



Lab 09: Encoding and Decoding Touch-Tone (DTMF) Signals

Pre-Lab and Warm-Up: You should read at least the Pre-Lab and Warm-up sections of this lab assignment and go over all exercises in the Pre-Lab section before going to your assigned lab session.

Verification: 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**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply demonstrate the step to the instructor. Turn in the completed verification sheet to your instructor when you leave the lab.

Lab Report: Write a lab report on Section 4 with graphs and explanations. Please **label** the axes of your plots and include a title for every plot. In order to keep track of plots, include each plot *inlined* within your report. If you are unsure about what is expected, ask the instructor who will grade your report.

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 to 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 Review

In a previous lab, you may have learned about L -Point Averaging Filters and/or Nulling Filters. Another very important FIR filter is known as the Band-Pass Filter (BPF). For the rest of the lab, you will learn how to design these filters and how to use them to do certain tasks for you. One practical example is the *dual tone multiple frequency* (DTMF) signals used to dial a telephone. Read the following **Background** section before coming to the lab to speed up the sign-off process in the lab.

1.2 Background: Telephone Touch Tone Dialing

Telephone touch-tone² 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.

¹If you do not have the function `freqz.m`, there is a substitute available in the DSP First toolbox called `freakz.m`.

²Touch Tone is a registered trademark

³More information can be found at: <http://www.genave.com/dtmf.htm>, or search for "DTMF" on the Internet.



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. Keys A-D (in the fourth column) are not implemented on commercial and household telephone sets, but are used in some military and other signaling applications.

1.3 DTMF Decoding

There are several steps to decoding a DTMF signal:

1. Divide the time 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.
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 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.

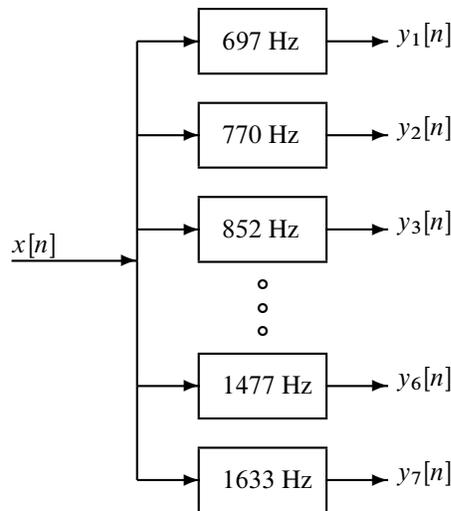


Figure 2: Filter bank consisting of bandpass filters (BPFs) which pass frequencies corresponding to the eight DTMF component frequencies listed in Fig. 1. The number in each box is the *center frequency* of the BPF.



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 Signal Concatenation

In a previous lab, a very long music signal was created by joining together many sinusoids. When two signals were played one after the other, the composite signal was 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`.

2.1.1 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 signal length is known ahead of time.

2.1.2 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?

```
fhtable = [1;2;3;4;5]*[80,110]
fs = 8000; xx = [ ];
keys = rem(3:12,10) + 1;
for ii = 1:length(keys)
    kk = keys(ii);
    xx = [xx,zeros(1,400)];
    krow = ceil(kk/2);
    kcol = rem(kk-1,2) + 1;
    xx = [xx, cos(2*pi*fhtable(krow,kcol)*(0:1199)/fs) ];
end
soundsc(xx,fs);
```



2.2 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

$$HH = \text{freqz}(\text{ones}(1, 5)/5, 1, ww)$$

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 Warm-up: DTMF Synthesis

3.1 DTMF Dial Function

```
function xx = dtmfodial(keyNames, fs)
%DTMFDIAL Create a signal vector of tones which will dial
%          a DTMF (Touch Tone) telephone system.
%
% usage:  xx = dtmfodial(keyNames, fs)
% keyNames = vector of characters containing valid key names
%          fs = sampling frequency
%  xx = signal vector that is the concatenation of DTMF tones.
%
dtmf.keys = ...
    ['1','2','3','A';
     '4','5','6','B';
     '7','8','9','C';
     '*','0','#','D'];
dtmf.colTones = ones(4,1)*[1209,1336,1477,1633];
dtmf.rowTones = [697;770;852;941]*ones(1,4);
```

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

Write a function, `dtmfodial.m`, to implement a DTMF dialer based on the frequency table defined in Fig. 1. A skeleton of `dtmfodial.m` is given in Fig. 3. 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 4×4 array that corresponds exactly to the keyboard layout in Fig. 1.
2. The output should be a vector of samples with sampling rate $f_s = 8000$ 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 0.20 sec., and a silence, about 0.05 sec. long, should separate the DTMF tone pairs. These times can be declared as fixed code in `dtmfodial`. (You do not need to make them variable in your function.)



3. The frequency information is given as two 4×4 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 4×4 matrices by using MATLAB's `find` function. For example, the key **6** 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 an 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 2.1 above and modify it for the 4×4 tables in `dtmf.dial.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 telephone handset, the output of your function will be able to dial the phone. You could use `specgram` to check your work.⁴

Instructor Verification (separate page)

3.2 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:

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

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 . Also, it is possible to choose β so that the maximum value of the frequency response magnitude will be one. 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 $\hat{\omega} = 0.2\pi$. Make the filter length (L) equal to 51. Figure out the value of β so that the maximum value of the frequency response magnitude will be one. Make a plot of the frequency response magnitude and phase.

Hint: use MATLAB's `freqz()` function to calculate these values.

- (b) 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. Typically, the passband width is defined as the length of the frequency region where $|H(e^{j\hat{\omega}})|$ is greater than $1/\sqrt{2} = 0.707$.

Note: you can use MATLAB's `find` function to locate those frequencies where the magnitude satisfies $|H(e^{j\hat{\omega}})| \geq 0.707$ (similar to Fig. 4).

Use the plot of the frequency response for the length-51 bandpass filter from part (a) to determine the passband width.

Instructor Verification (separate page)

- (c) If the sampling rate is $f_s = 8000$ Hz, determine the analog frequency components that will be passed by this bandpass filter. Use the passband width and also the center frequency of the BPF to make this calculation.

Instructor Verification (separate page)

⁴In MATLAB the demo called `phone` also shows the waveforms and spectra generated in a DTMF system.

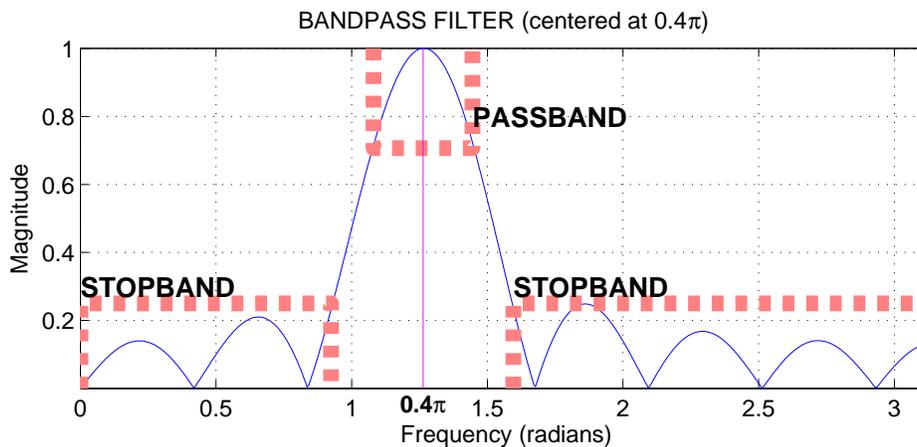


Figure 4: The frequency response of an FIR bandpass is shown with its passband and stopband regions.

4 Lab Exercises: 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: `dtmfscor`, `dtmfscor` and `dtmfdesign`. An additional M-file called `dtmfcut` is contained in the ZIP archive `DTMFdecodeLab.zip`. The main M-file should be named `dtmfscor.m`. It will call `dtmfdesign.m`, `dtmfcut.m`, and `dtmfscor.m`. The following sections discuss how to create or complete these functions.

4.1 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, as already shown in the Warm-up. In the section on useful filters in Chapter 9, 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:

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

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 MATLAB 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 .



```
function hh = dtmfdesign(fb, L, fs)
%DTMFDESIGN
%   hh = dtmfdesign(fb, L, fs)
%   returns a matrix (L by length(fb)) where each column contains
%   the impulse response of a BPF, one for each frequency in fb
%   fb = vector of center frequencies
%   L = length of FIR bandpass filters
%   fs = sampling freq
%
%   Each BPF must be scaled so that its frequency response has a
%   maximum magnitude equal to one.
```

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

- (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.
- (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 $L = 40$ and $L = 80$ 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) below.**
- (d) Generate the eight (scaled) bandpass filters with $L = 40$ and $f_s = 8000$. Plot the magnitude of the frequency responses all together on one plot (the range $0 \leq \hat{\omega} \leq \pi$ 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 $L = 80$ and $f_s = 8000$. 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 80 and length 40 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: For each of the eight BPFs, 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. 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, for which center frequency is it hardest to meet the specifications for the chosen value of L ?

4.2 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`.

```
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, 0.59
%   sc = 0 if max(|y[n]|) is less than 0.59
%
xx = xx*(2/max(abs(xx))); %--Scale the input 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 $\max_n |y_i[n]| \geq 0.59$; 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 1.0.⁵ Therefore the scoring threshold of 0.59 corresponds to a 59% 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 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.
- (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

⁵The 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.



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.

4.3 DTMF Decode Function: `dtmfrun.m`

The DTMF decoding function, `dtmfrun` must use information from `dtmfscor` 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 names found in xx.
%    keys = array of characters, i.e., the decoded key names
%    xx = DTMF waveform
%    L = filter length
%    fs = sampling freq
%
%
center_freqs = ... %<=====FILL IN THE CODE HERE
hh = dtmfdesign( center_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 beginning and end of 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 `dtmfscor` 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 `dtmfscor` 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 `dtmfscor` *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 16 cases. Hint: use MATLAB's logicals (e.g., `help find`) to implement the tests in a few statements.

4.4 Telephone Numbers

The functions `dtmfddial.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 `dtmfddial`. For the `dtmfrun` function to work correctly, all the M-files must be on the MATLAB path. It is also essential



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

Figure 8: Testing the DTMF system.

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:

407*89132#BADC

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

4.5 Demo

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



Lab: Encoding and Decoding Touch-Tone (DTMF) Signals

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 `dtmf_dial.m`:

Verified: _____

Date/Time: _____

Part 3.2(b): Measure the passband width of the BPF

Verified: _____

Date/Time: _____

Part 3.2(c): Determine the analog frequency components passed 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 values of L ?

$L = 111$ All Numbers _____ Most _____ None _____

$L = \text{Student's Length}$ All Numbers _____ Most _____ None _____