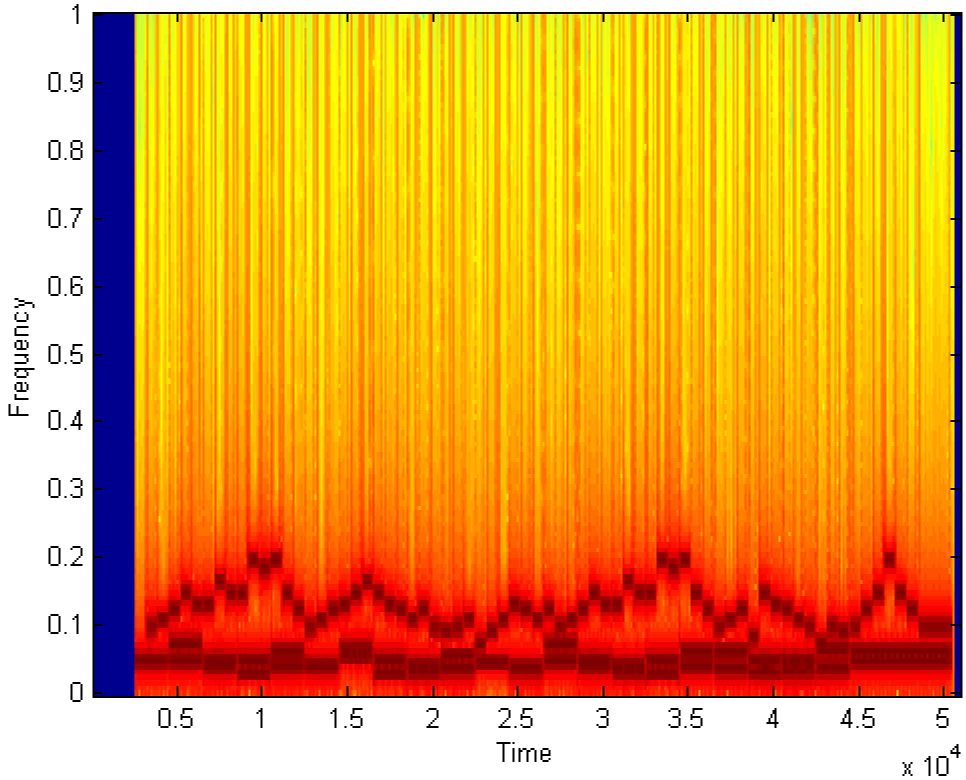
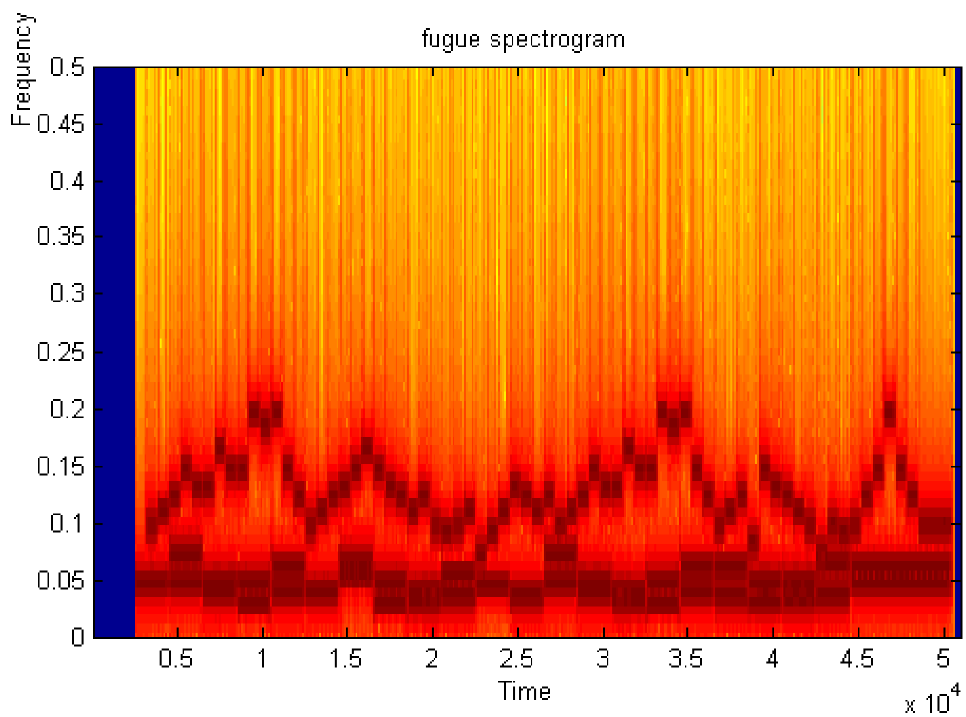
Lab 4: Music Synthesis - Dan Wortmann March 3rd, 2014

Table of Contents

4.0

```
load('bach_fugue')

%I decided to chose an fs of 11025 as my sampling frequency because it seems to be
%a pretty standard number for the hardware we use. However the playback
%frequency I decided on a little lower in order to get play the song at the
%right tempo.

fs = 11025;

synthesize(theVoices, fs);
soundsc(song,6000)

specgram(song)

%better zoomed in view of the specgram shows us a better view of the
%actual notes as they progress in the song.
```


Frequency
Time
x 10^4
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5


fugue spectrogram
Frequency
Time
x 10^4
0.5
0.45
0.4
0.35
0.3
0.25
0.2
0.15
0.1
0.05
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5

CODE OF FUNCTIONS

```
% function xx = envelope(A, phi, keynum, dur, fs)
% % KEY2NOTE Produce a sinusoidal waveform corresponding to a
% % given piano key number
% %
% % usage: xx = key2note (X, keynum, dur)
% %
% % xx = the output sinusoidal waveform
% % A = amplitude of the note
% % keynum = the piano keyboard number of the desired note
% % dur = the duration (in seconds) of the output note
% %
%
% %fs = 11025; %-- or use 8000 Hz
% tt = 0:(1/fs):dur;
% freq = 440 * 2^((keynum - 49)/12);
% %construct a vector of varying amplitudes for the envelope
% AA = [];
%
% %the attack - first 10%
% t1 = linspace(0, 1.15*A, round(.1*length(tt)));
% AA = [AA,t1];
%
% %the Delay - next 5%
% t2 = linspace(1.15*A, A, round(.05*length(tt)));
% AA = [AA,t2];
%
% %Sustain - middle 65%
% t3 = linspace(A, .85*A, round(.65*length(tt)));
% AA = [AA,t3];
%
% %remaining indices
% rest = length(tt) - length(AA);
%
% %Release - final 20%
% t4 = linspace(.85*A, 0, rest);
% AA = [AA,t4];
%
% %u = .1*length(tt) + .05*length(tt) + .65*length(tt) + .2*length(tt) %debug chec
%
% XX = AA*exp(j*phi); %calculate the complex amplitude
%
% xx = real( XX.*exp(j*2*pi*freq*tt) );
%
% end

%///////////////////////////////////////////////////////////////////////////

% function song = synthesize( theVoices, fs )
% %SYNTHESIZE Summary of this function goes here
% %
% bpm = 120;
```

```
% bps = bpm/60;
% seconds_per_beat = 1/bps;
% seconds_per_pulse = seconds_per_beat / 4;
%
% fs = 11025;
% spp = seconds_per_pulse * fs; %get samples per pulse
%
% numVoices = length(theVoices); %get number of melodies
%
% %loop through all the melodies
% %this will seperate out all the information per melody basis so they are
% %seperate for the future addition.
% for k = 1:numVoices
%     numNotes(k) = length(theVoices(k).noteNumbers);
%     if numNotes(k) > 0
%     pulse(k) = theVoices(k).startPulses(numNotes(k));
%     dur(k) = theVoices(k).durations(numNotes(k));
%     end
% end
%
% %initialize the song array that will hold the sounds
% song = zeros(1, ceil( spp * ( max(pulse) + max(dur))));
%
% for k = 1:numVoices
%     for n = 1:numNotes(k)
%         %get the durations (in seconds) of the notes at the kth index
%         timeDuration = seconds_per_pulse * theVoices(k).durations(n);
%         z = timeDuration; %check debug
%
%         %determine the sound of the tone with an envelope that uses a
%         %similar functions as the key2num
%         tone = envelope(10, 0, theVoices(k).noteNumbers(n), timeDuration);
%         toneLength = length(tone);
%         x = length(tone); %check debug
%
%         %determine the index of the begging of the sound to assigne the
%         %current and future tones to it
%         startSong = round( (theVoices(k).startPulses(n) - 1) * spp + 1);
%         y = startSong;    %check debug
%         song(startSong:startSong + toneLength - 1) = ...
%                        song(startSong:startSong + toneLength - 1) + tone;
%
%                         %add to the current index which contain the tones
%     end
% end
%
% end
```

Published with MATLAB® R2013a