Spectral Analysis Lab

Purpose

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

Background

- Read here (SpectralAnalysis Attachment) on spectral analysis.
- Remind yourself what DMTF tones (DTMF Tones Attachment) are.

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.

Here's a sample wav file (phonetones) corresponding to the sequence of all the keys: '123456789*0#'. If you input this into your `dtmfdecode` function, this is what you should get:

```
>> [s, fs] = wavread('phonetone');
>> dtmfdecode(s, fs)

ans =
```

```
123456789*0#
```

In previous labs, you have played with using lowpass and highpass filters to separate the row and column tones and also learned how to use the FFT to compute the frequency response of signals and systems at varying resolutions. In this lab, you can use any method you like to achieve the desired result. However, you might want to consider a spectral method. Because each DTMF tone is spectrally very simple -- it only comprises the sum of two pure tones -- it is quite natural to use a transform technique. 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 so 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.
- 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 '*' 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 estimate, or just pick the best on, 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 `s.^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, get the following zip file of stuff, which consists of `lab8.m` and a bunch of wavfiles. Unzip this and put this in the same directory as your `dtmfdecode` function. When you run it, you should get:

```
>> lab8
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 what the wavefiles are

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	inf
s2	8000	0	0	100	50	inf
s3	8000	40	40	200	100	inf
s4	4000	40	40	100	50	inf
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. Listen to it!