);

    %pass input through bandpass
    a = bandpass(a, [697 1633], fs);

    %normalise signal
    a1 = a * (1 / max(abs(a)));

    %Noise gate
    for n = 1:size(a,2)
        %0 out samples that are 0.2 of max magnitude
        if abs(a1(n)) < 0.2
            a(n) = 0;
        end
    end

    %pass through bandpass again to clear added noise
    a = bandpass(a, [697 1633], fs);

    %get the keys
    keys = dtmfrun(a,L,fs);

    %compare the generated key vs the number input
    s1 = size(keys,2);
    s2 = size(number,2);

    if s1 > s2
        s3 = s2;
    else
        s3 = s1;
    end

    for j = 1:s3
        if keys(j) == number(j)
```

```
            sum = sum + 1;
        end

        f = f + 1;
    end

end

%calc success %
disp(sum / f);
```

```
function  [hh, H, W] = dtmfdesign(fcent, L, fs)
%DTMFDESIGN
%     hh = dtmfdesign(fcent, L, fs)
%       returns a matrix where each column is the
%       impulse response of a BPF, one for each frequency
%  fcent = vector of center frequencies
%      L = length of FIR bandpass filters
%     fs = sampling freq
%
% The BPFs must be scaled so that the maximum magnitude
% of the frequency response is equal to one.
%=========================================
% [697;770;852;941;1209;1336;1477;1633]; list of centre frequencies

%%%% add your lines below to complete the code

%w: normalised angular frequency
%h: frequency response vector


nn = 0:L;

for i=1:size(fcent)

    %calculate filter coeffs
    bb = cos(2*pi*(fcent(i)/fs)*nn);

    %get the greatest unscaled val to scale BPF down
    [h, w] = freqz(bb, 1, 4096);
    maxVal = max(abs(h)); %find the maximum value

    %scaled
    bb_scaled = (1/maxVal)*bb;
    [h, w] = freqz(bb_scaled, 1, 4096);
    hh(:, i) = bb_scaled;
    H(:, i) = h;
    W(:, i) = w;

    % %plot stuff
    % plot(w, abs(h));
    % xlabel('Normalised Frequency')
    % ylabel('Frequency Response Magnitude')
    % xlim([0, 3.14]);
    % ylim([-0.2, 1.2]);
    % hold on;
    %
    %
    % %find the cutoff frequencies
    % lcf_index = find(diff(abs(h) > 0.7071) == 1) + 1;
```

```
    % ucf_index = find(diff(abs(h) < 0.7071) == 1) + 1;
    % lcf = w(lcf_index);
    % ucf = w(ucf_index);
    %
    % fprintf('For frequency number %d\n', i)
    % disp('Lower cutoff frequency:');
    % disp((lcf*fs)/(2*pi));
    % disp('Upper cutoff frequency:');
    % disp((ucf*fs)/(2*pi));
    % disp('Passband width');
    % disp(((ucf-lcf)*fs/(2*pi)));

end
end
```

```
function [nstart,nstop] = dtmfcut(xx,fs)
%DTMFCUT   find the DTMF tones within x[n]
%  usage:
%       indx = dtmfmain(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 = 0.1;       %-- 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,:);
```

```
function  [sc] = dtmfscore(xx, hh)
%DTMFSCORE
%          sc = dtmfscore(xx, hh)
%    returns a score based on the maximum amplitude of the filtered output
%     xx = input DTMF signal
%     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

xx_2 = xx*(1/max(abs(xx)));    %---Scale x[n] to the range [-2,+2]
sc = zeros(size(hh, 2), 1);

% get freq resp of signal
for i = 1:size(hh,2)
    y(i) = max(abs(conv(hh(:,i), xx_2)));

end

%sort the freq resp by magnitude
y_1 = sort(y);

n = size(y_1,2);

% return the top two frequencies by magnitude
for i = 1:size(hh,2)

    if (y(i) == y_1(n))
        sc(i) = 1;
    elseif (y(i) == y_1(n - 1))
        sc(i) = 1;
    end
end
```

```
function keys = dtmfrun(xx,L,fs)
%DTMFRUN    keys = dtmfrun(xx,L,fs)
%    returns the list of key numbers corresponding
%      to the DTMF waveform, xx.
%      L = filter length
%     fs = sampling freq

freqs = [697;770;852;941;1209;1336;1477;1633];  % list of centre frequencies

hh = dtmfdesign( freqs,L,fs );
%   hh = MATRIX of all the filters. Each column contains the impulse
%        response of one BPF (bandpass filter)

dtmf.keys = ...
['1','2','3','A';
'4','5','6','B';
'7','8','9','C';
'*','0','#','D'];
dtmf.colTones = [1209,1336,1477,1633];
dtmf.rowTones = [697;770;852;941];

[nstart,nstop] = dtmfcut(xx,fs);   %<--Find the start and end points of each tone

%%%% add your lines below to complete the code

k = 1;

[bb, H, W] = dtmfdesign(freqs, L, fs);

% loop through the indexes
for i = 1:size(nstart, 2)

    %get the score of the freqs
    freq_section = xx(nstart(i):nstop(i));
    sc = dtmfscore(freq_section, bb);

    %sum the freqs if the score is valid and match the sum to the key
    switch sum(nonzeros(sc .* freqs))
        case (697 + 1209)
            keys(i) = '1';
        case (697 + 1336)
            keys(i) = '2';
        case (697 + 1477)
            keys(i) = '3';
        case (697 + 1633)
            keys(i) = 'A';
        case (770 + 1209)
            keys(i) = '4';
        case (770 + 1336)
```

```
            keys(i) = '5';
        case (770 + 1477)
            keys(i) = '6';
        case (770 + 1633)
            keys(i) = 'B';
        case (852 + 1209)
            keys(i) = '7';
        case (852 + 1336)
            keys(i) = '8';
        case (852 + 1477)
            keys(i) = '9';
        case (852 + 1633)
            keys(i) = 'C';
        case (941 + 1209)
            keys(i) = '*';
        case (941 + 1336)
            keys(i) = '0';
        case (941 + 1477)
            keys(i) = '#';
        case (941 + 1633)
            keys(i) = 'D';
        otherwise
            keys(i) = 'e';
    end
end


end
```

```
function xx = dtmfdial(keyNames,varargin)
%DTMFDIAL Create a signal vector of tones which will dial
% a DTMF (Touch Tone) telephone system.
%
% usage: xx = dtmfdial(keyNames,SNR)
% keyNames = vector of characters containing valid key names (e.g. '12345567890')
% SNR = signal-to-noise ratio in dB (no noise will be added if this argment is left blank)
% xx = signal vector that is the concatenation of DTMF tones.
%

fs = 8000;

dtmf.keys = ...
['1','2','3','A';
'4','5','6','B';
'7','8','9','C';
'*','0','#','D'];
dtmf.rowTones = [1209,1336,1477,1633];
dtmf.colTones = [697;770;852;941];

t_tone = 0.2; %sec
t_gap = 0.05; %sec

chk = 0;
for k = 1:length(keyNames)
    chk = chk + not(sum(sum(keyNames(k)==dtmf.keys)));
end
if chk ~= 0
   error('input includes invalid characters (input should be either 0-9 or A-D or # or *)');
end

output = zeros(1,fs*t_gap);
for k = 1:length(keyNames)
    [ii, jj] = find(keyNames(k)==dtmf.keys);
    freq1 = dtmf.colTones(ii);
    freq2 = dtmf.rowTones(jj);
    tone1 = sin(2*pi*freq1*[1/fs:1/fs:t_tone]);
    tone2 = sin(2*pi*freq2*[1/fs:1/fs:t_tone]);

    output = [output, (tone1+tone2)];
    output = [output, zeros(1,fs*t_gap)];
end

noise = randn(1,length(output));

if nargin > 1
    SNR = varargin{1};
```

```
    xx= output + noise * 10^(-SNR/20);
else
    xx= output;
end
```