Part A: Speech Scrambling

In Part A of Experiment 7.4, we converted a low pass filter to a high pass filter by shifting its frequency response up by Fs/2. Note that this has the effect of reversing the shape of the filter since it moves its negative frequencies (which are the mirror image of its positive frequencies) up to Fs/2. In other words, we have reflected or *reversed* the spectrum about the frequency Fs/4 (2.5 kHz in this case).

Since this was accomplished by multiplying the time domain representation of the signal by (−1)n, we should be able to reverse the spectrum of a *signal* by multiplying it by the same sequence. This is indeed the case and the process is called (unimaginatively enough) *spectral reverse*. For reasons which you will soon see, it has been used as a technique for *scrambling* speech to prevent eavesdroppers, wiretappers, and spies from being able to understand it.

* First we need our sequence (-1)n:  
  >> flip = cos(pi\*[1:40000]’);
  + Note: Be sure to include the prime (‘) symbol.
* Now we multiply our top secret message by the encoding sequence:  
  >> out1 = sig1 .\* flip;
* Look at the spectrogram of the output signal and verify that the spectrum of the original signal has indeed been flipped.
* Listen to the signal. Can you understand it?
  + As a test of the security of the system, have a member of another lab group listen to your reversed signal and try to understand it!
* So how do we unscramble the signal? Simple, since the reflection of a reflection is the original, simply rereflect the scrambled signal.  
  >> out2 = out1 .\* flip;
* Listen to the rereflected signal and verify that it is unscrambled.
* **Draw a block diagram of the scrambler system you have built. Show the spectrum of the signal at each point in the system.**

Part B: Computer Music

In the previous Experiment we created discrete (single-frequency) tones, swept tones, and "fuzzy" tones. An artistic application for all these tones would be to string them together to form tunes. Indeed, one of the very early applications of the digital computer was in the production of music.

Let’s see if we can build up a simple tune.

* First we define the tempo to be 4/4 at 150 beats/minute.  
  >> quarter = [1:4000];  
  >> half = [1:8000];  
  >> eighth = [1:2000];
* Now let’s make a note.  
  >> n1=sin(0.2463 \* quarter);  
  >> sound(n1,Fs)
* Add a few more notes to our collection:  
  >> n2=sin(0.2765 \* quarter);  
  >> n3=sin(0.3103 \* quarter);
* Now we can build up a phrase:  
  >> phrase = [n1 n2 n3 n1];
* Add a repeat:  
  >> tune = [phrase phrase];
* Let’s check how it sounds:  
  >> sound(tune, Fs)
  + Can you recognize our “mystery tune” from this fragment?
  + If you would like to try to complete this tune (or make up a tune of your own), here is a [table of note frequencies](file.35.html).

Part C: Making Better Notes

Music made up of just a sequence of sine waves is pretty boring. The musical notes we’ve seen in previous labs had more harmonics. Although they have a strong fundamental and few harmonics, whistled notes and notes from a flute are not pure tones. Other instruments (e.g. violins, trumpets, the voice) have an even richer set of harmonics. In each case this *harmonic structure* is one of the characteristics that gives the instrument its individual sound.

Another is the variation in amplitude of a note with time, called the *envelope*. A note from a piano begins abruptly when the hammer strikes the string, then gradually decays away. A note from a trumpet can begin very softly, build to a loud *forte*, and sustain until the trumpeter runs out of breath.

In some cases the harmonic structure as well as the amplitude change with time. The various harmonics in a piano note decay at different rates. In a bell, not only do the different frequencies that make up the note decay at different rates, but they are not harmonically related (and so are called *partials* rather than harmonics). Nevertheless, the ear *perceives* a pitch, based on relationships between these partials.

The ear can also perceive pitch even when no discrete frequencies are present (i.e. in a signal which is *random*).

* As an example of how the shape of the envelope shapes the character of the note, let's make our fake flute into a fake piano by adding an envelope with a sharp attack and gradual decay.  
  >> p1 = n1 .\* (0.9995 .^ quarter);  
  >> plot(p1)  
  >> sound(p1,Fs)
* Let’s try a complete tune with our new “instrument”:  
  >> p2 = n2 .\* 0.9995 .^ quarter  
  >> p3 = n3 .\* 0.9995 .^ quarter  
  >> tune2 = [p1 p2 p3 p1 p1 p2 p3 p1];  
  >> sound(tune2,Fs)
* **Plot the time function and the spectrogram of this tune. Can you identify the notes?**
* As mentioned above, we can also create notes that are not periodic. First we need our FIR lowpass filter back.  
  >> b=fir1(50,200/5000);
* Now we turn it into a bandpass filter centered about our desired note, put white noise into it, and get out narrowband noise which is our note.  
  >>  r1=filter(b.\*cos([0:50]\*0.2463\*4),1,randn(size(n1)));  
  >> sound(r1,Fs)
* Add a few more notes to make a complete set, and we can get another version of our tune.  
  >> r2=filter(b.\*cos([0:50]\*0.2764\*4),1,randn(size(n1)));  
  >> r3=filter(b.\*cos([0:50]\*0.3103\*4),1,randn(size(n1)));  
  >> tune3 = [r1 r2 r3 r1 r1 r2 r3 r1];  
  >> sound(tune3, Fs)
* Plot the time function of this tune:  
  >> plot(tune3)  
  **Can you find the individual notes? Now plot the spectrogram. Can you find them now?**
* We can make an "orchestra" by adding together the sounds of several instruments.  
  >> sound(tune2+tune3, Fs)
* However, as you probably know, this tune is a *round*, i.e., it is intended to be added to a time-shifted version of itself. To make this work, we will need the entire tune. If you have not completed the tune yet, the notes are below.  
  Phrase1: G(.2463)/4, A(.2765)/4, B(.3103)/4, G(.2463)/4; Repeat.  
  Phrase2: B(.3103)/4, C(.3288)/4, D(.3690)/2; Repeat.  
  Phrase3: D(.3690)/8, E(.4142)/8, D(.3690)/8, C(.3288)/8 B(.3103)/4 G(.2463)/4; Repeat.  
  Phrase4: G(.2463)/4, D(.1845)/4, G(.2463)/2; Repeat.
  + *The notation is name(frequency)/duration, i.e. G(.2463)/4 is a quarter note G with 2πf/Fs=.2463, which is generated by G4 = sin(.2463 \* quarter);*
  + *Giving the notes more mnemonic names (G4 for a G quarter note, D2 for a D half note, etc.) should make it easier to build the tune without making errors.*
  + *"Repeat" means to repeat all the notes from the beginning of the phrase (once, not recursively).*
* To play the tune as a round, we need to add the tune to a copy of itself shifted by the length of one phrase. Since Matlab doesn't like adding vectors of different length, we will have to pad the beginning (or end) of each voice with zeros.  
  >> voice1=[Tune zeros(size(phrase1))];  
  >> voice2=[zeros(size(phrase1)) Tune];  
  >> soundsc(voice1+voice2, Fs);
* A more interesting effect can be obtained by having the different voices played by different "instruments". If you feel up to it, make another version of the complete tune using one of the other sounds we've created (or if you don't like any of them, make up your own instrument). Then play these two voices as a round as in the previous step.