Laboratory 7

Decoding DTMF: Filters in the Frequency Domain

7.1 Introduction

In Lab 6, 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 obtain a better understanding of 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 phase shift, 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 phase 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.

7.1.1 "The Question"

- How can we decode telephone touch-tone (DTMF) signals?

7.2 Background

7.2.1 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 7.1 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 7.1: 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 7.1 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.

7.2.2 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 7.2.

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.

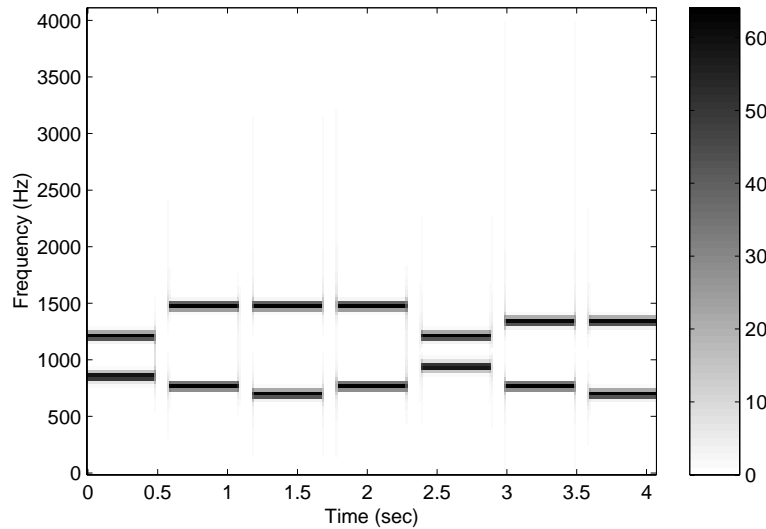[1]Note that each segment is some very small fraction of a second, and the segments usually overlap by 25-75%.

Figure 7.1: A spectrogram of a DTMF signal. Each horizontal bar indicates a sinusoid that exists over some time period.

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

$$h[k] = \begin{cases} \sin(2\pi f_c k/f_s) & 0 \le k \le M \\ 0 & else \end{cases} \tag{7.1}$$

From this equation, we have an FIR filter with order M. (Note that the support length of the impulse response is $M + 1$.) What should M be? M is a design parameter. You may remember from Lab 3 that correlating over a long time produces better estimates of similarity. Thus, we should get better differentiation between passed frequencies and rejected frequencies if M is large. There is a tradeoff, though. The longer M is, the more computation that is required to perform the convolution. Thus for efficiency reasons we would like M to be as small as possible. More computation also equates to more expensive devices, so we prefer smaller M for reasons of device economy as well. Since we have seven DTMF frequencies, we will also have seven bandpass filters in our system; in our decoder system, we will choose a different value of M for each bandpass filter.

Because of the relatively small set of frequencies of concern in DTMF decoding, we will see that larger M 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

$$R = \frac{|\mathcal{H}(f_c)|}{\max |\mathcal{H}(\hat{f})|} \tag{7.2}$$

where $\hat{f}$ is in the set of DTMF frequencies and $\hat{f} \ne f_c$. 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 only the M that we select but also the center
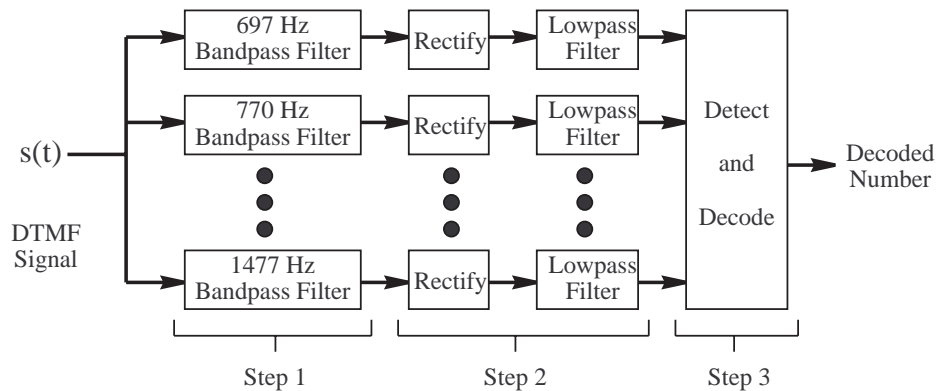
Figure 7.2: A block diagram of the DTMF decoder system. The input is a DTMF signal, and the output is a string of numbers corresponding to the original signal.

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 7.3 to see the effects of these two types of rectifiers.

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 frequency of the filter. To accomplish this smoothing we will use a simple moving average filter with impulse response

$$h_{LP} = \begin{cases} \frac{1}{M_{LP}+1} & 0 \leq k \leq M_{LP} \\ 0 & else \end{cases} \tag{7.3}$$

The order of this filter is M_{LP}. The value M_{LP} (and thus the corresponding strength of the smoothing filter) is a design parameter of the decoder system. When choosing M_{LP}, 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, transient effects will dominate the output of the filter. If it is too short, the system may "smooth over" short DTMF tones or periods of silence. Note that in our decoder system, we will apply the same smoothing filter to the output of each filter. Figure 7.3 shows the results of smoothing for half-wave and full-wave rectified signals.

Step 3: "Detect and Decode"

Once we have processed the outputs of the bandpass filters, we can now detect whether or not a DTMF tone is present and, if it is, determine which key was pressed to produce it. Ultimately, we want to convert our signal into a sequence of keys pressed to produce this sequence. The detect-and-decode step itself involves three steps.

---

[2]Note that the names "full-wave rectifier" and "half-wave rectifier" come from the circuit implementation of these systems
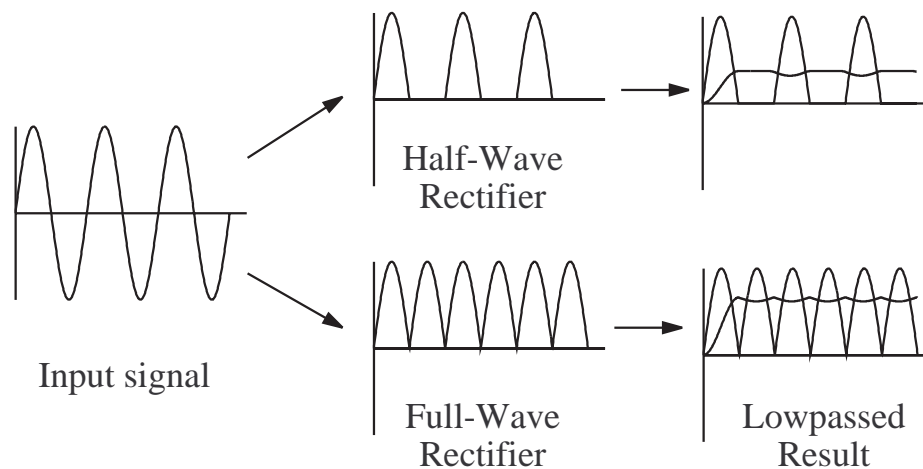
Figure 7.3: A comparison of half-wave and full-wave rectification. Notice that full-wave rectification allows us to achieve a higher output signal level after lowpass filtering.

The first step is to detect whether a DTMF tone is actually present at a particular time. If it is not, we risk making an error in our decoding of the input signal. We detect the presence of a DTMF tone by comparing the rectified and smoothed bandpass filter outputs to a threshold, c. If any of the signals are greater than the threshold, then we decide that a DTMF tone is present. Figure 7.4a shows the rectified-and-smoothed output from one of the bandpass filters and the threshold to which it is compared. The threshold is a design parameter of the decoder. We generally want to the threshold to be high enough that noise will not "trigger" the detector during a period of silence, but low enough that noise won't pull the signal from a DTMF tone below the threshold. Figure 7.4b shows a noisy DTMF tone with the threshold.

When the input signal is noisy, there is an additional problem during the transient portions at the beginning and end of a DTMF tone. Near the threshold crossing, the noise could cause the signal to cross the threshold several times, as shown in Figure 7.4c; this might cause a single DTMF tone to be decoded as multiple key presses. To avoid this problem, we do not make a detection decision for every sample of the input signal. Instead, we only make a decision every 100 samples. This makes it more likely that there will only be one decision made in the vicinity of the threshold crossing. It also reduces computation time somewhat. Note that the number 100 is somewhat arbitrary. We can choose a smaller number, but then we increase the risk the multiple-crossing problem. Alternatively, we can make it larger; however, it we make it too large, our detector may miss short tones or silences.

The second step is to decode of the DTMF tones that we have detected in the previous step. By "decode," we simply mean that we must decide which key was pressed to generate a particular DTMF tone. To do this, we determine which two bandpass filters have the largest output at each time when a DTMF tone was detected. Then, we effectively perform a table look-up to see which key was pressed at these times. The result is a sequence of decoded numbers corresponding to key presses. However, each DTMF tone will generally produce a sequence of identical numbers since it is "decoded" at many times during the duration of the DTMF tone. To translate this sequence of numbers into a sequence of key presses, we need a third step.

The third step simply combines adjacent, identical numbers in the decoded sequence. That is, a "run" of identical numbers is replaced by a single number. Through this process, each DTMF tone is finally represented by a single number. Note that for this process to work correctly, our sequence of numbers must also contain an indication of when no tone was present. Otherwise, any repeated key press would be decoded as only a single key press.
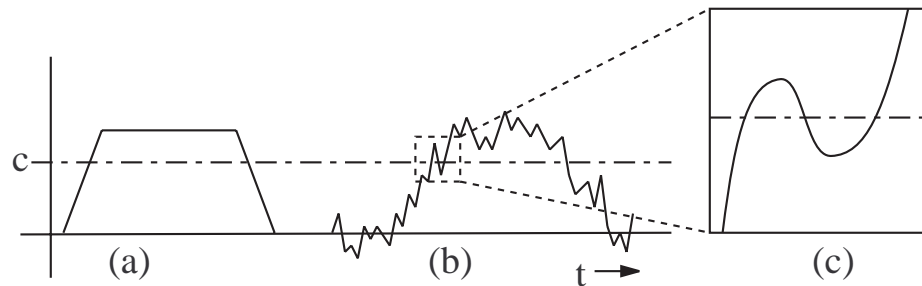
Figure 7.4: An illustration of the detector subsystem. (a) A clean DTMF signal is compared to a threshold, *c*. (b) The threshold should be set so that noise will not produce false tone detections or miss true tone detections in the presence of noise. (c) Near the threshold crossing, noise can cause multiple detections.

7.2.3 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 noise.

7.2.4 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.

7.3 Some MATLAB commands for this lab

- **Computing the frequency response of an FIR filter**: The MATLAB command `freqz` returns the frequency response of a filter at a specified number of discrete-time frequencies. The general usage of `freqz` for causal FIR filters is

  ```
  >> [H,w] = freqz(bb,1,n);
  ```

  Here, `bb` is the set of filter coefficients (i.e., the impulse response) of the FIR filter, n is the number of points in the range $[0, \pi)$ at which to evaluate the frequency response, `H` is the frequency response, and `w` is the set of n corresponding discrete-time frequencies, which are spaced uniformly from 0 to π. The frequency response, `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.

Alternatively, we can evaluate the frequency response only at a specified set of frequencies by replacing n with a vector of discrete-time frequencies. Thus, the command

```
>> H = freqz(bb,1,[pi/3, pi/2, 2*pi/3]);
```

returns the frequency response at the discrete-time frequencies $\frac{\pi}{3}$, $\frac{\pi}{2}$, and $\frac{2\pi}{3}$.

When we apply a filter to a sampled signal with sampling frequency fs (in samples per second), we can evaluate the frequency response at the discrete-time frequencies corresponding to a specified set of continuous time-frequencies in Hertz in the following manner:

```
>> H = freqz(bb,1,[100 200 400 500]/fs*2*pi);
```

This converts the specified continuous-time frequencies into discrete-time frequencies and evaluates the frequency response at those points.

- **Sorting a vector:** The MATLAB command sort sorts a vector in ascending order. Thus, given a vector x, the command

```
>> y = sort(x);
```

produces a vector y such that y(1) is the smallest value in x and y(end) is the largest value in x.

- **Creating matrices of ones and zeros:** In order to create arrays of arbitrary size containing only ones or only zeros, we use the MATLAB ones and zeros commands. Both commands take the same set of input parameters. If only one input parameter is used, a square matrix with the specified number of rows and columns is generated. For instance, the command

```
>> x = ones(5);
```

produces a 5×5 matrix of ones. Two parameters specify the desired number of rows and columns in the matrix. For instance, the command

```
>> x = zeros(4, 8);
```

produces a 4×8 matrix (i.e., four rows and eight columns) containing only zeros. To generate column vectors or row vectors, we set the first or second parameter to 1, respectively.

- **The DTMF Dialer:** dtmf_dial.m is a DTMF "dialer" function. It takes a vector of key presses (i.e., a phone number) and produces the corresponding audio DTMF signal. *Note that this function as provided is incomplete; you will be directed to complete it in the laboratory assignment.* (The lines of code that you need to complete are marked with a ?.) To produce the DTMF signal that lets you dial the number 555-2198, use the command:

```
>> signal = dtmf_dial([5 5 5 2 1 9 8]);
```

An optional second parameter will cause the function to display a spectrogram of the resulting DTMF signal:

```
>> signal = dtmf_dial([5 5 5 2 1 9 8],1);
```

This function assumes a sampling frequency of 8192 samples per second. Each DTMF tone has a length of 1/2 second, and the tones are separated by 1/10 second of silence. Note that the number 10 corresponds to a `'#'`, 11 corresponds to a `'0'`, and 12 corresponds to a `'*'`.

- **The DTMF Decoder:** `dtmf_decode.m` is an (incomplete) DTMF decoder function. (Once again, the lines of code that you need to complete are marked with a ?.) It takes a DTMF signal (as generated by `dtmf_dial`) and returns the sequence of key-presses used to create the signal. Thus, if our DTMF signal is stored in `signal`, we decode the signal using the command:

```
>> decoded = dtmf_decode(signal);
```

An optional second parameter will cause the function to display a plot of the smoothed and rectified outputs of each bandpass filter:

```
>> decoded = dtmf_decode(signal,1);
```

- **Bandpass Filter Characterization:** `dtfm_filt_char.m` is a function that we will use to help us calculate gain-ratios for the bandpass filters used in the DTMF decoder. We use the function to focus on one of the bandpass filters at a time. The function takes two parameters: the order, `M`, of one of the bandpass filter's impulse responses and the center frequency in Hertz, `frq`, of that filter. The function returns a vector containing the gain (i.e., the magnitude of the frequency response) at each of the DTMF frequencies, from lowest to highest. It also produces a plot of the frequency response with locations of the DTMF frequencies indicated. Use the following command to execute the function:

```
>> gains = dtmf_filt_char(M,frq);
```

A second optional parameter lets you suppress the plot:

```
>> gains = dtmf_filt_char(M,frq,0);
```

- **Testing the robustness of the DTMF decoder:** `dtmf_attack.m` is a function that tests the DTMF decoder in the presence of random noise. This function generates a standard seven digit DTMF signal, 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.)

For instance, to test the system with a noise power of 2.5, we use the following command:

```
>> success_rate = dtmf_attack(2.5);
```

The result is a number that provides the fraction of the 10 trials that were successful.

Although `dtmf_attack` is a complete function, it calls `dtmf_dial` and `dtmf_decode`, each of which you must complete.

7.4 Demonstrations in the Lab Section

- Examining the frequency response of FIR filters

- Dual tone multi-frequency signals

- Generating "synthetic" DTMF signals.

- Bandpass filters

- The DTMF decoder

- Noise and the DTMF decoder

7.5 Laboratory Assignment

1. (The DTMF dialer.) Before we can decode a DTMF signal, we need to be able to produce DTMF signals. In this problem, we'll write a function that takes a phone number and produces the corresponding DTMF signal, just like the telephone would produce if you dial the number.

   Download the function `dtmf_dial.m`, which is a nearly complete dialer function. You simply need to replace the 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. Note how each key press produces a different pattern on the spectrogram.
   - [4] What is the phone number that has been dialed in Figure 7.1?

2. (The bandpass filters of the DTMF Decoder.) 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 choose the filter orders and create their impulse responses. In this problem you will be identifying good values for M.

   (a) (The impulse response of one bandpass filter.) First, we need to be able to create the impulse response for a bandpass filter. Using equation (7.1) with a sampling frequency $f_s = 8192$ Hz and $M = 50$, use MATLAB to create a vector containing the impulse response, h, of a 770 Hz bandpass filter[3].
   - [4] What is the command that you used to create this impulse response?
   - [2] Use `stem` to plot your impulse response.

   (b) (The frequency response of one bandpass filter.) 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 it's far simpler to use the `freqz` command. Here, you'll use `freqz` to examine the frequency response and gain-ratio of a bandpass filter like the ones we'll use in the DTMF decoder.

---

[3]Remember that if a filter has order M, the support length of the impulse response should be $M + 1$.

- [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.
- [3] 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? (Hint: You might want to come back to this problem after you've worked the remainder of this problem.)

(c) (Choosing M for this bandpass filter.) Now, we'd like to see what happens when we change M for your 770 Hz bandpass filter. We've provided you with a function that will facilitate this. Download the file `dtmf_filt_char.m`. This function will help you to visualize the frequency response of these filters and to determine their gain at the DTMF frequencies.

- [1] Use this function to verify that the gains you calculated in Problem 2b were correct.
- [2] Include the frequency-response plot that `dtmf_filt_char` produces in your report.
- [4] The frequency response of this filter is characterized by several "humps" which are typically called *lobes*. Describe the frequency response in terms of such lobes. Vary M and examine the plots that result (you do not need to include these plots). Describe the differences in the frequency response as M (which represents the length of the filter's impulse response) is changed.
- [1] What happens to the relative heights of adjacent lobes as M is changed?
- [2] What features of the filter's frequency response contribute to the gain ratio R?
- [2] For what values of M do we achieve gain ratios greater than 10?

(d) (A function for computing gain ratios.) You'll need to compute the gain-ratio repeatedly while finding good design parameters for the bandpass filters, so in this problem you'll automate this task. Write a function that accepts a vector of gains (such as that returned by `dtmf_filt_char`) and computes the gain ratio, R. (Hint: This is a simple function if you use the `sort` command. You can assume that the center frequency gain is the largest value in the vector of gains.)

- [6] Include the code for this function in your report.

(e) (Specifying the bandpass filters.) For each bandpass filter that corresponds to one of the seven DTMF frequencies, we want to find a choice of M that yields a good gain ratio but also minimizes the computation required for filtering.

To do this, for each bandpass filter frequency, use `dtmf_filt_char` and your function from Problem 2d to calculate R for all M between 1 and 200. Then, plot R as a function of M. You can save some computation time by setting the third parameter of `dtmf_filt_char` to zero to suppress plotting. You should be able to identify at least one local maximum[5] of R on the plot. The "optimal" value of M that we are looking for is the smallest one that produces a local maximum of R that is greater than 10.

- [4] Create this plot of R as a function of M for the bandpass filter with a center frequency of 770 Hz. Include the resulting plot in your report.
- [2] Identify the "optimal" value of M for this filter, the associated center frequency gain, and the resulting value of R.
- [12] Repeat the above two steps for the remaining six bandpass filters. (You do not need to include the additional plots in your report.) Create a table in which you record the center frequency, the optimal M value, the associated center frequency gain, and the resulting value of R.

3. (Completing the DTMF decoder.) Now we have designed the bank of bandpass filters that we need for the DTMF decoder. In this problem, we'll use the parameters that we found to help us complete the decoder design.

---

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

[5]A *local maximum* is basically just a point on the plot that is larger than all other values in its vicinity. It may or may not be the highest possible peak, which is called the *global maximum*.

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.

(a) (Setting the M's and the gains of the bandpass filters.) First, you need to record your "optimized" values of M and the center frequency gains in the function. Replace the question marks on line 29 by a vector of your optimized values of M. They should be in order from smallest frequency to largest frequency. Do the same on line 32 for the variable G, which contains the center frequency gains.

- [2] Make these modifications to the code. (At the end of this problem, make sure that you include your completed function in your report.)

(b) (Setting the impulse responses of the bandpass filters.) Also, you need to define the impulse response for each bandpass filter on line 49. Use equation (7.1) for this, where the filter's order is given by M(i).

- [3] Make this modifications to the code.

(c) (Selecting the order of the post-rectifier smoothing filter.) Next, you need to specify the post-rectifier smoothing filter, h_smooth. Temporarily set both h_smooth (line 36) and threshold (line 40) 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 use moving average filters of order M_{LP}, as defined by the MATLAB command

```
>> h_smooth = ones(M_LP+1,1)/(M_LP+1);
```

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.

- [4] Examine the behavior of the smoothed signal when you replace line 36 with moving average filters with order M_{LP} equal to 20, 200, and 2000. Which filter order, M_{LP} gives us the best tradeoff between transient effects and smoothing?
- [1] Set h_smooth to be the filter you have just selected.

(d) (Detection threshold.) Finally, you need to identify a good value for threshold. threshold determines when our system detects the presence of a DTMF signal. `dtmf_decode` plots the threshold on its figure as a black dotted line. 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 the start-up transients for each DTMF tone. (Hint: When choosing a threshold, consider what might happen if we add noise to the input signal.)

- [4] 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.
- [2] Now, execute `dtmf_decode` and include the resulting plot in your report. (Note: You can include this plot in black and white, if you like.)
- [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.

(e) Remember to include the code for your completed `dtmf_decode` function in your report.

4. (Robustness of the DTMF decoder to noise.) 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`. Execute `dtmf_attack` with various noise powers. Find a value of noise power for which some but not all of the trials fail.

- [3] What value of noise power did you find? (Hint: use the parameter searching method discussed in the background section to speed your search).
- [6] 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 at (or at least near) 1 and your maximum noise power has a success rate at (or near) 0. Try to get a good plot of the transition between high success rates and low success rates. While making this plot, pay attention to the types of errors that the decoder is making.

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

- [3] What types of errors is the system making when it decodes the noisy signals?
- [2] Speculate about what could you do to the decoder in order to increase the system's tolerance to additive noise.

5. On the front page of your report, please provide an estimate of the average amount of time spent outside of lab by each member of the group.