GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

EE 2200 Winter 1999
Lab #8: Everyday Sinusoidal Signals

Date: week of 1 Mar 1999

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**.

The lab report for this lab will be FORMAL: discuss your results from section 4. Staple the **Instructor Verification** sheet to the end of your lab report.

The report will due during the week of **8-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.[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 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[2]

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

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

[1] 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 EE-2200 WebCT page.

[2] 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.

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

We will always use capital `H` 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 an interval of length 2π 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[3] Dialing

Telephone touch pads generate *dual tone multi 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
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.

The frequencies in Fig. 1 were chosen 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.[4] This makes it easier to detect exactly which tones are present in the dial signal in the presence of line distortions.

2.2 DTMF Decoding

There are several steps to decoding a DTMF signal:

1. Divide the signal into shorter time segments representing individual key presses.
2. Filter the signal 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.

[3] Touch Tone is a registered trademark
[4] More information can be found at: `http://arrow.cso.uiuc.edu/telecom/dtmf/dtmf.html`

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 which each pass only one of the seven possible DTMF frequencies. The input signal for all the filters is the same DTMF signal.

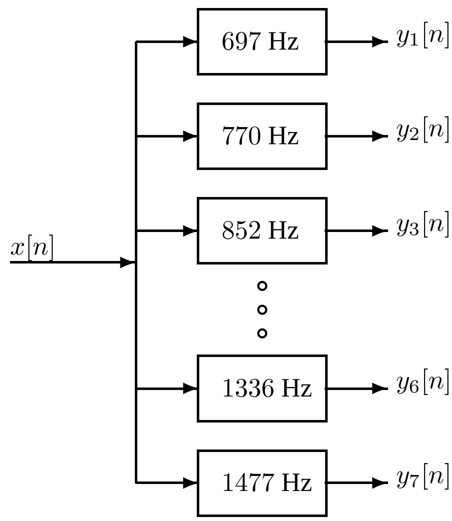

Figure 2: Filter bank consisting of bandpass filters which pass frequencies corresponding to the seven DTMF component frequencies listed in Fig. 1.

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 4.

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 sum of the lengths of the two signals `xx` and `xxnew`. A third signal could be added later on by doing another concatenation to `xx`.

Explain how the following program uses frequency information stored in a table to generate a long signal via concatenation. Determine the table entries and also 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]*[100,250];
fs = 8000;
xx = [ ];
for kk = 1:10
  xx = [xx,zeros(1,400)];
  k1 = ceil(kk/2);
  k2 = rem(kk-1,2) + 1;
  xx = [xx, cos(2*pi*ftable(k1,k2)/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 frequency response (magnitude) of the 5-point averager, created from `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
```

Instructor Verification (separate page)

3.3 DTMF Dial Function

Write a function, `dtmfdial`, to implement a DTMF dialer 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 it implements the following:

1. The input to the function is a vector of numbers that ranges between 1 and 12, with 1–9 corresponding to the digits, 10 corresponds to the ***** key, 11 is the **0** key, and 12 is the **#** key. Pay attention to the ordering of frequencies, because the **0** key is located in the middle of the bottom row.

2. The output should be a vector 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 hard-wired into `dtmfdial.` Remember that each DTMF signal is the sum of a pair of (equal amplitude) sinusoidal signals.

3. The frequency information is given as two 4×3 matrices: one contains the row frequencies, the other has the column frequencies. You can translate a digit such as **8** into the correct location in these 4×3 matrices by using quotients and remainders. For example, the digit **8** is in row 3 and column 2. If we divide 8 by 3, then the quotient is 2 and the remainder is 2. To get the correct row, we must round the quotient up to 3. You will have to generalize this idea to make the dialing work properly. Check out `help rem` and `help ceil` for some hints.

```
function dtmfsig = dtmfdial(nums,fs)
%DTMFDIAL  Create a vector of tones which will dial
%           a DTMF (Touch Tone) telephone system.
%
% usage:  dtmfsig = dtmfdial(nums)
%      nums = vector of numbers ranging from 1 to 12
%        fs = sampling frequency
%   dtmfsig = vector containing the corresponding tones.
%
tone_cols = ones(4,1)*[1209,1336,1477];
tone_rows = [697;770;852;941]*ones(1,3);
```

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

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 may use `specgram` to check your work.[5]

Instructor Verification (separate page)

4 Lab: DTMF Decoding

A DTMF decoding system needs two pieces: a bandpass filter (BPF) to isolate individual frequency components, and a detector to determine whether or not a given component is present. The detector must "score" each possibility 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. 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.

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. 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:

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

where L is the filter length, and f_s is the sample frequency. The constant β gives flexibility for scaling 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.

(b) Complete the M-file `dtmfdesign.m` which is described in Fig. 4. This function should produce all seven bandpass filters needed for the DTMF filter bank system. Store the filters in the columns of the matrix `hh`.

(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.

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

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

Figure 4: Skeleton of the `dtmfdesign.m` function. Complete this function with additional lines of code.

(d) Generate the seven (scaled) bandpass filters with $L = 25$ and $f_s = 11025$. Plot the magnitude of the frequency responses all together on one plot. Indicate the locations of each of the 7 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. Hint: use the `hold` and `stem()` commands.

(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 the frequency response where $|\mathcal{H}(\hat{\omega})|$ is close to one. Typically, the passband width is defined as the length of the frequency region where $|\mathcal{H}(\hat{\omega})|$ is greater than $1/\sqrt{2} = 0.707$.

The *stopband* of the BPF filter is defined by the region of the frequency response where $|\mathcal{H}(\hat{\omega})|$ is close to zero. In this case, it is reasonable to define the stopband as the region where $|\mathcal{H}(\hat{\omega})|$ is less than 0.2.

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 the filter's passband narrow enough so that only one frequency component lies in the passband and the others are in the stopband?

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. 5. 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.

(b) Use the following rule for scoring: the score equals one when $\max_n |y[n]| \geq 0.45$; otherwise, it is zero. The signal $y[n]$ is the output of one of the BPFs.

(c) Make sure that the input signal $x[n]$ is normalized to the range $[-1, +1]$ prior to filtering. With this scaling the two sinusoids that make up $x[n]$ should each have amplitudes of approximately 0.5.[6] Therefore the scoring threshold of 0.45 corresponds to a 90% level for detecting the presence of the sinusoid.

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

```
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 signal
%     hh = impulse response of ONE bandpass filter
%
%   The signal detection is done by filtering xx with a length-L
%   BPF, hh, and then finding the maximum amplitude of the output.
%   The score is either 1 or 0.
%      sc = 1 if max(|y[n]|) is greater than, or equal to, 0.45
%      sc = 0 if max(|y[n]|) is less than 0.45
%
xx = xx/max(abs(xx));   %---Scale x[n] to the range [-1,+1]
```

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

(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 $\mathcal{H}(\hat{\omega})$ 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.

(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 sinusoidal 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 `dtmfscore` to determine which key was pressed based on an input DTMF tone. The skeleton of this function in Fig. 6 includes the help comments. The function `dtmfrun` works as follows: first, it designs the seven 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). Once you get your system working there should be no errors, but when you try to reduce the filter length, the error indicator would tell you that the filter length is getting too small.

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.

4.4 Telephone Numbers

The functions `dtmfdial.m` and `dtmfrun.m` can be used to test the entire DTMF system as shown in Fig. 7. For the `dtmfrun` function to work correctly, all the M-files must be on the MATLAB path. It 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 several times with a variety of phone numbers.

```
function keys = dtmfrun(xx,L,fs)
%DTMFRUN    keys = dtmfrun(xx,L,fs)
%    returns the list of key numbers corresponding
%      to the DTMF waveform, xx.
%      L = filter length
%     fs = sampling freq
%
freqs = [697,770,852,941,1209,1336,1477];
hh = dtmfdesign( freqs,L,fs );
%   hh = 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 6: Skeleton of `dtmfrun.m`. Complete the `for` loop in this function with additional lines of code.

```
>>fs = 11025;  %<--use this sampling rate in all functions
>>xx = dtmfdial( 1:12, 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
```

Figure 7: Testing the DTMF system.

Lab #8
EE-2200
Winter-1999
INSTRUCTOR VERIFICATION PAGE

Staple this page to the end of your Lab Report.

Name: ____ Date of Lab: ____________

Part 3.2: Place overlay markers on a frequency response:

Verified:________________________ Date/Time:____________

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

Verified:________________________ Date/Time:____________