UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN
Department of Electrical and Computer Engineering

ECE 498MH Signal and Image Analysis

Homework 4

Fall 2013

Assigned: Friday, September 27, 2013 Due: Friday, October 11, 2013

Reading: Signal Processing First (SPF) Chapter 6

Problem 4.1

Find the impulse response of the following system:

$$y[n] = 0.5x[n] + 0.6x[n-1] + x[n-2]$$

Problem 4.2

The input to each of the following systems is $x[n] = \cos(\omega n)$ for some frequency ω. In each case, find the output of the system. In each case you should express the output of the system as $y[n] = A\cos(\omega n + \theta)$, where A and θ are terms that depend on ω. Thus, for example, in part (b), it's not sufficient to write the output as a sum of three cosines; you need to write the output as a single scaled, phase-shifted cosine.

(a) $y[n] = x[n] - x[n-1]$

(b) $y[n] = x[n] + 0.6x[n-3] + 0.4x[n-4]$

(c) $y[n] = 0.5x[n] + 0.6x[n-1] + x[n-2]$

Problem 4.3

Find $y[n] = x[n] * h[n]$, where

$$h[n] = \begin{cases} 1 & 0 \le n \le 3 \\ 0 & \text{otherwise} \end{cases}$$

$$x[n] = \begin{cases} 1 & 0 \le n \le 7 \\ 0 & \text{otherwise} \end{cases}$$

Matlab Exercises

In this problem you will synthesize a short tune using pure tones, then analyze notes from two different musical instruments, then resynthesize the same tune as played by each of the same two instruments. Your handed-in homework for this problem consists of four waveform files: a pure-tone song, a song synthesized to sound like a musical instrument, and then low-pass filtered and high-pass filtered versions of the latter.

In order to do this problem, you might find it useful to know that the standard musical scale is logarithmic. Thus if $\cos(2\pi F_1 t)$ and $\cos(2\pi F_2 t)$ are one octave apart, that is equivalent to saying that $F_2 = 2F_1$, or that

$\ln F_2 = \ln F_1 + \ln 2$. Similarly if $\cos(2\pi F_1 t)$ and $\cos(2\pi F_2 t)$ are one semitone apart, that is equivalent to saying that $F_2 = (2^{1/12})F_1$, or that $\ln F_2 = \ln F_1 + (1/12)\ln 2$.

All musical tones are defined relative to piano note A4, the A in the center of the piano, which is at 440Hz. In this lab we will use the notes starting at A5 (one octave above A4), because the D/A cards in most computers work better at slightly higher frequencies. Thus we will use

$$\begin{aligned} F_A &= 880 \\ F_B &= 987.77 \\ F_C &= 1046.5 \\ F_D &= 1174.7 \\ F_E &= 1318.5 \\ F_F &= 1396.9 \\ F_G &= 1480.0 \end{aligned}$$

(a) Create pure tones of one second in length, at each frequency. Thus, for example, define `FS=16000;`, then `t=[0:(FS-1)]/FS;`, then `FA=880;`, then `toneA=cos(2*pi*FA*t);`. Play your tones C through G, then A and B, to make sure that they sound like a major scale.

(b) Create a short tune by concatenating the notes you need. Use one-second measures, thus in order to insert a quarter-note on the A, you would insert `toneA(1:(FS/4))`. Thus, for example, the first seven notes of "doh, a deer" could be created as: `dohadeer = [toneC(1:(3*FS/4)), toneD(1:(FS/4)), toneE(1:(3*FS/4)), toneC(1:(FS/4)), toneE(1:(FS/2)), toneC(1:(FS/2)), toneE(1:FS)];`. Of course, you might want to use several lines to construct this, using the syntax `dohadeer=[dohadeer, toneD(1:(FS/4))];` and so on. Play your song. Tweak it until you are happy with it (tweaking may be easiest if you put these commands into a script file). If you need notes beyond the ones you've already created, go ahead and create them by doubling or halving the frequencies you've already used.

Save this song using `wavwrite`, and hand in a copy electronically.

(c) Download samples of two different musical instruments from the University of Iowa musical instrument samples library, at http://theremin.music.uiowa.edu/MIS.html. The first note in each waveform is a C; since the C you have synthesized is C6, you probably want to download the scale C6B6. Perform a Fourier analysis of the C6 note from each of your two instruments: cut out exactly ten periods from the note, take the Fourier transform of those ten periods, and find the magnitude and phase of the first seven transform components using commands something like `magX=abs(fft(x)); MTRUMPET=magX(11:10:71);` and `phaX=unwrap(angle(fft(x))); PTRUMPET=phaX(11:10:71);`.

(d) Construct synthesized versions of your two musical instruments at each note needed for your tune. Thus, for example,

```
trumpetC = zeros(1,FS);

for k=1:7,

trumpetC=trumpetC+MTRUMPET(k)*cos(2*pi*k*FC*t+PTRUMPET(k));

end
```

You may find it useful to put these commands into a script, so that you can copy them several times, and change the type of instrument and the name of the note in each place.

When you have created these notes, listen to them. They may not sound entirely natural, but they should sound recognizably similar to the instrument you analyzed.

Another way to do this is to create a matrix with all of the seven tones, A-G, in the rows of the matrix, thus

```
notes=[FA,FB,FC,FD,FE,FF,FG]
tones=zeros(7,FS);
for n=1:7, for k=1:7,
tones(n,:)=tones(n,:)+MTRUMPET(k+1)*cos(2*pi*k*notes(n)*t+PTRUMPET(k+1));
end
```

When you have created these notes, listen to them. They may not sound entirely natural, but they should sound recognizably similar to the instrument you analyzed.

(e) Construct versions of your short song as played by the instrument you have analyzed. Save a version of this song, and send it to me.

(f) Construct an averaging low-pass filter: `hlpf=0.1*ones(1,10);`. Use `ylpf=conv(hlpf,song);` to convolve your song with the lowpass filter. Use `wavwrite` to save a copy of the lowpass filtered song, and send it to me.

(g) Construct a first-difference high-pass filter: `hhpf=[1,-1;`. Use `yhpf=conv(hlpf,song);` to convolve your song with the lowpass filter. Use `wavwrite` to save a copy of the lowpass filtered song, and send it to me.