E85.2607 – Advanced Digital Signal Theory
Assignment 2: Delays and Reverb
Due: 2010-03-04 by 5pm

Instructions

Send me an email containing your solutions by the due date listed above. Please attach a zip file containing your solutions, including the following:

1. A writeup including any requested plots as a **PDF** file. All Word documents will be ignored.
2. Any Matlab code you wrote as separate plain text files.
3. Any interesting sounds you generated.

Your submission should be your own work. If you're having trouble please send me an email or come to my office hours.

1 Subtractive synthesis

Following up on homework 1, you are going to make some more music in Matlab using subtractive synthesis. The idea in subtractive synthesis is to begin with a spectrally rich signal (e.g. white noise, or an oscillator with a rich harmonic spectrum) and use filters to shape the spectrum and change the sound.

In this problem you're going to use comb filters to play your favorite tune (aka Mary Had a Little Lamb).

1. Write a function to design an FIR comb filter that has peaks at integer multiples of the given frequency (i.e. has a "harmonic" spectrum with the given fundamental frequency).

   We want to play "notes" using this filter so be sure that there is no peak at 0 Hz or things won't sound right. Recall from class that you can control this using the gain parameter. Your filter should keep this gain fixed.

   The function should have the following signature:

   ```
   function [b, a] = fircomb(frequency, fs)
   % [b, a] = fircomb(frequency, fs)
   %
   % Designs a comb filter with peaks at multiples of the given
   % frequency (in Hz),
   %
   % Returns a list of feedforward (b) and feedback (a) coefficients of
   % the same form used by the filter, freqz, etc. commands.
   ```

2. Use `freqz` to plot the magnitude response of your FIR comb filter with a frequency of 441 Hz with $f_s = 44100$. Now try setting the frequency to 440 Hz. Does this work? Why not?

3. Our synthesizer is never going to work if we can't play A4! Fix `fircomb` so that it works for the above case (hint: think fractional delays). Call this function `fircomb_fixed`.

4. Write the analogous function to design an IIR comb filter called `iircomb_fixed`. Plot the magnitude response of an IIR filter (designed with this function) for a frequency of 880Hz with $fs = 44100$.

5. Which of these comb filters do you want to use to make music? Why?

6. Lets play some notes. Write a Matlab function to make a note by passing white noise through the appropriate comb filter. The function should have the following signature (this is starting to feel like homework 1 all over again, isn't it):

```
function x = noise_excited_comb(frequency, duration, fs)
% x = noise_excited_comb(frequency, duration, fs)
%
% Generates a note at the given frequency (in Hz) using a noise
% excited comb filter that lasts for the given duration (in seconds).
```

7. Write a Matlab script to play Mary Had a Little Lamb (or your favorite tune) using `noise_excited_comb`.

   It sounds a bit metallic, doesn't it. Comment on ways you could improve the sound.

8. Noise is kind of boring. Find some other sounds to pass through comb filters to play notes. What characteristics should such sounds have?

   Scour your hard drive (or the internet) to find something appropriate (be creative!), and compose the same tune by passing your new signal through comb filters.

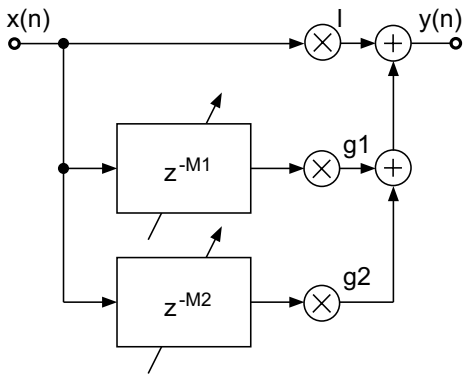
9. Plot spectrograms of the two signals you generated and comment on the differences in the way they sound/appear on the spectrogram. Play around with the `caxis`, `xlim`, and `ylim` commands to make it easier to see whats going on by changing the color axis and zooming in and out of the plots. Don't forget to add `colorbars`!

   Please include wave files of the signals you generated with your submission.

2 Chorus

For this problem you're going to do the unthinkable: create a time-varying effect in Matlab. This is not something I would normally recommend, but its worth trying, if only just once.

The block diagram for a chorus effect looks something like this:



The difficulty in translating this diagram into Matlab lies in the diagonal arrows over the delays. Matlab's filtering tools only work for LTI systems, so you won't be able to use `filter`. The simplest way to create a time-varying delay in Matlab is to simulate the difference equation directly in the time-domain using a **gasp** for loop.

1. In order to build a chorus effect you're going to need to generate some kind of modulation signal to vary the delay. One way to do this is to decompose the M_i terms in the block diagram above into a time-varying *signal* as follows:

$$M_i[n] = M_{i,\text{fixed}} + d\, m[n]$$

where $M_{i,\text{fixed}}$ is a fixed delay of the sort you're used to, $m[n]$ is a modulating function that changed with time, and d is a parameter controlling the *depth* of the delay, i.e. how far the overall delay is allowed to move away from $M_{i,\text{fixed}}$.

Following DAFX's example, you should use lowpass noise to generate $m[n]$. (Why noise instead of a simple oscillator?)

You can generate this by creating white noise in the usual way and then filtering it. For this assignment you can let Matlab do the hard work of designing the filter parameters for you using the `butter` function:

```
[b, a] = butter(2, cutoff * 2 / fs, 'low');
```

This command will give you the transfer function coefficients for a 2nd order lowpass filter with a cutoff frequency of `cutoff` Hz given a sampling rate of `fs`. Take a look at the Matlab help (`doc butter`) to learn more about `butter` and Butterworth filters.

Write the following function to synthesize $M_i[n]$ for use in your chorus effect:

```
function y = time_varying_delay(length, fixed_delay, depth, fs)
% y = time_varying_delay(length, fixed_delay, depth, fs)
%
% Generates a time-varying delay signal for use in a chorus
% effect.
%
% Inputs:
%   length       -  length of the output signal in samples
%   fixed_delay  -  fixed portion of the delay signal in ms
%   depth        -  maximum delay variation in ms
%   fs           -  sampling rate
```

You should hardcode the lowpass filter cutoff frequency. Use something in the vicinity of 1-5 Hz.

2. Write a function to implement a chorus effect using `time_varying_delay`. The free parameters will be the number of delays to use (i.e. the number of parallel blocks in the block diagram), and the depth of the delay variation. The other parameters (the gains on each delay line ($g1, g2$ in the block diagram) and the fixed portion of the delay ($M_{i,\text{fixed}}$)) should be set randomly within your function.

The function should have the following signature:

```
function y = chorus(x, ndelays, depth, fs)
% y = chorus(x, ndelays, gains, fs)
%
% Inputs:
%   x        -  input signal
%   ndelays  -  number of delays to use
```

```
%   depth     -  maximum delay variation in ms
%   fs        -  sampling rate
```

Don't worry about fractional delays for this. (Why doesn't it matter for chorus?)

3. Download `http://www.ee.columbia.edu/~ronw/adst/homeworks/hw2-vocals.wav` and run it through your chorus. Play around with the different parameter settings and find one that you like. Include a wave file of the sound you synthesize and a brief statement on how you chose the parameters in your submission.

   Warning: the run time is going to be extremely slow. It will take (at least) a few minutes to process this track. When debugging/messing around I suggest you use a short sound, say a 1 or 2 second long excerpt, or you're going to be waiting around all day.

3 Convolution Reverb

(This problem has been blatantly plagiarized from Matt Hoffman.)

Since reverberation in a stationary room is a linear, time-invariant phenomenon, we can simulate it by using the sampled impulse response of the room as the coefficients of a digital FIR filter. Alternatively, we can generate an artificial impulse response that resembles a plausible room impulse response.

1. Write a MATLAB function that generates an artificial room impulse response and convolves an input signal by that impulse response. The function declaration should look like:

```
function output = convReverb(x, t60, mix, fs)
% output = convReverb(x, t60, mix, fs)
%
% where x is the input signal and t60 is the time in seconds it takes
% for the impulse response to decay to -60 dB (a common measure of
% reverb length), mix is the amount of the reverberated signal to mix
% with the input signal, and fs is the sampling rate.
```

   To generate the impulse response, do the following:

   - Generate t60 samples of Gaussian noise (use the RANDN function).
   - Multiply the noise by an exponentially decaying amplitude envelope, such that the envelope gets down to -60 dB (1/1000) by the last sample. If you want to be a good loop-hating MATLAB programmer, consider using the CUMPROD function to generate the envelope.
   - Set the first 100 ms of the impulse response to 0 to simulate the amount of time it takes for the reverberations to "diffuse."
   - Set four samples of the impulse response equal to 1 around 43, 61, 87, and 97 ms to simulate four early reflections.
   - Convolve the impulse response with the input signal x, multiply by mix, and add the result to the input signal x.

2. Use your `convReverb` and `chorus` functions to process the sound from the previous problem. The goal is to make it sound (something) like a choir of Freddie Mercury clones singing in a big church. In your submission include the sound file and a description of how you chose the reverb and chorus parameters to get this effect.