GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2003
Lab #9: Everyday Sinusoidal Signals

Date: 18–24 Mar 2003

******* Lab #9 will be graded out of 150 points. *******

You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section before your assigned lab time. You **MUST** complete the online Pre-Post-Lab exercise on Web-CT at the beginning of your scheduled lab session. You can use MATLAB and also consult your lab report or any notes you might have, but you cannot discuss the exercises with any other students. You will have approximately 20 minutes at the beginning of your lab session to complete the online Pre-Post-Lab exercise. The Pre-Post-Lab exercise for this lab includes some questions about concepts from the previous Lab report as well as questions on the Pre-Lab section of this lab.

The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. After completing the warm-up section, turn in the verification sheet to your TA.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports but the submitted work should be original and it should be your own work.

FORMAL Lab Report: You must write a formal lab report that describes your system for DTMF decoding (Section 4). The report is due during the week of 25-Mar to 31-Mar.

---

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.[1] 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 Background: Telephone Touch Tone Dialing

Telephone touch-tone[2] pads generate *dual tone multiple frequency* (DTMF) signals to dial a telephone. When any key is pressed, the sinusoids of the corresponding row and column frequencies (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 at 770 Hz and 1336 Hz together.

The frequencies in Fig. 1 were chosen (by the design engineers) to avoid harmonics. 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.[3] This makes it easier to detect exactly which tones are present in the dialed signal in the presence of non-linear line distortions. [4]

[1] If you do not have the function `freqz.m`, there is a substitute called `freekz.m` in the *SP-First* toolbox.

[2] Touch Tone is a registered trademark

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

[4] 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 dis-

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

Figure 1: DTMF encoding table for Touch Tone dialing. When any key is pressed the tones of the corresponding column and row are generated and summed.

1.2 DTMF Decoding

There are several steps to decoding a DTMF signal:

1. Filter the signal to extract the possible frequency components. Bandpass filters can be used to isolate the sinusoidal components.

2. Determine the short time intervals where *distinct* keys have been pressed. Gaps between separate key presses must be detected, and then a starting and stopping time can be found for each time interval.

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

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 seven bandpass filters—each one passing only one of the seven possible DTMF frequencies. The input signal for all the filters is the same DTMF signal.

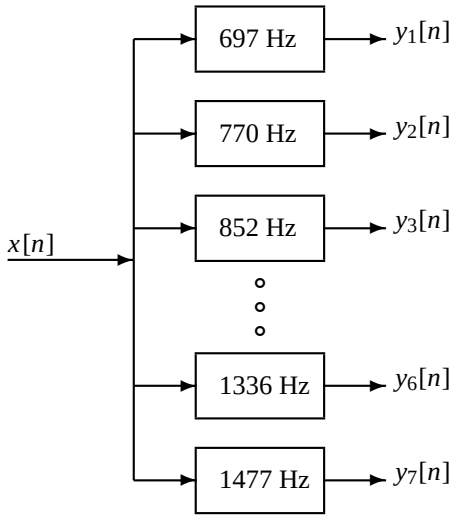

Figure 2: Filter bank consisting of bandpass filters (BPFs) that pass frequencies corresponding to the seven DTMF component frequencies listed in Fig. 1. The number is each box is the *center frequency* of the passband of the BPF.

cussion 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.utexas.edu/academic/otl/SpecSheets/DTMFdetection.html`.

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 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 in Section 4.

2 Pre-Lab

2.1 Bandpass Filter Design

You will need a bandpass filter design function for this lab. In a previous lab, you wrote an M-file called `myBPF.m` that will create the impulse response of an FIR filter whose frequency response looks like Fig. 3.

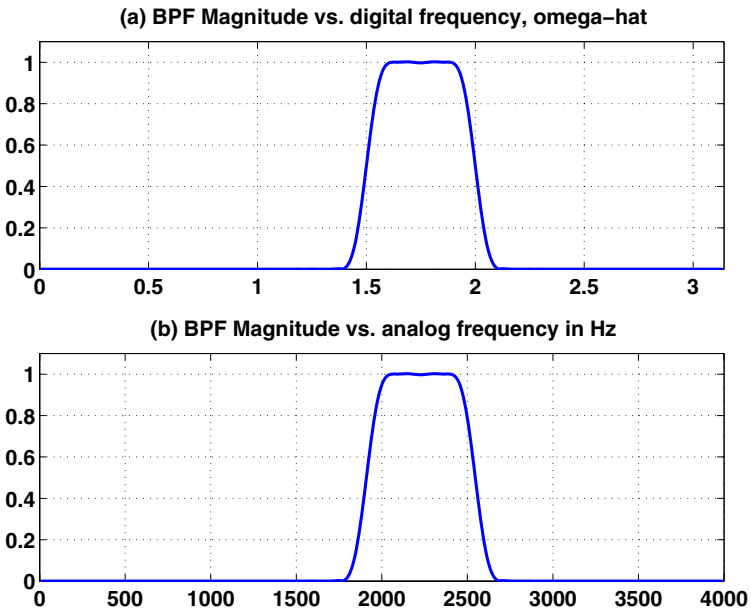

Figure 3: Frequency response of a length-101 bandpass filter (BPFs) created via the MATLAB call `myBPF([1.5,2],100)`. (a) $\left|H_B(e^{j\hat{\omega}})\right|$ plotted versus $\hat{\omega}$, and (b) the same frequency response versus analog frequency f (Hz), assuming that $f_s = 8000$ Hz.

(a) Write a few lines of MATLAB code to make the plot in Fig. 3(a).

(b) Modify the code in the previous part to change the frequency axis to get the plot in Fig. 3(b). Recall that the relationship between analog frequency and digital frequency is $\hat{\omega} = 2\pi(f/f_s)$.

2.2 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 could be 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.

2.2.1 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 signal length is known ahead of time.

2.2.2 Encoding from a Table

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?

```
ftable = [1;2;3;4;5]*[80,110]
fs = 8000;
xx = [ ];
disp('--- Here we go through the Loop ---')
keys = rem(2:11,10) + 1;
for ii = 1:length(keys)
   kk = keys(ii);
   xx = [xx,zeros(1,500)];
   krow = ceil(kk/2);
   kcol = rem(kk-1,2) +1;
 xx = [xx, cos(2*pi*ftable(krow,kcol)*(0:1499)/fs) ];
end
soundsc(xx,fs);
```

2.3 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 7-point averager, created from

```
HH = freqz((1/7)*ones(1,7),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/7*[-3,-2,-1,1,2,3],0.3*ones(1,6),'r.'), hold off
```

3 Warm-up: DTMF Synthesis

3.1 DTMF Dial Function

Write a function, `dtmfdial.m`, to implement a DTMF dialer based on the frequency table defined in Fig. 1. A skeleton of `dtmfdial.m` is given in Fig. 4.

```
function xx = dtmfdial(keyNames,fs)
%DTMFDIAL  Create a signal vector of tones that 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';
    '4','5','6';
    '7','8','9';
    '*','0','#'];
dtmf.colTones = ones(4,1)*[1209,1336,1477];
dtmf.rowTones = [697;770;852;941]*ones(1,3);
```

Figure 4: Skeleton of `dtmfdial.m`, a DTMF phone dialer. Complete this function more code.

In this warm-up, you must complete the dialing code so that it implements the following:

1. The input to the function is a vector of characters, each one being equal to one of the key names on the telephone. The MATLAB structure called `dtmf` contains the key names in the field `dtmf.keys` which is a 4×3 matrix that corresponds exactly to the keyboard layout in Fig. 1.

2. The output should be a vector of samples (at $f_s = 8000$ Hz) containing the DTMF tones, one tone pair per key. Remember that each DTMF signal is the sum of a pair of (equal amplitude) sinusoidal signals. The duration of each tone pair should be exactly 0.25 sec., and a gap of silence, about 0.10 sec. long, should separate the DTMF tone pairs. These times can be declared as fixed code in `dtmfdial.` (You do not need to declare the durations as variables in your function.)

3. The frequency information is given as two 4×3 matrices (`dtmf.colTones` and `dtmf.rowTones`): one contains the column frequencies, the other has the row frequencies. You can translate a key such as the **6** key into the correct location in these 4×3 matrices by using MATLAB's `find` function. For example, the **6** key is in row 2 and column 3, so we would generate sinusoids with frequencies equal to `dtmf.colTones(2,3)` and `dtmf.rowTones(2,3)`.

   To convert any key name to its corresponding row-column indices, consider the following example:

   ```
   [ii,jj] = find('3'==dtmf.keys)
   ```

   Also, consult the MATLAB code in Section 2.2 above and modify it for the 4×3 tables in `dtmfdial.m`.

4. You should implement error checking so that an illegitimate key name is rejected.

Your function should create the appropriate tone sequence to dial an arbitrary phone number. When played through a speaker into a telephone handset, the output of your function will be able to dial the phone. You could use `specgram` to check your work.[5]

Instructor Verification (separate page)

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

3.2 Bandpass Filter Design

In a previous lab, you developed a MATLAB function that designed lowpass, highpass and bandpass filters according to the "Hamming-sinc"formula:

$$h[n] = (0.54 - 0.46\cos(2\pi n/M))\,\frac{\sin(\hat{\omega}_c(n - M/2))}{\pi(n - M/2)} \qquad \text{for } n = 0, 1, 2, \ldots, M \tag{1}$$

where M is the filter order, which must be an even integer. The first term in $h[n]$ is called a Hamming window; the second term is a "sinc function." The design parameter $\hat{\omega}_c$ is called the ***cutoff frequency*** of the filter because it determines where the passband and stopband regions of the frequency response will be located. In the following parts, you should use your previously written BPF design function to create filters similar to what will be needed in the DTMF decoder.

Filter Specifications: Generate a bandpass filter with a passband from 2000 Hz to 2500 Hz when the sampling rate is $f_s = 8000$ Hz. Furthermore, make the two stopbands of the filter be $0 \leq f \leq 1500$ Hz and $3000 \leq f \leq 4000$ Hz.

(a) In order to carry this out, it is necessary to convert the passband and stopband edges into values for $\hat{\omega}$. Determine all those values.

(b) Make the filter length (L) equal to 101, and design a BPF with the correct passband (but don't worry about the stopband). Plot the frequency response (magnitude) of the resulting BPF, and verify that it has the correct passband region. Juggle the values of the cutoff frequencies to get the exact passband width.
Reminder: The *passband* of the BPF filter is defined by the region of the frequency response where $|H(e^{j\hat{\omega}})|$ is close to its maximum value of one. In this case, the passband width is defined as the length of the frequency region where $|H(e^{j\hat{\omega}})|$ is greater than 0.99 and less than 1.01.
Note: you could use MATLAB's `find` function to locate those frequencies where the magnitude satisfies $\big||H(e^{j\hat{\omega}})| - 1\big| \leq 0.01$.

(c) From the plot of the frequency response (magnitude) of the BPF in the previous part, measure the actual stopband locations.
Reminder: The *stopband* of the BPF filter is defined by the region of the frequency response where $|H(e^{j\hat{\omega}})|$ is close to zero. In this case, we have defined the stopband as the region where $|H(e^{j\hat{\omega}})|$ is less than 0.01.

Instructor Verification (separate page)

(d) Since the sampling rate is $f_s = 8000$ Hz, the frequency response of the digital BPF can be plotted versus analog frequency. Make this plot and show that the passband and stopband are at the correct frequency locations for this bandpass filter.

Instructor Verification (separate page)

(e) Reduce the filter length, and juggle the cutoff frequencies to find the minimum order FIR filter that will meet the specifications given above.

4 Lab: DTMF Decoding

A DTMF decoding system processes a signal that is a sequence of sounds, each one being the sum of two sinusoidal components chosen from the fixed set of possible DTMF frequencies. The DTMF decoding system needs two modules: a set of bandpass filters (BPFs) to isolate individual frequency components, and a detector to determine whether or not a given component is present. The number of BPFs is equal to the number of possible DTMF frequencies. The detector must compare all the BPF outputs and determine which two frequencies are the most likely ones to be contained in the DTMF tone. In a practical system where noise and interference are also present, this detection process is a crucial part of the system design. We will initially work with noise-free signals to understand the basic functionality of the decoding system, and then test on some signals that contain noise.

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

4.1 Narrow Bandpass Filter Design: `dtmfBPF.m`

The FIR filters used in the filter bank (Fig. 2) will be the "Hamming-sinc" filters used in the Warm-up. Since ***very narrow*** bandpass filters are need for this system, one important constraint must be obeyed. In the design of a "Hamming-sinc" lowpass filter the parameter $\hat{\omega}_c$ cannot be made arbitrarily small. The cutoff frequency $\hat{\omega}_c$ must be *greater than* $3.5\pi/L$, where L is the length of the filter. If this constraint is not obeyed, the frequency response (magnitude) will not reach the value of one in the passband. For bandpass filter design, this means that the "requested passband cutoff frequencies" must be separated by at least $7\pi/L$.

```
function  hh = dtmfBPF(fcent, L, fs)
%DTMFBPF
%    hh = dtmfBPF(fcent, L, fs)
%      returns a matrix (L by length(fcent)) where each column contains
%      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
%
%   Each BPF must be scaled so that its frequency response has a
%    passband within 1% of one; and stopbands less than 0.01.
```

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

(a) Complete the M-file `dtmfBPF.m` described in Fig. 5. This function should produce all seven bandpass filters needed for the DTMF filter bank system. The filter specs are given in the next part. Store the filters in the columns of the matrix `hh` whose size is $L \times 7$.

(b) *Filter Specifications:* For each of the seven BPFs, choose L so that only one frequency lies within the passband of the BPF and all other DTMF frequencies lie in the stopband. The bandpass filters should have a maximum frequency response value within 1% of one in the passband. You should run some experiments to learn how to choose L, the length of the filters.

(c) Generate the seven bandpass filters with $L = 501$ and $f_s = 8000$. 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). Use a very dense grid for $\hat{\omega} \in [-\pi, \pi]$; at least 5000 points along the frequency axis. Indicate the locations of each of the seven DTMF frequencies (697, 770, 852, 941, 1209, 1336, and 1477 Hz) on

this plot to illustrate whether or not the passbands are narrow enough to separate the DTMF frequency components, i.e., convert f to $\hat{\omega}$.
Hint: use the `hold` command and markers to denote the DTMF frequencies, similar to what you did in the warm-up.

(d) Now try to determine empirically the minimum length L so that the frequency response will satisfy the specifications on passband width and stopband rejection given above. Since these specifications are very stringent, the filter length will be quite long (in the hundreds).

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. Explain how each filter's passband is narrow enough to pass only one frequency component while rejecting others that are in the stopband. Since the same value of L is used for all the filters, which filter drives the problem? In other words, for which center frequency is it hardest to meet the specifications for the chosen value of L?

4.2 DTMF Detection Function: `dtmfdetect.m`

The second module 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 *detection* function that picks the most likely possibilities.

(a) Complete the `dtmfdetect` function based on the skeleton given in Fig. 6. The input signal matrix `yy` to the `dtmfdetect` function should contain the output of all the BPFs. The function will make the decisions by looking at the BPF filter outputs. The task of breaking up the signal so that the boundaries of the individual key presses are known is done by the function `dtmfseg` prior to calling `dtmfdetect`.

```
function  dkeys = dtmfdetect(yy, nstart, nstop)
%DTMFDETECT
%  usage:     dkeys = dtmfdetect(yy, nstart, nstop)
%  returns list of keys based on the max amplitude of the filtered output
%     yy = all filtered outputs from the 7 bandpass filter
%  nstart = vector of starting indices
%   nstop = vector of ending indices
%
%   The signal detection is done by finding the two largest outputs
%   within each segment
%     dkeys = detected keys
%
```

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

(b) Use the following rule for detection: find the maximum signal amplitude $\max_n |y_i[n]|$) in each time segment. The signal $y_i[n]$ is the output of the i-th BPF. Since you need one row and one column to select the key, you should find the maximum of two groups.

(c) When debugging your program it might be useful to plot the outputs of each BPF that the output $y[n]$ is either 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.

The function `dtmfdetect` does the actual decoding in the following manner: for each time segment, exactly one row frequency and one column frequency are selected as the largest so that a single key is identified. There is a remote possibility that there might be a detection error if several frequencies have the same peak amplitude. In this case, you should return an error indicator (perhaps by setting the key equal to `Z`). The easiest way to write the `dtmfdetect` function is to use MATLAB's `max` function to implement the logical tests.

4.3 DTMF Processing Function: `dtmfprocess.m`

The DTMF decoding function, `dtmfprocess` run the entire processing chain. It does the filtering and then calls `dtmfdetect` to determine the sequence of keys that were pressed. The skeleton of this function in Fig. 7 includes the help comments.

```
function keys = dtmfprocess(xx,L,fs)
%DTMFPRCESS   keys = dtmfprocess(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';
   '4','5','6';
   '7','8','9';
   '*','0','#'];
dtmf.colTones = ones(4,1)*[1209,1336,1477];
dtmf.rowTones = [697;770;852;941]*ones(1,3);
center_freqs = .... %<============================FILL IN THE CODE HERE
hh = dtmfBPF( center_freqs,L,fs );
%   hh = L by 7 MATRIX of all the filters. Each column contains the
%         impulse response of one BPF (bandpass filter)
%
[nstart,nstop] = dtmfseg(xx,fs);   %<--Find the tone bursts
....   %<======== Filter the signal thru the BPFs
....   %<======== Modify nstart and nstop for filter delay
keys = dtmfdetect(xxfiltered,nstart,nstop);    %<== Do the detection
```

Figure 7: Skeleton of `dtmfprocess.m`. Complete the `for` loop in this function with more code.

The function `dtmfprocess` works as follows: first, it designs the seven bandpass filters that are needed, then it processes the signal through the seven BPFs. The implementation of the FIR bandpass filters is done with the `conv` function, or with `firfilt`. The running time of the convolution function is proportional to the filter length L. Therefore, the long filter length L is fighting two competing constraints: L is very large so that the passband of each BPF is narrow enough to isolate individual frequency components, but its length makes the system run slowly. The task of finding the individual time intervals for each key can be done with the `dtmfseg` function which should be called from `dtmfprocess`. Finally it calls the user-written `dtmfdetect` function to determine the list of decoded keys.

4.4 Testing Telephone Numbers

The functions `dtmfdial.m` and `dtmfprocess.m` can be used to test the entire DTMF system as shown in Fig. 8. You could use standard phone numbers, or random digits (e.g., `char('0'+9.99*rand(1,11))`) in `dtmfdial`. For the `dtmfprocess` function to work correctly, all the M-files must be on the MATLAB

```
>>fs = 8000;  %<--use this sampling rate in all functions
>>tk = ['*','#','0','1','2','3','4','5','6','7','8','9'];
>>xx = dtmfdial( tk, fs );
>>soundsc(xx, fs)
>>dtmfprocess(xx, L, fs)   %<--use your value of L
ans =
    *#0123456789
```

Figure 8: Testing the complete DTMF system.

path. It is also essential to have short pauses of about 0.1 secs. in between the tone pairs so that `dtmfseg` will not make errors when it parses 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."

`704*1328956#`

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

4.5 Testing with Noisy Signals

The filters designed for the DTMF system in this lab are "overkill." They are very long FIR filters with extremely narrow passbands. It is actually possible to make the DTMF system work with much shorter filters, *if the signals are noise-free.* However, we can imagine the following scenario where decoding in the presence of noise might be a desirable activity. Suppose that a clandestine recording of a phone call had been made with a microphone that was not very close to the telephone handset. Then the DTMF tones might be hard to hear because the background noise is relatively loud. For example, suppose an air conditioner is running close to the microphone and the telephone is a speaker phone sitting across the room.

Two test signals are available for testing these noisy conditions:

(a) A DTMF signal plus additive noise can be found in the variable `xzz1` in the file `lab9dat.mat`. Process this signal to determine the phone number dialed. A similar test will be run when your code is evaluated in the next lab.
Note: it would be interesting to listen to the signal an hear the noise that it interfering with the DTMF signal.

(b) The second noisy signal can be found in the variable `xzz2` in the file `lab9dat.mat`. In this case the noise level is higher, but it should still be possible to decode the signal. Run the `dtmfprocess` M-file to find the phone number. In addition, make some spectrograms of the input and output signals in order to explain why the processing is successful. As before, listen to the signal to hear the amount of noise.

4.6 Demo

When you submit your lab report, you must demonstrate your work to your TA. Have your code and files ready for the demo. You should call `dtmfprocess` for a signal `xx` provided by your TA. The output should be the decoded telephone number. The evaluation criteria is shown at the end of the verification sheet.

Lab #9
ECE-2025
Spring-2003
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: Complete the dialing function `dtmfdial.m`. Listen to a phone number.

Verified:_____ Date/Time:_____

Part 3.2(c): Measure the stopband locations of the length-101 FIR bandpass filter.

Verified:_____ Date/Time:_____

Part 3.2(d): Make a plot versus analog frequency. Determine the analog frequency components passed and stopped by the BPF when $f_s = 8000$ Hz. Give the range of frequencies.

Verified:_____ Date/Time:_____

DTMF Decoding Evaluation

Does the designed DTMF decoder decode the telephone numbers correctly for the following cases?

Value of L _____

Case 1 (no noise): All Numbers _____ Most _____ None _____

Case 2 (noisy): All Numbers _____ Most _____ None _____