Laboratory # 6

Decoding DTMF: Filters in the Frequency Domain

6.1 Introduction

In Lab 5, you examined the behavior of several different filters. Some of the filters were "smoothing filters" that averaged the signal over many samples. Others were "sharpening" filters that accentuated transitions and edges. While it is very useful to understand the effects of these filters in the *time-domain* or (for images) the *spatial-domain*, it is often not easy to quantify these effects, especially when we are dealing with more complicated filters. Thus, just as we did with signals, we would like to understand the behavior of our filters in the *frequency-domain*.

Assuming that our filter is linear and time-invariant, we can talk about the filter having a *frequency response*. We derive the frequency response in the following way. We know that if we put a complex exponential signal into such a filter, the output will be a scaled and shifted complex exponential signal with the same frequency. The amount of scaling and time shifting, though, is dependent on the frequency of the input signal. If we send a complex exponential signals with some frequency through the filter, we can measure the scaling and time shifting of that signal. The collection of complex numbers which corresponds to this scaling and shifting for all possible frequencies is known as the filter's *frequency response*. The magnitude of the frequency response at a given frequency is the filter's *gain* at that frequency.

In this lab, we will be using the frequency response of filters to examine the problem solved by *telephone touch-tone dialing*. The problem is this: given a noisy audio channel (like a telephone connection), how can we reliably transmit and detect phone numbers? The solution, which was developed at AT&T, involves the transmission of a sum of sinusoids with particular frequencies. In order for this solution to be feasible, we must be able to easily decode the resulting signal to determine which numbers were dialed. We will see that we can do this easily by considering filters in the frequency domain.

6.2 Background

6.2.1 DTMF signals and Touch Tone™ Dialing

Whenever you hit a number on a telephone touch pad, a unique tone is generated. Each tone is actually a sum of two sinusoids, and the resulting signal is called a *dual-tone multifrequency* (or *DTMF*) signal. Table 6.1 shows the frequencies generated for each button. For instance, if the "6" button is pressed, the telephone will generate a signal which is the sum of a 1336 Hz and a 770 Hz sinusoid.

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

Table 6.1: DTMF encoding table for touch tone dialing. When any key is pressed, the tones of the corresponding row and column are generated.

We will call the set of all seven frequencies listed in this table the *DTMF frequencies*. These frequencies were chosen to minimize the effects of signal distortions. Notice that none of the DTMF frequencies is a multiple of another. We will see what happens when the signal is distorted and why this property is important.

Looking at a DTMF signal in the time domain does not tell us very much, but there is a common signal processing tool that we can use to view a more useful picture of the DTMF signal. The *spectrogram* is a tool that allows us to see the frequency properties of a signal as they change over time. The spectrogram works by taking multiple DFTs over small, overlapping segments[1] of a signal. The magnitudes of the resulting DFTs are then combined into a matrix and displayed as an image. Figure 6.1 shows the spectrogram of a DTMF signal. Time is shown along the x-axis and frequency along the y-axis. Note the bars, each of which represents a sinusoid of a particular frequency existing over some time period. At each time, there are two bars which indicate the presence of the two sinusoids that make up the DTMF tone. From this display, we can actually identify the number that has been dialed; you will be asked to do this in the lab assignment.

6.2.2 Decoding DTMF Signals

There a number of steps to perform when decoding DTMF signals. The first two steps allow us to determine the strength of the signal at each of the DTMF frequencies. We first employ a bank of bandpass filters with center frequencies at each of the DTMF frequencies. Then, we process the output of each bandpass filter to give us an indication of the strength of each filter's output. The third step is to "detect and decode." From the filter output strengths, we detect whether or not a DTMF signal is present. If it is not, we refrain from decoding the signal until a tone is detected. Otherwise, we select the two filters with the largest output strengths and use this information to determine which key was pressed. A block diagram of the DTMF decoder can be seen in Figure 6.2.

Step 1: Bandpass Filters

From Lab 2, you may recall that correlating two signals provides us with a measure of how similar those two signals are. Since convolution is just "correlation with a time reversal," we can use this same idea to design a filter that passes a given frequency. If our filter's impulse response "looks like" the signal we want to pass, we should get a large amplitude signal out; similarly, signals that are different will produce smaller output signals.

When performing DTMF decoding, we want filters that pass only one of the DTMF frequencies and reject all of the rest. We can make use of the correlation idea above to develop such a *bandpass filter*. We want our impulse response to be similar to a signal with the frequency that we wish to pass; this is the filter's *center frequency*. This means that for

[1]Note that each segment is some very small fraction of a second, and the segments usually overlap by 25-75%.

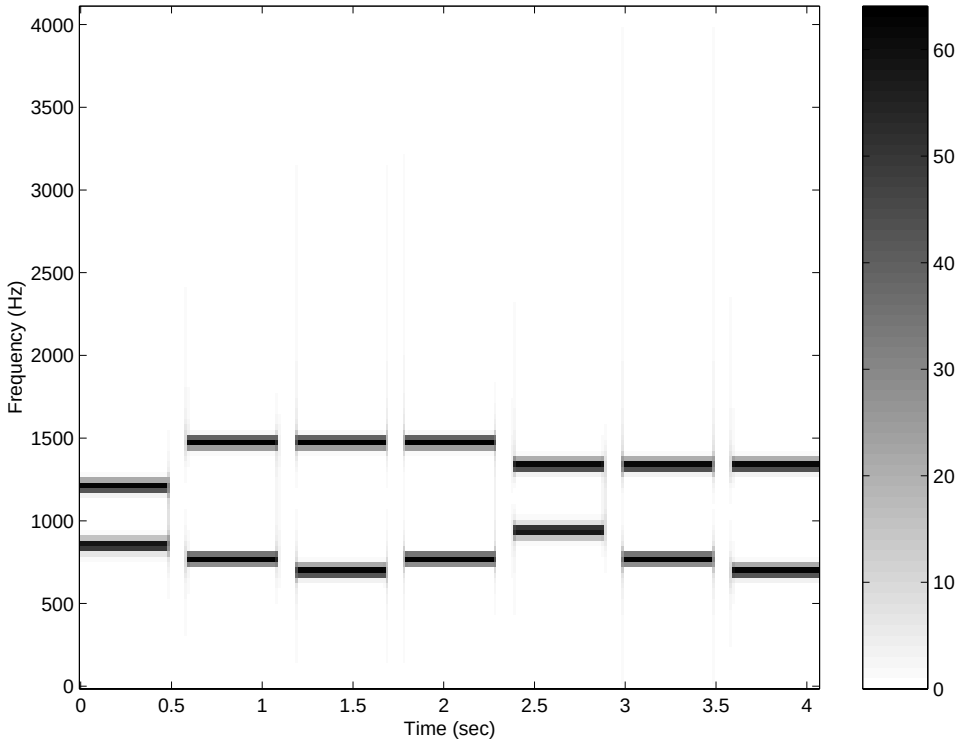

Figure 6.1: A spectrogram of a DTMF signal. Each horizontal bar indicates a sinusoid that exists over some time period.

a bandpass filter with center frequency f, we want our impulse response, h, to be equal to

$$h[k] = \begin{cases} \sin(2\pi f_c k/f_s) & 0 \leq k < L \\ 0 & else \end{cases} \tag{6.1}$$

From this equation, we have an FIR filter with order $L-1$. What should L be? For this filter, L is a design parameter. You may remember from Lab 2 that correlating over a long time produces better estimates of similarity. Thus, we should get better differentiation between passed frequencies and rejected frequencies if L is large. There is a tradeoff, though. The longer L is, the more computation that is required to perform the convolution. Thus for efficiency reasons we would like L to be as small as possible. More computation also equates to more expensive devices, so we prefer smaller L for reasons of device economy as well.

Because of the relatively small set of frequencies of concern in DTMF decoding, we will see that larger L do not necessarily produce better frequency differentiation. In order to judge how good a bandpass filter is at rejecting unwanted DTMF frequencies, we will define the *gain-ratio*, R. Given a filter with center frequency f_c and frequency response $\mathcal{H}$, the gain-ratio is

$$R = \frac{|\mathcal{H}(f_c)|}{\max|\mathcal{H}(\hat{f})|} \tag{6.2}$$

where $\hat{f}$ is in the set of DTMF frequencies and $\hat{f} \neq f_c$. In words, we define R to be the ratio of the filter's gain at its center frequency to the *next-highest* gain at one of the DTMF frequencies. Having a high gain-ratio is desirable, since it indicates that the filter is rejecting the other possible frequencies.

Note that since we will be comparing the outputs of a variety of bandpass filters, we also need to normalize each filter by the center frequency gain. Thus, we will need to record not

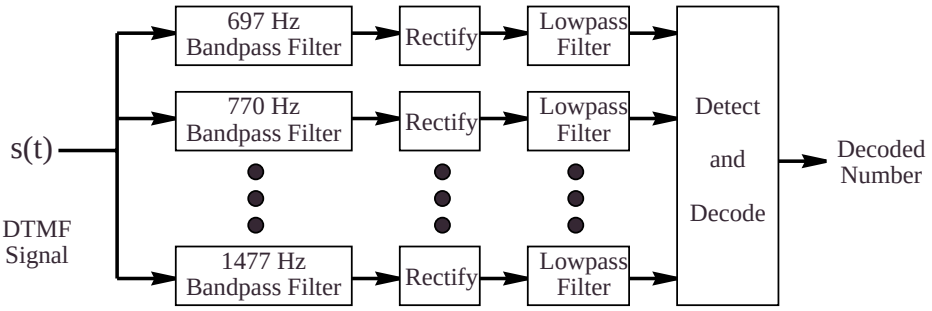

Figure 6.2: A block diagram of the DTMF decoder system. The input is a DTMF signal, and the output is a string of numbers corresponding to the original signal.

only the L that we select but also the center frequency gain. You will be directed to record and include these gains in the lab assignment.

Step 2: Determining filter output strengths

In order to measure the strength of the filter's output, we actually want to measure (or follow) the envelope of the filter outputs. To follow just the positive envelope of the signal, we first need to eliminate the negative portions of the signal. If we simply truncate all parts of the signal below zero, we have applied a *half-wave rectifier*. Alternately, we can simply take the absolute value of the signal, in which case we have applied a *full-wave rectifier*. It is possible to build rectifiers using diodes, and it turns out that half-wave rectifiers are easier to design. However, full-wave rectifiers are preferable, and they are no more difficult to implement in MATLAB. Thus, we will use full-wave rectifiers[2]. See Figure 6.3 to see the effects of these two types of rectifiers.

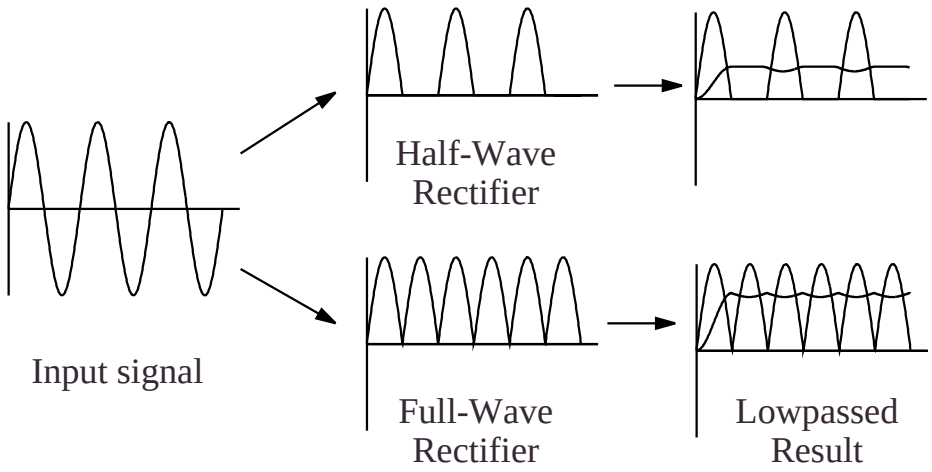

Figure 6.3: A comparison of half-wave and full-wave rectification. Notice that full-wave rectification allows us to achieve a higher output signal level after lowpass filtering.

If we now pass the rectified signal through a smoothing filter, the output will be a nearly constant signal whose value is a measure of the strength of the filter's input at the center

[2]Note that the names "full-wave rectifier" and "half-wave recitifer" come from the circuit implementation of these systems

frequency of the filter. Here, we will use a simple moving average filter to accomplish this smoothing. When designing this filter, there is a tradeoff between amount of smoothing and transient effects. If our filter's impulse response is not long enough, the output signal will still have significant variations. If it is too long, edge effects will dominate the output of the filter. Figure 6.3 shows the results of smoothing for half-wave and full-wave rectified signals.

Step 3: "Decode and Detect"

With the results of this processing, we can determine the keys pressed to generate the signal. If we have designed our filters properly, the outputs from two of the filters should be much stronger than the outputs from the others. Given the center frequencies of these two filters, we can perform a look-up on Table 6.1 to determine which key was pressed.

There are, however, a few additional details that must be considered. First, if we decode a key press for every sample of the input, then our system will be very vulnerable to small perturbations in the input signal. If we instead consider only every 100^{th} sample[3], we can make our system relatively insensitive to these variations.

Additionally, we need to be able to detect whether a DTMF tone is actually present. If it is not, we risk making an error in our decoding of the input signal. Here, we detect the presence of a DTMF tone by comparing the largest filter output strength to a threshold. If it is greater than the threshold, then the signal is detected and we decode a key press for that time. Otherwise, we record a "null" key press.

Finally, we need to combine identical key presses from successive detections. This makes our final decoding independent of how long a given key is pressed. By recording "no signal" detections as a null key press, we are able to distinguish between repetitions of a key press and a single, long key press.

6.2.3 Decoder Robustness

Whenever designing a communication system, like the DTMF coder/decoder described here, it is important to consider how the system behaves in the presence of undesirable effects. For instance, the telephone system could corrupt our DTMF signal with some amount of static. Under such conditions, how well would the decoder work? How much noise can the system tolerate? These are all questions about the *robustness* of the decoder system to noise. No system can work perfectly under less than ideal conditions, so it is important to understand when and how a system will fail. In the lab assignment, we will examine the robustness of this system under various types of distortion.

6.2.4 Sidenote: Searching Parameter Spaces

Quite frequently, you will find yourself in the position of searching for a "good value" for a particular parameter about which you have no other information. In these cases, there are some techniques that we can employ to speed the search. The basic idea is that we want to get "in the ballpark" before we worry about finding locally optimum solutions. To do this, we think about varying parameters over factors of 2 or factors of 10. Thus, you might try parameter values of 0.01, 0.1, 1, 10, and 100 to get a general notion of how the system responds to a parameter. Once we have done this, we can then isolate a smaller range over which to optimize. This prevents us from spending too much time searching aimlessly.

6.3 Laboratory Demonstrations

1. Dual tone multi-frequency signals in the real world

[3]Note that in this system, we are using a sampling rate of 8192 Hz.

2. Generating "synthetic" DTMF signals.
3. Bandpass filters
4. The DTMF decoder
5. The DTMF coder and clipping of the input signal
6. Noise and the DTMF decoder

6.4 Laboratory Assignment

1. The first thing we want is a function that produces a DTMF signal when given a sequence of numbers to be dialed – a "dialer" function. Download the function `dtmf_dial.m`. `dtmf_dial` is a mostly complete dialer function. You simply need to replace the two question marks by code that completes the function. The first missing line of code generates a DTMF tone for each number in the input and appends it to the output signal. The second line of code appends a short silence to the signal to separate adjacent DTMF tones.

   - [8] Complete the function and include the code in your lab report.
   - [3] Using your newly completed dialer function, execute the following command to create a DTMF signal and display it's spectrogram:

     ```
     signal = dtmf_dial([1 2 3 4 5 6 7 8 9 10 11 12],1);
     ```

     Include the resulting figure in your report.
   - [3] On the spectrogram of the signal, identify the time segment that corresponds to '9', and label the two frequency components that you can see with the appropriate frequency.
   - [4] What is the phone number that has been dialed in Figure 6.1?

2. As we have noted, a key part of the DTMF decoder is the bank of bandpass filters that is used to detect the presence of sinusoids at the DTMF frequencies. We have specified a general form for the bandpass filters, but we still need to design the specific filters. In this problem you will be identifying good design characteristics for these bandpass filters.

   (a) Using equation 6.1 with a sampling frequency $f_s = 8192$ Hz and $L = 50$, use MATLAB to create a vector containing the impulse response, `h`, of a 770 Hz bandpass filter.

   - [4] What is the command that you used to create this impulse response?
   - [2] Plot your impulse response.
   - [1] Describe the resulting impulse response.

   (b) When we talk about the response of a filter to a particular frequency, we can think about filtering a unit amplitude sinusoid with that frequency and measuring the amplitude and phase shift of the resulting signal. We can certainly do this in MATLAB, but there are simpler ways to find the frequency response of a filter in MATLAB. The command `freqz`, which is part of MATLAB's Signal Processing Toolbox, lets us find the frequency response. If our filter has an impulse response `h`, we can use `freqz` to calculate the frequency response at a set of discrete-time frequencies $\omega_0 \in [0, 2\pi)$. For instance, if we wish to calculate the frequency response of a filter at 100 Hz, 200 Hz, and 300 Hz, given a sampling frequency `fs`, we use the command

```
H = freqz(h,1,[100; 200; 300]*2*pi/fs);
```

(Note that `freqz` is a little strange; if we try asking for the frequency response at a single point, it will interpret your command differently. When calling `freqz` like this, make sure you request more than one discrete-time frequency). The result, `H`, is a vector of complex numbers which define the *gain* (`abs(H)`) and *phase-shift* (`angle(H)`) of the filter at the given frequencies.

- [4] Use `freqz` to calculate the frequency response of your 770 Hz bandpass filter at all seven of the DTMF frequencies[4]. Calculate the gain at each frequency, and include these numbers in your report.
- [2] From the frequency response of your filter at these frequencies, calculate the gain-ratio, R.
- [2] Do you think that this is a good gain-ratio for our bandpass filters?

(c) Download the file `dtmf_filt_char.m`. This function will allow us to quickly characterize our bandpass filters. Use the command

```
gain = dtmf_filt_char(50,770);
```

to calculate the gain of a 50-point (i.e., $L = 50$) bandpass filter with center frequency $f_c = 770$ Hz as defined by equation 6.1. The vector `gain` has the magnitude response to each of the DTMF frequencies, from the smallest to the largest.

- [1] Verify that the gains you calculated before were correct.
- [2] Include the plot that `dtmf_filt_char` produces in your report.
- [6] Increase L and examine the plots that result (you do not need to include these plots). What features of the filter's frequency response contribute to the gain ratio R? For what values of L do we achieve good gain ratios (i.e., greater than 10)?

(d) For each DTMF frequency, we want to find an optimal choice of L. Consider the following procedure. For each DTMF frequency, vary L until you find the smallest value of L for which the gain-ratio, R, is greater than 10. Then, without increasing L substantially, you should be able vary L to find a local maximum of R. (A *local maximum* of a function $f(n)$ is a point $f(n_0)$ that is greater than $f(n_1)$ for all n_1 in some neighborhood of n_0.)

You should automate the calculation of R, since you will need to do it repeatedly. Use MATLAB to calculate R (Hint: try using the `sort` command) for all L between 1 and 200. Consider using MATLAB to plot R as a function of L. You can save some computation time by setting the third parameter of `dtmf_filt_char` to zero to suppress plotting.

Perform this two-step optimization for a bandpass filter at each DTMF frequency.

- [10] Create a table in which you record the center frequency, the smallest L for which $R > 10$, the center frequency gain at this L, and the resulting value of R for each of your seven bandpass filters.
- [10] Create a second table in which you record same information for your optimized value of L.

3. Now we have designed the bank of bandpass filters that we need for the DTMF decoder. Download the file `dtmf_decode.m`. This function is a nearly complete implementation of the DTMF decoder system described earlier in this lab. There are several things that you need to add to the function.

[4] Remember that our system uses a sampling frequency of 8192 Hz

(a) First, you need to record your "optimized" values of L and the center frequency gains in the function. Find the line in `dtmf_decode` that has `L = ?`. Replace the question mark by a vector of your optimized values of L. They should be in order from smallest frequency to largest frequency. Do the same for the variable `G`, which contains the center frequency gains.

- [4] Make these modifications to the code.

(b) Next, we need to design a post-rectifier smoothing filter, `h_smooth`. Temporarily set both `h_smooth` and `threshold` equal to 1 and run `dtmf_decode` on the DTMF signal you generated in Problem 1. This function displays a figure containing the rectified and smoothed outputs for each bandpass filter. With `h_smooth` equal to 1, no smoothing is done and we only see the results of the rectifier in this figure. We will consider moving average filters of length n, as defined by the MATLAB command `h_smooth = ones(n,1)/n;`.

- [4] Examine the behavior of the smoothed signal with moving average filters of length 20, 200, and 2000. We want the smoothed output to be effectively constant during most of the duration of the DTMF tones, but we don't want to smooth so much that we might miss short DTMF tones or pauses between tones. Which filter gives us the best tradeoff between transient effects and smoothing?
- [1] Set `h_smooth` to be the filter you have just selected.

(c) Finally, we need to identify a good value for `threshold`. `threshold` determines when we detect the presence of a DTMF signal. `dtmf_decode` plots the threshold on its figure as a black dotted line. A signal is considered to be present within an averaging frame when the mean value of any of the smoothed output signals over the frame is greater than the threshold. We want the threshold to be smaller than the large amplitude signals during the steady-state portions of a DTMF signal, but larger than the signals during most of the transient portions.

- [5] By looking at the figure produced by `dtmf_decode`, what would be a reasonable threshold value? Why did you choose this value?
- [2] Set `threshold` to the value you have just selected, execute `dtmf_decode`, and include the resulting plot in your report. (Note: this plot can be black and white.)
- [2] `dtmf_decode` should output the same vector of "key presses" that was used to produce your signal. What "key presses" does the function produce? Do these match the ones used to generate the DTMF signal? If not, you've probably made a poor choice of threshold.

4. In the introduction to this lab, we indicated that we would be transmitting our DTMF signals over a noisy audio channel. So far, though, we have assumed that the decoder sees a perfect DTMF signal. In this problem, we will examine the effects of additive noise on the DTMF decoder.

(a) Download the file `dtmf_attack.m`. This file generates a standard seven digit DTMF signal (using your completed `dtmf_dial` function), adds a specified amount of noise to the signal, and then passes it through your completed `dtmf_decode` function. The decoded string of key presses is compared to those that generated the signal. Since the noise is random, this procedure is repeated ten times. The function then outputs the fraction of trials decoded successfully. The function also displays the plot from the last execution of `dtmf_decode`. (Note: since each call to `dtmf_decode` takes a little time, this function is rather slow. Be patient with it.)

- [4] Execute `dtmf_attack` with various noise powers. Find a value of noise power for which some but not all of the trials fail. What value of noise power did you find? (Hint: use the parameter searching method discussed in the background section to speed your search).
- [8] Make a plot of the fraction of successes versus noise power. Include at least 10 values on your plot. Make sure that your minimum noise power has a success rate of 1 and your maximum noise power has a success rate of 0. Try to get a good plot of the transition between high success rates and low success rates.

(b) By examining the plots for failure trials and the types of errors that the decoder is making, you should be able to identify the source of the errors.

- [3] What types of errors is the system making when it decodes the noisy signals?
- [3] What might be happening within the system to cause these decoding errors?
- [2] Speculate about what could you do to increase the system's tolerance to additive noise.