GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2001
Lab #8: Everyday Sinusoidal Signals

Date: 5-29 March 2001

******* Lab #8 will be graded out of 150 points. *******

This is *the official* Lab #8 description; it is based on Lab A of Lab C.7 in Appendix C of the text, but the warm-up has been changed quite a bit.

The Warm-up section of each lab must be completed in Lab and the steps marked *Instructor Verification* must also be signed off **during the lab time**.

Although the report on Lab #7 is not due until the week of 19-March, you should finish it as soon as possible after returning from Spring Break so that you can begin working on Lab #8. This lab is much more difficult ant time consuming than any of the previous labs. To be specific, you should try to finish the Verifications for and Section 4 of Lab #8 during the week of 12-March. The **Instructor Verification** sheet should be handed in during lab time during the week of 12–15 March.[1]

The final lab report for this lab will be **FORMAL:** discuss your results from Sections 4 and 5.

The formal report will be **due during the week of 26–29 March at the start of your lab**.

---

1 Introduction

This lab introduces a practical application where sinusoidal signals are used to transmit information: a touch-tone dialer. Bandpass FIR filters can be used to extract the information encoded in the waveforms. The goal of this lab is to design and implement bandpass FIR filters in MATLAB, and do the decoding automatically. In the experiments of this lab, you will use `firfilt()`, or `conv()`, to implement filters and `freqz()` to obtain the filter's frequency response.[2] As a result, you should learn how to characterize a filter by knowing how it reacts to different frequency components in the input.

1.1 Frequency Response of FIR Filters

The output or *response* of a filter for a complex sinusoid input, $e^{j\hat{\omega}n}$, depends on the frequency, $\hat{\omega}$. Often a filter is described solely by how it affects different frequencies—this is called the *frequency response*. The frequency response of a general FIR linear time-invariant system is[3]

$$H(e^{j\hat{\omega}}) = \mathcal{H}(\hat{\omega}) = \sum_{k=0}^{M} b_k e^{-j\hat{\omega}k} \tag{1}$$

MATLAB has a built-in function for computing the frequency response of a discrete-time LTI system. The following MATLAB statements show how to use `freqz` to compute and plot the magnitude (absolute value)

[1] No later than 19–23 March.

[2] If you are working at home and do not have the function `freqz.m`, there is a substitute available called `freekz.m`. You can get it from the ECE-2025 WebCT page.

[3] The notation $H(e^{j\hat{\omega}})$ is used in place of $\mathcal{H}(\hat{\omega})$ for the frequency response because we will eventually connect this notation with the z-transform, $H(z)$, in Chapter 7.

of the frequency response of an L-point averaging system as a function of $\hat{\omega}$ in the range $-\pi \leq \hat{\omega} \leq \pi$:

```
bb = ones(1,L)/L;         %-- Filter Coefficients
ww = -pi:(pi/100):pi;     %-- omega hat frequency axis
HH = freqz(bb, 1, ww);     %<--freekz.m is an alternative
subplot(2,1,1);
plot(ww, abs(HH))
subplot(2,1,2);
plot(ww, angle(HH))
xlabel('Normalized Radian Frequency')
```

We will always use capital `HH` for the frequency response. For FIR filters, the second argument of `freqz( _, 1, _ )` must always be equal to `1`. The frequency vector `ww` should cover the interval $-\pi \leq \hat{\omega} \leq \pi$ for $\hat{\omega}$, and its spacing must be fine enough to give a smooth curve for $H(e^{j\hat{\omega}})$.

2 Background

2.1 Telephone Touch Tone[4] Dialing

Telephone touch pads generate *dual tone multiple frequency* (DTMF) signals to dial a telephone. When any key is pressed, the tones of the corresponding column and row (in Fig. 1) are generated and summed, hence dual tone. As an example, pressing the **5** key generates a signal containing the sum of the two tones 770 Hz and 1336 Hz together.

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

Figure 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. Common household phones do not have the fourth column.

The frequencies in Fig. 1 were chosen (by the design engineers) to avoid harmonics.[5] 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.[6] This makes it easier to detect exactly which tones are present in the dial signal in the presence of non-linear line distortions. [7]

2.2 DTMF Decoding

There are several steps to decoding a DTMF signal:

[4] Touch Tone is a registered trademark

[5] Keys A-D are not implemented on commercial telephone sets, but are used in some military and other signaling applications.

[6] More information can be found at: `http://www.winternet.com/~genave/dtmf.html`, or search for "DTMF" on the internet.

[7] A recent paper on a DSP implementation of the DTMF decoder, "A low complexity ITU-compliant dual tone multiple frequency detector", by Dosthali, McCaslin and Evans, in *IEEE Trans. Signal Processing*, March, 2000, contains a short discussion of the DTMF signaling system. You can get this paper on-line from the GT library, and you can also get it at `http://www.ece.utexas.edu/ bevans/papers/2000/dtmf/index.html`.

1. Divide the 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 sinusoidal components.
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**, *****, or **#** by converting frequency pairs back into key names according to Fig. 1.

It is possible to decode DTMF signals using a simple FIR filter bank. The filter bank in Fig. 2 consists of eight bandpass filters which each pass only one of the eight possible DTMF frequencies. The input signal for all the filters is the same DTMF signal.

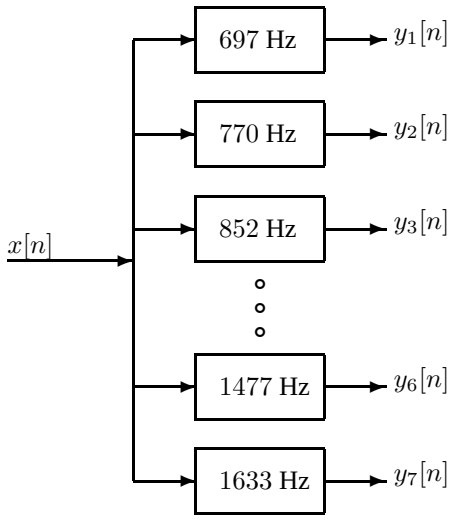

Figure 2: Filter bank consisting of bandpass filters which pass frequencies corresponding to the eight DTMF component frequencies listed in Fig. 1.

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 detect (or measure) which two are the large ones, then we know the two corresponding frequencies. These frequencies are then used to determine 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 that sinusoid. More discussion of the detection problem can be found in Section 5.

3 Warm-up: DTMF Synthesis

3.1 Signal Concatenation

In a previous lab, a very long music signal was created by joining together many sinusoids. When two signals are played one after the other, the composite signal is created by the operation of *concatenation.* In MATLAB, this can be done by making each signal a row vector, and then using the matrix building notation as follows:

```
xx = [ xx, xxnew ];
```

where xxnew is the sub-signal being appended. The length of the new signal is equal to the sum of the lengths of the two signals xx and xxnew. A third signal could be added later on by concatenating it to xx.

Explain how the following program uses frequency information stored in a table to generate a long signal via concatenation. Determine the size of the table and all of its entries, and then state the playing order of the frequencies. Determine the total length of the signal played by the soundsc function. How many samples and how many seconds?

Instructor Verification (separate page)

```
ftable = [1;2;3;4;5]*[100,250]
fs = 11025;
xx = [ ];
disp('--- Here we go through the Loop ---')
keys = ceil(9.9*rand(1,7)+0.1);   %--- Make a random test sequence
for ii = 1:length(keys)
  kk = keys(ii);
  xx = [xx,zeros(1,400)];
  krow = ceil(kk/2);
  kcol = rem(kk-1,2) + 1;
  disp(['kk=',num2str(kk),'  krow=',num2str(krow),...
    ' kcol=',num2str(kcol), '  freq=',num2str(ftable(krow,kcol))])
  xx = [xx, cos(2*pi*ftable(krow,kcol)/fs*(0:1199)) ];
end
soundsc(xx,fs);
```

3.1.1 Comment on Efficiency

In MATLAB the concatenation method, xx = [xx, xxnew], would append the signal vector xxnew to the existing signal xx. However, this becomes an *inefficient* procedure if the signal length gets to be very large. The reason is that MATLAB must re-allocate the memory space for xx every time a new sub-signal is appended via concatenation. If the length xx were being extended from 400,000 to 401,000, then a clean section of memory consisting of 401,000 elements would have to be allocated followed by a copy of the existing 400,000 signal elements and finally the append would be done. This is clearly inefficient, but would not be noticed for short signals.

An alternative is to pre-allocate storage for the complete signal vector, but this can only be done if the final length is known ahead of time.

3.2 Overlay Plotting

Sometimes it is convenient to overlay information onto an existing MATLAB plot. The MATLAB command hold on will inhibit the figure erase that is usually done just before a new plot. Demonstrate that you can do an overlay by following these instructions:

(a) Plot the magnitude response of the 5-point averager, created from HH=freqz(ones(1,5)/5,1,ww). Make sure that the horizontal frequency axis extends from $-\pi$ to $+\pi$.

(b) Use the stem function to place vertical markers at the zeros of the frequency response.

```
hold on, stem(2*pi/5*[-2,-1,1,2],0.3*ones(1,4),'r.'), hold off
```

3.3 DTMF Dial Function

Write a function, `dtmfdial`, to implement a DTMF dialer based on the frequency table defined in Fig. 1. A skeleton of `dtmfdial.m` is given in Fig. 3. In this warm-up, you must complete the dialing code so that

```
function dtmfsig = dtmfdial(nums,fs)
%DTMFDIAL  Create a vector of tones which will dial
%          a DTMF (Touch Tone) telephone system.
%
% usage:  dtmfsig = dtmfdial(nums,fs)
%      nums = vector of numbers ranging from 1 to 16
%        fs = sampling frequency
%   dtmfsig = vector containing the corresponding tones.
%
tone_cols = ones(4,1)*[1209,1336,1477,1633];
tone_rows = [697;770;852;941]*ones(1,4);
```

Figure 3: Skeleton of `dtmfdial.m`, a DTMF phone dialer. Complete this function with additional lines of code.

it implements the following:

1. The input to the function is a vector of numbers, each one being between 1 and 16. Note that these input numbers are **NOT** in general equal to the key number on the telephone. For convenience in programming we will make the following correspondences between the integers 1-16 and the keys listed in the table in Fig. 1: (1-**1**), (2-**2**), (3-**3**), (4-**A**), (5-**4**), (6-**5**), (7-**6**), (8-**B**), (9-**7**), (10-**8**), (11-**9**), (12-**C**), (13-*****), (14-**0**), (15-**#**), and (16-**D**).

2. The output should be a vector of samples containing the DTMF tones, sampled at $f_s = 11025$ Hz. The duration of each tone should be about 0.25 sec., and a silence, about 0.1 sec. long, should separate the DTMF tones. These times can be fixed in `dtmfdial.` (You do not need to make them variable in your function.) Remember that each DTMF signal is the sum of a pair of (equal amplitude) sinusoidal signals.

3. The frequency information is given as two 4×4 matrices (`tone_rows` and `tone_cols`): one contains the row frequencies, the other has the column frequencies. You can translate a digit such as **7** (which stands for the **6** key) into the correct location in these 4×4 matrices by using quotients and remainders. For example, the key **6** is in row 2 and column 3, so we would generate sinusoids with frequencies equal to `tone_rows(2,3)` and `tone_cols(2,3)`.

   To convert an integer index to its corresponding row-column indices, consider the following example: If we divide 7 by 4, then the quotient is 1 and the remainder is 3. To get the correct row, we must round the quotient up to 2. You will have to generalize this idea to make the dialing work properly, but the code given in Section 3.1 contains the basic idea. Check out `help rem` and `help ceil` for some hints. Also, consult the MATLAB code in Section 3.1 above and modify it for the 4×4 tables in `dtmfdial.m`.

Your function should create the appropriate tone sequence to dial an arbitrary phone number. When played through a telephone handset, the output of your function will be able to dial the phone. You could use `specgram` to check your work.[8]

Instructor Verification (separate page)

[8] In MATLAB the demo called `phone` also shows the waveforms and spectra generated in a DTMF system.

4 Bandpass Filter Design

Sections 4 and 5 should be included in your lab report.

4.1 Simple Bandpass Filter Design

The L-point averaging filter is a lowpass filter. Its passband width is controlled by L, being inversely proportional to L. It is also possible to create a filter whose passband is centered around some frequency other than zero. One simple way to do this is to define the impulse response of an L-point FIR as:

$$h[n] = \frac{2}{L}\cos(\hat{\omega}_c n), \qquad 0 \leq n < L$$

where L is the filter length, and $\hat{\omega}_c$ is the center frequency that defines the frequency location of the passband. For example, we pick $\hat{\omega}_c = 0.2\pi$ if we want the peak of the filter's passband to be centered at 0.2π. The bandwidth of the bandpass filter is controlled by L; the larger the value of L, the narrower the bandwidth. This particular filter is also discussed in the section on useful filters in Chapter 7 of *DSP First*.

(a) Generate a bandpass filter that will pass a frequency component at $\hat{\omega} = 0.5\pi$. Make the filter length (L) equal to 32. Make a plot of the frequency response magnitude and phase. Measure the response of the filter (magnitude and phase) at the following frequencies of interest: $\hat{\omega} = 0.1\pi k$, for $k = -10, -9, \ldots, 9, 10$. Summarize the values in a table.
Hint: use MATLAB's `freqz()` function to calculate these values, or the `find()` function to extract this information from the vector that produce the plot.

(b) The *passband* of the BPF filter is defined by the region of the frequency response where $|\mathcal{H}(\hat{\omega})|$ is close to its maximum value of one. Typically, the passband width is defined as the length of the frequency region where $|\mathcal{H}(\hat{\omega})|$ is greater than $1/\sqrt{2} = 0.707$. Note: you can use MATLAB's `find` function to locate those frequencies where the magnitude satisfies $|\mathcal{H}(\hat{\omega})| \geq 0.707$ (similar to Fig. 4).

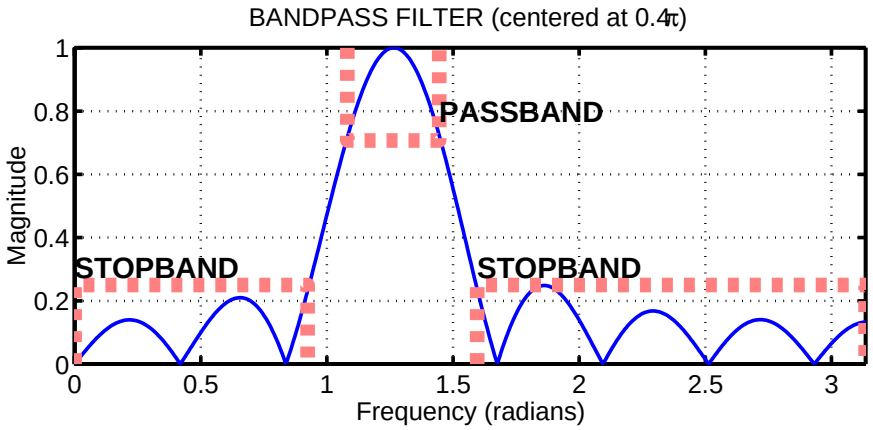

Figure 4: The frequency response of an FIR bandpass is shown with its passband and stopband regions.

Use the plot of the frequency response for the length-32 bandpass filter from part (a), and determine the passband width. Make two other plots of BPFs for $L = 16$ and $L = 64$, and measure the passband width in both. Then explain how the width of the passband is related to filter length L, i.e., what happens when L is doubled or halved.

(c) Comment on the selectivity of the length-64 bandpass filter. Use the frequency response to explain how the filter is able to pass the components at $\hat{\omega} = \pm 0.5\pi$, while reducing or rejecting the others at $\hat{\omega} = 0.1\pi k$ for $k \neq \pm 5$.

(d) If the input signal to the length-32 FIR BPF is:

$$x[n] = 10 + 10\cos(0.5\pi n - \pi/2) + 10\cos(0.6\pi n - \pi/4)$$

Determine (by hand) the formula for the output signal. (Hint: use the magnitude and phase measurements from part (a).) Comment on the relative amplitudes of the three signal components in the output signal.

5 Lab: DTMF Decoding

A DTMF decoding system needs two pieces: a set of bandpass filters (BPF) 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 DTMF tone. In a practical system where noise and interference are also present, this scoring process is a crucial part of the system design, but we will only work with noise-free signals to understand the basic functionality in the decoding system.

To make the whole system work, you will have to write three M-files: `dtmfrun`, `dtmfscore` and `dtmfdesign`. An additional M-file called `dtmfcut` can be downloaded from Web-CT. The main M-file should be named `dtmfrun.m`. It will call `dtmfdesign.m`, `dtmfcut.m`, and `dtmfscore.m`. The following sections discuss how to create or complete these functions.

5.1 Simple Bandpass Filter Design: `dtmfdesign.m`

The FIR filters that will be used in the filter bank (Fig. 2) are a simple type constructed with sinusoidal impulse responses. In the section on useful filters in Chapter 7, a *simple* bandpass filter design method is presented in which the impulse response of the FIR filter is simply a finite-length cosine of the form:

$$h[n] = \beta\cos\left(\frac{2\pi f_b n}{f_s}\right), \qquad 0 \leq n \leq L-1$$

where L is the filter length, and f_s is the sampling frequency. The constant β gives flexibility for scaling the filter's gain to meet a constraint such as making the maximum value of the frequency response equal to one. The parameter f_b defines the frequency location of the passband, e.g., we pick $f_b = 852$ if we want to isolate the 852 Hz component. The bandwidth of the bandpass filter is controlled by L; the larger the value of L, the narrower the bandwidth.

(a) Devise a strategy for picking the constant β so that the maximum value of the frequency response will be equal to one. Write the one or two lines of MATLAB code that will do this scaling operation in general. There are two approaches here:

   (a) *Mathematical:* derive a formula for β from the formula for the frequency response of the BPF. Then use MATLAB to evaluate this closed-form expression for β.

   (b) *Numerical:* let MATLAB measure the peak value of the unscaled frequency response, and then have MATLAB compute β to scale the peak to be one.

```
function  hh = dtmfdesign(fcent, L, fs)
%DTMFDESIGN
%    hh = dtmfdesign(fcent, L, fs)
%      returns a matrix (L by length(fcent)) where each column is the
%      impulse response of a BPF, one for each frequency in fcent
%  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.
```

Figure 5: Skeleton of the `dtmfdesign.m` function. Complete this function with additional lines of code.

(b) Complete the M-file `dtmfdesign.m` which is described in Fig. 5. This function should produce all eight bandpass filters needed for the DTMF filter bank system. Store the filters in the columns of the matrix `hh` whose size is $L \times 8$.

(c) The rest of this section describes how you can exhibit that you have designed a correct set of BPFs. In particular, you should justify how to choose L, the length of the filters. **When you have completed your filter design function, you should run the $L = 25$ and $L = 100$ cases, and then you should determine empirically the minimum length L so that the frequency response will satisfy the specifications on passband width and stopband rejection given in part (f).**

(d) Generate the eight (scaled) bandpass filters with $L = 25$ and $f_s = 11025$. Plot the magnitude of the frequency responses all together on one plot (the range $0 \leq \hat{\omega} \leq \pi$ is sufficient because $|H(e^{j\hat{\omega}})|$ is symmetric). Indicate the locations of each of the eight DTMF frequencies (697, 770, 852, 941, 1209, 1336, 1477, and 1633 Hz) on this plot to illustrate whether or not the passbands are narrow enough to separate the DTMF frequency components. Hint: use the `hold` command and markers as you did in the warm-up.

(e) Repeat the previous part with $L = 100$ and $f_s = 11025$. The width of the passband is supposed to vary inversely with the filter length L. Explain whether or not that is true by comparing the length 100 and length 25 cases.

(f) As help for the previous parts, recall the following definitions: The *passband* of the BPF filter is defined by the region of $\hat{\omega}$ where $|H(e^{j\hat{\omega}})|$ is close to one. Typically, the passband width is defined as the length of the frequency region where $|H(e^{j\hat{\omega}})|$ is greater than $1/\sqrt{2} = 0.707$.

The *stopband* of the BPF filter is defined by the region of $\hat{\omega}$ where $|H(e^{j\hat{\omega}})|$ is close to zero. In this case, it is reasonable to define the stopband as the region where $|H(e^{j\hat{\omega}})|$ is less than 0.25.

Filter Design Specifications: Choose L so that only one frequency lies within the passband of the BPF and all other DTMF frequencies lie in the stopband.

Use the `zoom on` command to show the frequency response over the frequency domain where the DTMF frequencies lie. Comment on the selectivity of the bandpass filters, i.e., use the frequency response to explain how the filter passes one component while rejecting the others. Are the filter's passbands narrow enough so that only one frequency component lies in the passband and the others are in the stopband?

5.2 A Scoring Function: `dtmfscore.m`

The final objective is decoding—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.

(a) Complete the `dtmfscore` function based on the skeleton given in Fig. 6. The input signal `xx` to the `dtmfscore` function must be a short segment from the DTMF signal. The task of breaking up the signal so that each short segment corresponds to one key is done by the function `dtmfcut` prior to calling `dtmfscore`.

The implementation of the FIR bandpass filter is done with the `conv` function, but we could also use `firfilt`. The running time of the convolution function is proportional to the filter length L. Therefore, the filter length L must satisfy two competing constraints: L should be large so that the bandwidth of the BPF is narrow enough to isolate individual frequency components, but making it too large will cause the program to run slowly. **Try to make your system work reliably with the smallest possible value for L.**

```
function  sc = 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(|y[n]|) is greater than, or equal to, 0.96
%      sc = 0 if max(|y[n]|) is less than 0.96
%
xx = xx*(2/max(abs(xx)));   %---Scale x[n] to the range [-2,+2]
```

Figure 6: Skeleton of the `dtmfscore.m` function. Complete this function with additional lines of code.

(b) Use the following rule for scoring: the score equals one when $\max_n |y_i[n]| \geq 0.96$; otherwise, it is zero. The signal $y_i[n]$ is the output of the i-th BPF.

(c) Prior to filtering and scoring, make sure that the input signal $x[n]$ is normalized to the range $[-2, +2]$. With this scaling the two sinusoids that make up $x[n]$ should each have amplitudes of approximately 1.0.[9] Therefore the scoring threshold of 0.96 corresponds to a 96% level for detecting the presence of one sinusoid.

(d) 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 $H(e^{j\hat{\omega}})$ must be equal to one. 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.

[9] The two sinusoids in a DTMF tone have frequencies that are not harmonics. When plotted versus time, the peaks of the two sinusoids will eventually line up.

(e) When debugging your program it might be useful to have a plot command inside the `dtmfscore.m` function. If you plot the first 200–500 points of the filtered output, you should be able to see two cases: either $y[n]$ is a strong sinusoid with an amplitude close to one (when the filter is matched to one of the component frequencies), or $y[n]$ is relatively small when the filter passband and input signal frequency are mismatched.

5.3 DTMF Decode Function: `dtmfrun.m`

The DTMF decoding function, `dtmfrun` must use information from `dtmfscore` to determine which key was pressed based on an input DTMF tone. The skeleton of this function in Fig. 7 includes the help comments.

```
function keys = dtmfrun(xx,L,fs)
%DTMFRUN    keys = dtmfrun(xx,L,fs)
%    returns the list of key numbers found in xx.
%     xx = DTMF waveform
%      L = filter length
%     fs = sampling freq
%
freqs = [697,770,852,941,1209,1336,1477,1633];
hh = dtmfdesign( freqs,L,fs );
%   hh = L by 8 MATRIX of all the filters. Each column contains the
%         impulse response of one BPF (bandpass filter)
%
[nstart,nstop] = dtmfcut(xx,fs);   %<--Find the tone bursts
keys = [];
for kk=1:length(nstart)
   x_seg = xx(nstart(kk):nstop(kk));    %<--Extract one DTMF tone
....   %<=======================================FILL IN THE CODE HERE
end
```

Figure 7: Skeleton of `dtmfrun.m`. Complete the `for` loop in this function with additional lines of code.

The function `dtmfrun` works as follows: first, it designs the eight bandpass filters that are needed, then it breaks the input signal down into individual segments. For each segment, it will have to call the user-written `dtmfscore` function 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.

The input signal to the `dtmfscore` function must be a short segment from the DTMF signal. The task of breaking up the signal so that each segment corresponds to one key is done with the `dtmfcut` function which is called from `dtmfrun`. The score returned from `dtmfscore` *must be* either a 1 or a 0 for each frequency. Then the decoding works as follows: If exactly one row frequency and one column frequency are scored as 1's, then an unique key is identified and the decoding is probably successful. In this case, you can determine the key by using the row and column index. It is possible that there might be an error in scoring if too many or too few frequencies are scored as 1's. In this case, you should return an error indicator (perhaps by setting the key equal to -1). There are several ways to write the `dtmfrun` function, but you should avoid excessive use of "`if`" statements to test all 12 cases. Hint: use MATLAB's logicals (e.g., `help find`) to implement the tests in a few statements.

5.3.1 Testing

Once you get your system working there should be no errors with a large value of L, but when you try to reduce the filter length, the error indicator (key equal to -1) would tell you that the filter length is getting

too small. Run tests to find the minimum value for L that gives reliable operation. In you lab report, describe how you tested the system to get this minimum value for L.

5.4 Telephone Numbers

The functions `dtmfdial.m` and `dtmfrun.m` can be used to test the entire DTMF system as shown in Fig. 8. You could also use random digits (e.g., `ceil(15.9*rand(1,22)+0.09)`) in place of `1:16` in `dtmfdial`. For the `dtmfrun` function to work correctly, all the M-files must be on the MATLAB path. It

```
>>fs = 11025;  %<--use this sampling rate in all functions
>>xx = dtmfdial( 1:16, fs );
>>soundsc(xx, fs)
>>L = 201;     %<--overkill, this filter length is way too long
>>dtmfrun(xx, L, fs)
ans =
    1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16
```

Figure 8: Testing the DTMF system.

is also essential to have short pauses in between the tone pairs so that `dtmfcut` can parse out the individual signal segments.

If you are presenting this project in a lab report, demonstrate a working version of your programs by running it on the following phone number:

`404AC7365D`

In addition, make a spectrogram of the signal from `dtmfdial` to illustrate the presence of the dual tones.

Lab #8
ECE-2025
Spring-2001
INSTRUCTOR VERIFICATION PAGE

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or professors might ask. Turn this page in at the end of your lab period.

Name: _____ Date of Lab: _____

Part 3.1: Explain the row-column indexing into frequency table: Write the table below, and then write down the playing order of the frequencies. Determine the total length of the `soundsc()` signal in seconds.

Verified:_____ Date/Time:_____

Part 3.3: Complete the dialing function `dtmfdial.m`:

Verified:_____ Date/Time:_____