GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 2004
Lab #9: Everyday Sinusoidal Signals

Date: 26-Oct – 1-Nov 04

******* 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 you cannot give or receive written material or electronic files. Your 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 2–8 Nov.

---

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

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. Lowpass, highpass, or 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 doing frequency estimation during that time.

4. Determine which key was pressed, **0**–**9**, *****, or **#** by converting frequency pairs back into key names according to Fig. 1.

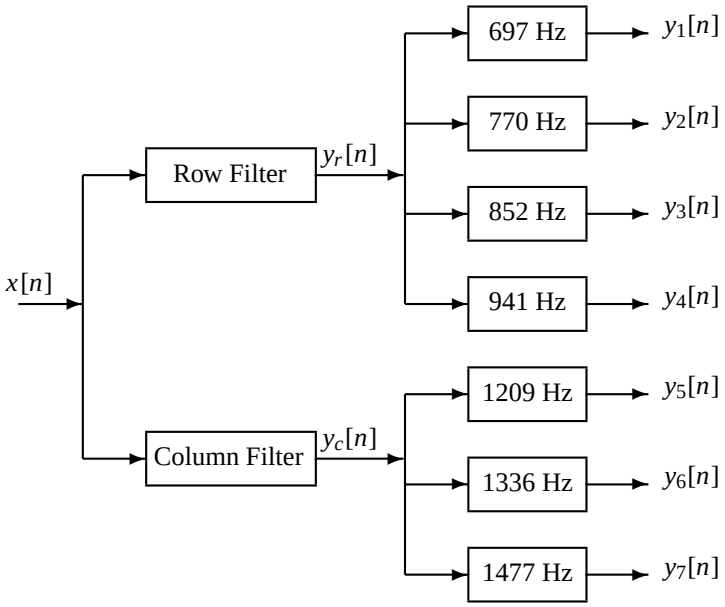

Figure 2: Filter bank consisting of two stages: row/column filters to separate the dual tones, followed by frequency estimators, which could be bandpass filters (BPFs), that determine the frequency corresponding to the individual sinusoidal components of the DTMF signal as listed in Fig. 1.

Here is how the system in Fig. 2 should work: When the input to the filter bank is a DTMF signal, the outputs from the row filter, $y_r[n]$, and the column filter, $y_c[n]$, both contain a single sinusoid. These signals

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

can then be processed further to determine the frequencies of the two sinusoids that make up the dual tone. It turns out that it is relatively simple to estimate the frequency of a signal that is *known to be just one sinusoid.* The MATLAB function `onefreq` is provided in this lab as a module that will perform frequency estimation of a single sinusoid. After we estimate the row and column frequencies, these frequencies are then used to find the key from the DTMF code table in Fig. 1. More discussion of the frequency estimation 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 a M-files called `VHLPF.m`, `VHHPF.m`, and `VHBPF.m`. The bandpass design function `VHBPF.m` will create the impulse response of an FIR filter whose frequency response looks like Fig. 3.

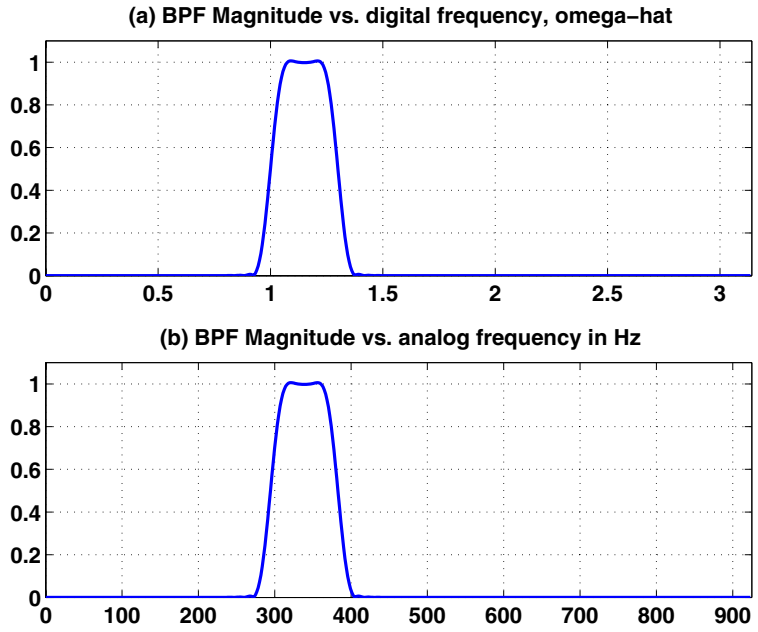

Figure 3: Frequency response of a length-141 bandpass filter (BPFs) created via the MATLAB call `VHBPF(1.0,1.3,100)`. (a) $\left|H_B(e^{j\hat{\omega}})\right|$ plotted versus $\hat{\omega}$, and (b) the same frequency response versus analog frequency f (Hz), assuming that $f_s = 1850$ Hz.

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

(b) Modify the code in part (a) 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 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 the vector `xx` every time a new sub-signal is appended via concatenation. If the length of `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 = 1850;
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(1,200)];
   krow = ceil(kk/2);
   kcol = rem(kk-1,2) +1;
   xx = [xx, cos(2*pi*ftable(krow,kcol)*(0:500)/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
```

2.4 Plotting Multiple Signals

The MATLAB function `strips` is a good way to plot several signals at once, e.g., the outputs from the row and column filters. Observe the plot(s) made by `strips(cos(2*pi*linspace(0,1,201)'*(4:10)));` In the *SP-First* toolbox, the function `striplot` can be used to plot multiple signals contained in the columns of a matrix via: `striplot(xx,fs,size(xx,1));`

3 Warm-up: DTMF Synthesis

3.1 Touch-Tone Dial Function

Write a function, `DTMFdial.m`, to implement a Touch-Tone 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.
%
TT.keys =  ['1','2','3';
            '4','5','6';
            '7','8','9';
            '*','0','#'];
TT.colTones = ones(4,1)*[1209,1336,1477];
TT.rowTones = [697;770;852;941]*ones(1,3);
```

Figure 4: Skeleton of `DTMFdial.m`, a Touch-Tone phone dialer. Complete this function by adding more lines of code to generate the dual-tone sinusoids.

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 `TT` contains the key names in the field `TT.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 $f_s = 4000$ Hz) containing the DTMF sinusoids—each key being the sum of two sinusoids. 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 0.2 sec., and a gap of silence, exactly 0.1 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 (`TT.colTones` and `TT.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 `TT.colTones(2,3)` and `TT.rowTones(2,3)`.

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

   ```
   [ii,jj] = find('3'==TT.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. In fact, 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 Touch-Tone 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 "VonHann-sinc" formula:

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

where M is the filter order, which *must be an even integer.* The first term is a "sinc function," and the second term in $h[n]$ is called a VonHann window. 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 LPF design function to create a filter similar to what will be needed in the Touch-Tone decoder.

Filter Specifications: Generate a lowpass filter with a passband from 0 Hz to 300 Hz when the sampling rate is $f_s = 1850$ Hz. Furthermore, make the stopband of the filter be $400 \leq f \leq \frac{1}{2} f_s$ 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 31, and design a LPF with the correct passband (but don't worry about the stopband yet). Plot the frequency response (magnitude) of the resulting LPF, and verify that it has the correct passband region. Juggle the value of the cutoff frequency to get the exact passband edge at 300 Hz.
Reminder: The *passband* of the LPF 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 0.9925 and less than 1.0075.
Note: you could use MATLAB's `find` function to locate those frequencies where the magnitude of $H(e^{j\hat{\omega}})$ satisfies $\big||H(e^{j\hat{\omega}})| - 1\big| \leq 0.0075$.

(c) From the plot of the frequency response (magnitude) of the LPF in the previous part, measure the actual stopband location.
Reminder: The *stopband* of the LPF 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 0.0075.

Instructor Verification (separate page)

(d) Since the sampling rate is $f_s = 1850$ Hz, the frequency response of the digital LPF can be plotted versus analog frequency. Make this plot and show that the passband is at the correct frequency location for this lowpass filter.

Instructor Verification (separate page)

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

4 Lab: Touch-Tone Decoding

A Touch-Tone 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 Touch-Tone decoding system needs two modules: a set of filters to isolate frequency components, and a frequency estimator to determine the value of each individual frequency component that is present in the signal. With reference to Fig. 2, two filters must be used to separate the row-frequency components from the column-frequency components. Then a frequency estimator will be used to determine which two frequencies are the

most likely ones to be contained in the Touch-Tone signal. In a practical system where noise and interference are also present, this frequency estimation 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 four M-files: `DTMFsys`, `DTMFfreqs`, `DTMFkeys` and `DTMFrowcol`. The main M-file should be named `DTMFsys.m`. It will call `DTMFrowcol.m`, `DTMFfreqs.m`, and `DTMFkeys.m`. The following sections discuss how to create, use or complete these functions.

4.1 Low Sampling Rate Causes Aliasing

Often, the design of a DSP system can be done at a very low sampling rate, and even though aliasing occurs, the system will still function correctly. In the case of the DTMF decoder, it has been observed by many engineers that the a sampling rate of

$$\boxed{f_s = 1850 \text{ Hz}}$$

would still permit the design of a simple system. Using this value for f_s will cause aliasing, but the aliasing *primarily affects the column-frequency components.*

(a) Using $f_s = 1850$ Hz, determine the locations in the $\hat{\omega}$-domain for the seven DTMF frequencies.

(b) Which analog frequency corresponds to the lowest $\hat{\omega}$ frequency?

4.2 Row-Column Filter Design: `DTMFrowcol.m`

The FIR filters used in the row-column separation (Fig. 2) will be the "VonHann-sinc" filters used in the Warm-up.

(a) *Filter Specifications:* Since only two filters are required, determine the types of filters needed, and also determine the specifications on each, i.e., give the desired passband cutoff frequency, stopband cutoff frequency.

(b) In the design of a "VonHann-sinc" filter the parameter $\hat{\omega}_c$ must be given. Determine the value that you will use for the "cutoff frequency" of the two filters.

(c) Generate the two filters with $M = 100$ and plot the magnitude of the frequency responses 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 $\hat{\omega} \in [-\pi, \pi]$, with at least 500 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 and stobands are sufficient to separate the row and column DTMF frequency components, i.e., convert f to $\hat{\omega}$ *including aliasing.*
Hint: use the `hold` command and markers to denote the DTMF frequencies, similar to what you did in the warm-up.

(d) Finally, carry out the design of the two filters by finding a *minimum* value for the filter order M, so that the passband and stopband obey the constraint that their values be within 0.0075 of the desired values of zeros and one. Use the same value of M for both filters.

Use the `zoom on` command to view the frequency response over the frequency domain where the DTMF frequencies lie. Comment on the selectivity of the filters, i.e., use the frequency response to explain how the two filters separate the row and column components.

4.3 Estimate the Frequencies: `DTMFfreqs.m`

The second module is frequency estimation—a process that works easily if we can guarantee that we have a signal that contains only one sinusoid. After the row-column separation filter the single sinusoid property is true for any isolated DTMF key-press. A MATLAB function called `onefreq.m` is provided to you for doing frequency estimation. The "help" for `onefreq.m` can be found in Fig. 5.

```
function omegahat = onefreq( xn )
%ONEFREQ   determine the freq when x[n] is one sinusoid
%
% usage:  omegahat = onefreq( xn )
%
%   xn = input signal (must contain at least 5 points)
%  omegahat = frequency when x[n] is Acos(omegahat*n+phi)
```

Figure 5: Help for the `onefreq.m` function. The input `xn` should be a vector of points from a portion of the single frequency sinusoid.

The frequency estimator in `onefreq` requires a minimum of five data points to produce a valid answer, but can take an `xn` vector of any length. If there is no noise or interference in the signal, short segments would be fine, but for noisy signals, longer data vectors produce better results.

(a) In order to validate that you know how to use `onefreq`, run the following test. Create a single DTMF tone for one key. Filter the tone through the row filter that you designed in Section 4.2, and then take 50 points from the middle of the output signal $y_r[n]$, and use those to call `onefreq`. Explain vai calculations that you obtained the correct frequency from `onefreq`.

(b) Repeat the experiment of the previous part making one change: call `onefreq` repeatedly with 5-point data vectors. In other words, if the output of the row filter is called `yrow` and its length is 200, then call `onefreq` 40 times with data vectors such as `yrow(1:5)`, `yrow(6:10)`, `yrow(11:55)`, etc. Plot the results to convince yourself that the frequency estimate is (more or less) constant, except and the beginning and end of the tone.

(c) Comment on why the data points from the beginning and end of the filtered signal might provide poor frequency estimates.

(d) Now write a function `DTMFfreqs.m` that will process an entire signal by calling `onefreq` at some regular interval. Here is a template

```
function  omegahats = DTMFfreqs(yy, nskip)
%DTMFFREQS
%  usage:     omegahats = DTMFfreqs(yy, nskip)
%  returns vector of frequencies (between -pi and +pi)
%     yy = filtered signal that is guaranteed to contain only one sinusoid
%            put yyrow in the first colum, and yycol in the 2nd column
%  nskip = interval for doing the frequency estimation,
%          i.e., yy(1:nskip), yy(nskip+1:2*nskip), yy(2*nskip+1:3*nskip), etc.

for nn=1:nskip:end_of_data?
   omegahats(?,1) = onefreq( yy(nn:nn+nskip-1,1) ); %--signals are in
   omegahats(?,2) = onefreq( yy(nn:nn+nskip-1,2) ); %--the columns
end
```

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

4.4 Identify the Keys: `DTMFkeys.m`

The third module is decoding—a process that turns the frequency pairs into actual key names such as **'1'** or **'5'** or **'#'**, and so on. In order to make the signal detection an automated process, we need a function that does two things: (i) finds the beginning and end of tones, and (ii) translates frequencies into key names.

(a) Since there will be gaps in between the actual tones, it will be necessary to characterize how `DTMFfreqs` treats the gaps. Run the following test on `DTMFfreqs`. Generate two DTMF tones for exactly the same key (at $f_s = 1850$ Hz). Make the tone duration equal to 0.2 secs., and put 0.1 secs. of silence (all zeros) in between. Use a small value for `nskip` in `DTMFfreqs` and plot the resulting frequency estimates as a stem plot. Describe how the silence region is treated.

(b) Complete the `DTMFkeys` function based on the skeleton given in Fig. 7. The input matrix `omegahats` to the `DTMFkeys` function is a $P \times 2$ matrix containing the estimated row and column frequencies, where P is the number of frequency estimates. These are done at some regular time interval dictated by the choice of `nskip` in `DTMFfreqs`. The strategy should be to get many estimates for one tone, and then your code for `DTMFkeys` needs to look for runs of the same frequency, and for gaps where there is silence.
Note: there is an old function called `DTMFcut` that you might find from labs given in previous semesters. If you use that M-file to implement `DTMFkeys`, *you will receive no credit for the implementation.*

```
function  dkeys = DTMFkeys( TT, omegahats, nskip, fs )
%DTMFKEYS   returns list of keys based on the estimated frequencies
%
%  usage:     dkeys = DTMFkeys( TT, omegahats, nskip, fs )
%
%            TT = structure containing freqs and key names
%     omegahats = P by 2 matrix of estimated frequencies
%         nskip = value used in DTMFfreqs.m
%            fs = sampling frequency
%         dkeys = detected keys
%
%**** It will be necessary to figure out the beginning and end
%**** of the DTMF tones.  Look for runs of the same frequency,
%**** and gaps where there is silence.
```

Figure 7: Skeleton of the `DTMFkeys.m` function. Complete this function with additional lines of code.

(c) When debugging your program it might be useful to plot the outputs of the row and column filters to see where the tone bursts start and stop. Then you can line up the outputs from `DTMFfreqs` to see if your method for finding the beginnings and ends is working correctly.
Hint: use the `strips` function in MATLAB to plot the filter outputs on the same graph, or `striplot` in the *SP-First* toolbox.

The function `DTMFkeys` does the actual decoding in the following manner: for each time segment, exactly one row frequency and one column frequency are obtained. Then these are used to identify a single key. Recall that the column frequencies are aliased, so the frequency matching between estimates and the table in Fig. 1 must account for aliasing. There is a remote possibility that there might be a detection error if the estimated frequencies don't match the known DTMF frequencies. In this case, you should return an error indicator (perhaps by setting the key equal to `Z`).

4.5 The Overall Touch-Tone System: `DTMFsys.m`

The Touch-Tone system function, `DTMFsys` runs the entire processing chain. It does the filtering, calls `DTMFfreqs` to find the frequencies, and then calls `DTMFkeys` to determine the sequence of keys that were pressed. The skeleton of this function in Fig. 8 includes the help comments.

```
function keys = DTMFsys(xx,M,fs)
%DTMFSYS   keys = DTMFsys(xx,M,fs)
%    returns the list of key names found in xx.
%   keys = array of characters, i.e., the decoded key names
%     xx = Touch-Tone waveform
%      M = filter order
%     fs = sampling freq
%
TT.keys =  ['1','2','3';
            '4','5','6';
            '7','8','9';
            '*','0','#'];
TT.colTones = ones(4,1)*[1209,1336,1477];
TT.rowTones = [697;770;852;941]*ones(1,3);
band_edges = .... %<===========================FILL IN THE CODE HERE
[hrow,hcol] = DTMFrowcol( M, fs, band_edges );
%   hrow = (M+1) by 1 column vector of the row filter containing its
%          impulse response (i.e., filter coefficients)
%   hcol = (M+1) by 1 column vector of the column filter containing its
%          impulse response (i.e., filter coefficients)
%
yyrow = %<======== filter the input signal
yycol = %<======== filter again.  Make these signals column vectors

omegahats = DTMFfreqs( [yyrow(:),yycol(:)], nskip );

 keys = DTMFkeys( TT, omegahats, nskip, fs );
```

Figure 8: Skeleton of `DTMFsys.m`. Complete the filtering and the `for` loop in this function with more code. Modify the code as you see fit.

The function `DTMFsys` works as follows: first, it designs the row and column filters, then it processes the input signal with the `conv` function, or with `firfilt`. The output signals contain only one sinusoid, so the function `DTMFfreqs` calls `onefreq` to find that frequency, and also to find the beginning and end of the tones. The function `DTMFkeys` uses the frequencies to determine the key names.

4.6 Testing Telephone Numbers

The functions `DTMFdial.m` and `DTMFsys.m` can be used to test the entire Touch-Tone system as shown in Fig. 9. You could use random digits (e.g., `char('0'+9.99*rand(1,11))`), or standard phone numbers in `DTMFdial`. For the `DTMFsys` function to work correctly, all the M-files must be on the

```
>> fs = 1850;  %<--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)
>> DTMFsys(xx, M, fs)   %<--use your value of M, the filter order
ans =
    *#0123456789
```

Figure 9: Testing the complete Touch-Tone system.

MATLAB path. It is also essential to have short pauses of about 0.1 secs. in between the tone pairs so that `DTMFkeys` can find the beginning and ending times of 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."

```
404555#1132*7986
```

In addition, make two spectrograms of the signal from `DTMFdial`, one at $f_s = 4000$ Hz and the other at $f_s = 1850$ Hz, to illustrate the presence of the dual tones before and after aliasing.

4.7 Testing with Unknown Signals

Two test signals in the MAT file `lab09f04support.mat` are available for testing. Similar tests will be run when your code is evaluated in the next lab.

(a) An unknown Touch-Tone signal can be found in the variable `xxTT1`. The durations of the tones are all equal, and the gaps between tones are all the same. Furthermore, the gaps are longer than 80 millisecs., and the tones are at least 200 millisecs. in duration. Process this signal to determine the phone number dialed.

(b) A second Touch-Tone signal can be found in the variable `xxTT2`. This signal has variable length tones and gaps, and the gaps are longer than 80 millisecs., and the tones are at least 200 millisecs. in duration. Process this signal to determine the phone number dialed.
Note: you can listen to the signal or display its spectrogram to understand more about it.

4.7.1 Demo

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

Lab #9
ECE-2025
Fall-2004
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-31 FIR lowpass filter.

Verified:_____ Date/Time:_____

Part 3.2(d): Make a plot versus analog frequency. Determine the analog frequency components passed and stopped by the LPF when $f_s = 1850$ Hz.

Verified:_____ Date/Time:_____

Touch-Tone Decoding Evaluation

Does the designed Touch-Tone decoder get the correct telephone numbers for the following cases?

Value of M, the filter order: _____

Case 1 (equal durations): All Numbers _____ Most _____ None _____

Case 2 (variable durations): All Numbers _____ Most _____ None _____