8

Spectral Laboratory

8.1 Purpose

The purpose of this lab is to design a program to decode DTMF tones..

8.2 Background

- Read Section 14.1 on spectral analysis.
- Remind yourself what DTMF tones are. (We discussed these in Section 4.4 of Lab #4).

There are twelve keys on the normal phone arranged in a 4x3 matrix. When you press a key, the circuitry in the phone produces a sound that comprises the sum of two pure sine tones of equal amplitude and distinct frequencies. One of the tones – the *row tone* -- has a choice of four frequencies (697, 770, 852 and 941 HZ), each one indicating one of the rows of the keypad. The other tone – the *column tone* -- has a choice of three frequencies (1209, 1336 and 1477 Hz), each one indicating a column of the keypad. The frequencies of the row and column tones are monotonically increasing and also disjoint (that is, all the row tones are of lower frequency than the column tones). The sum of the two tones indicates a row and column in the keypad matrix, and hence identifies a unique key. For example, when you press the '8' key, you get the sum of sin waves at 852 (indicating the third row) and 1336 Hz (indicating the second column). Here is the touch-tone key pad with all the tone frequencies indicated.

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

The task of the circuitry in the phone office is to decode the DTMF tones produced in the phone to determine which keys were pressed. This means we have to analyze the sum of two tones and determine the frequency of each one.

8.3 Assignment

The sole assignment is to write a Matlab function `dtmfdecode` that decodes DTMF tones and produces a transcript of the number that it encodes.

```
function str = dtmfdecode(s, fs)
% DTMFDECODE Decode DTMF tones
%            str = decodedtmf(s, fs)
%            Accepts a array, s, which corresponds to the DTMF tones
%              sampled at fs.
%            Produces a string transcript that decodes the tones.
```

In this project, you will be given a .wav file created from tones keyed in by a phone. Here's what you know about these phone tones:

- There are between 1 and 10 tones in the file.
- The tones have a minimum length of 100 msecs
- The tones are separated from each other by a minimum of 50 msecs of silence.
- The signal-to-noise ratio will be at least 20 dB. That is, the energy of a section of the file which corresponds to a tone will be at least 10 time greater than a section of the file that corresponds to silence.
- The amplitude of the data has been normalized to 1.
- There can be a variable amount of 'lead-in' silence at the beginning of the file and 'lead-out' silence at the end. The amount could be zero.

The sound file `phonetones.wav` is a recording of the sequence of all the keys: '123456789*0#'. If you input this into your `dtmfdecode` function, this is what you should get:

```
>> [x, fs] = audioread('phonetone.wav');
>> dtmfdecode(x, fs)
ans =
     123456789*0
```

In Lab #4, you used lowpass and highpass filters to separate the row and column tones. In this lab, you can use any method you like. However, you might want to consider a spectral method, since each tone is spectrally very simple: it only comprises the sum of two pure tones. The issues you will need to think through include the following:

- If you are using the DFT to calculate the spectrum, you will need to chunk the input data `s` into a series of frames of some length `frameLength`. You want to keep the frame length short enough that a single frame doesn't include sound data corresponding to more than one digit. It may want to be some 10's of msecs long.
- If you use an N-pt DFT, you will want to remind yourself of the relation between the n^{th} point of the DFT and the frequency ω, and the relation between ω and f given the sample rate f_s.
- Before you take the DFT of each frame, you will probably want to window it in some way (why?) If your frame length is too short, you may have problems with estimating the peaks of the spectrum (why?). So the length of each frame needs to be not too long and not too short.

- You will want to choose a length of DFT that gives you adequate spectral resolution. You might remember that the star ('*') key is the hardest case to resolve, since the frequencies are closest together (highest row frequency and lowest column frequency).
- If you are using a frame length that is smaller than the length of a tone corresponding to a given digit, then you will be making multiple estimates of the spectral frequency for that digit. You may choose to average those estimates, or just pick the best one, in some way.
- You need to know whether a given frame is "in" a digit and when it is in silence. One simple way to do this is to measure the *total energy* of a given frame (or you could just measure the energy in the spectral components of interest). You can measure the energy by calculating `x.^2`, or simply using Matlab's `var` command. If you add two sine waves of equal amplitude and different frequency and scale the sum so that the maximum amplitude is one, what do you expect to measure for the energy?

Here are a few tone files for you to play with:

- All the digits, no noise: `phonetones.wav`
- All the digits with some noise added: `phonetones_noise.wav`
- A star tone: `startone.wav`
- Information please!: `info.wav`

To test your code, there are a bunch of wavfiles numbered `s1.wav` – `s6.wav`. Put these in the same directory as your `dtmfdecode` function. When you run it, you should get:

```
>> lab8_2024
s1: 123456789*0# is O.K.
s2: 123456789*0# is O.K.
s3: 123456789*0# is O.K.
s4: 123456789*0# is O.K.
s5: 123456789*0# is O.K.
s6: 123456789*0# is O.K.
```

Here's are the parameters of the wave files:

File	Sampling rate (Hz)	Lead-in silence (msecs)	Lead-out silence (msecs)	Tone length (msecs)	Tone spacing (msecs)	SNR (dB)
s1	8000	40	40	100	50	∞
s2	8000	0	0	100	50	∞
s3	8000	40	40	200	100	∞
s4	4000	40	40	100	50	∞
s5	8000	40	40	100	50	30
s6	8000	40	40	100	50	10

If you can't get the last one (s6) to work, that's O.K. It's pretty challenging.