Andy Lew
DSP Project 1 Report

Abstract:

The objective of this project was to learn more about how to synthesize music using sinusoids through the process of additive synthesis. The main program that was used to achieve this was MATLAB. Different functions were used to acquire certain frequencies that were related to certain notes correlating to a piano. The two methods of finding the frequencies were just-tempered tuning and well-tempered tuning with just-tempered tuning being based off of ratios corresponding to a single note and well-tempered tuning being based off a formula to calculate the different frequencies. A main function was then used alongside these functions to synthesize the music and play it through the speakers of a computer. Harmonics and envelopes were added as a finishing touch to make the sound produced more realistic. The song used to test the functions was Bach's "Fugue #2 for the Well-Tempered Clavier." Unfortunately the timing of the final synthesized song was not completely correct, while most of the notes were able to be played.

Procedure:

Two functions were first created in MATLAB named key2notewt and key2notejt which found frequencies through the well-tempered tuning method and just-tempered tuning method respectively. The key2notewt function would take in three parameters: X, keynum and dur and then output a sinusoidal waveform. X corresponded to the complex amplitude of a sinusoid in the form of A*e^(j*phi), keynum was the piano keyboard number of the desired note to be played and dur was the duration of the output note in seconds. First the sampling frequency was set to 11025 Hz and the time the signal would run for was calculated by creating a vector from 0 to the duration given stepping by the period of sampling frequency (1/sampling frequency). The frequency for the output signal was then found through the use of the formula 440*2^(*keynum-49)/12) where the 440 corresponds to the frequency of A_4, the 49 being its key number and the 12 being the number of keys in a chromatic scale across an octave. Finally the output waveform was created in the form of Re(X*e^(j*2*pi*f*t)).

The key2notejt function served the same purpose as the key2notewt function, but instead of well-tempered tuning, it used just-tempered tuning. Just-tempered tuning completely relies on ratios to a single note's frequency rather than using a formula. Once again the sampling frequency was set 11025 Hz and the time for the signal to run was created in a vector. The note that the just-tempered tuning method was based on was C_4 which had a key number of 40. Certain distances away from the root note would correspond to different ratios which could be used to calculate the new note's frequency. For example E_4 was 4 steps after C_4 which corresponded to a (25/16) ratio that would be multiplied by C_4's frequency (261.63 Hz) to find out it's frequency. The ratios were then reversed where the ratio corresponding to 1 step and 12 steps were swapped and 2 steps and 11 steps were swapped and etc. To find out any note's frequency that came before C_4, the root notes' frequency would be multiplied by the new ratio corresponding from the difference in key numbers and 1/i where i was how many octaves below the new note was from the root note. Finally the output waveform was created in the form of Re(X*e^(j*2*pi*f*t)).

The main function synthandplay that was used to synthesize the music was created last that implemented the two previous functions. Synthandplay would take in three parameters: songfile, fs and tuning and then output a sinusoidal waveform and play it through the computer's speakers. Songfile corresponded to the name of the *.mat file to read in, fs was the sampling frequency and tuning was used to determine if the function was to use well-tempered tuning or just-tempered tuning. Playback parameters were then created for future use which included bpm (beats per minute), beats_per_second, seconds_per_pulse and samples_per_pulse. Bpm is based off of what song was to be played and all the other parameters were then based off of eachother. The given .*mat file was then read in which creates the theVoices variable. The number of voices was calculated by finding the number of rows in the theVoices variable. Since theVoices variable followed a structure where it would use pulses instead of typical musical notation (beats), so the playback parameters needed to be used frequently to convert pulses into actual seconds. To generate the sinusoids to place in a "soundtrack" vector, first it was needed to find the last pulse and total duration of the piece to determine the size of the vector to store the sinusoids in. The size of the soundtrack was found by summing up all of the durations of the longest voice, multiplying that by samples_per_pulse and then adding the number of notes of the longest voice.

To synthesize the soundtrack, each note's sinusoidal waveform was created individually and then placed in the correct spot according to their starting pulse. A one pulse rest note was first created by calling the key2notewt function with duration of one pulse in terms of seconds. The function then cycled through all the voices and every note in each voice and produced a sinusoid pertaining to the characteristics of that single note. The place of each tone was calculated by looking at the different durations and start pulses of two consecutive notes and inserting rest pulse notes when they were needed. The final soundtrack variable was then inserted into the soundsc function to play the song from the speakers of the computer.

To make a more "colorful" and appealing sound, harmonics and envelopes were used. An envelope was created and applied within the envelope function. Envelopes usually follow a ADSR format which consists of an attack, delay, sustain and release. The attack tends to be a sharp rising edge, the delay is a short sudden drop, the sustain remains relatively constant and the release drops quickly back down to 0. All of these portions were generated using the linspace function and based the distance off the length of the input sinusoid. Harmonics were used to create a more realistic musical sound rather than a computer generated tone. Harmonics are different multiples of frequency of a certain tone. For example a 2nd harmonic would multiply the frequency of it's sinusoid by 2. Instead creating a separate function to add harmonics to the sinusoids, this was done within the key2notewt and key2notejt functions. New harmonics were created and added by making more sinusoids with their complex amplitude and frequencies scaled.

Results:

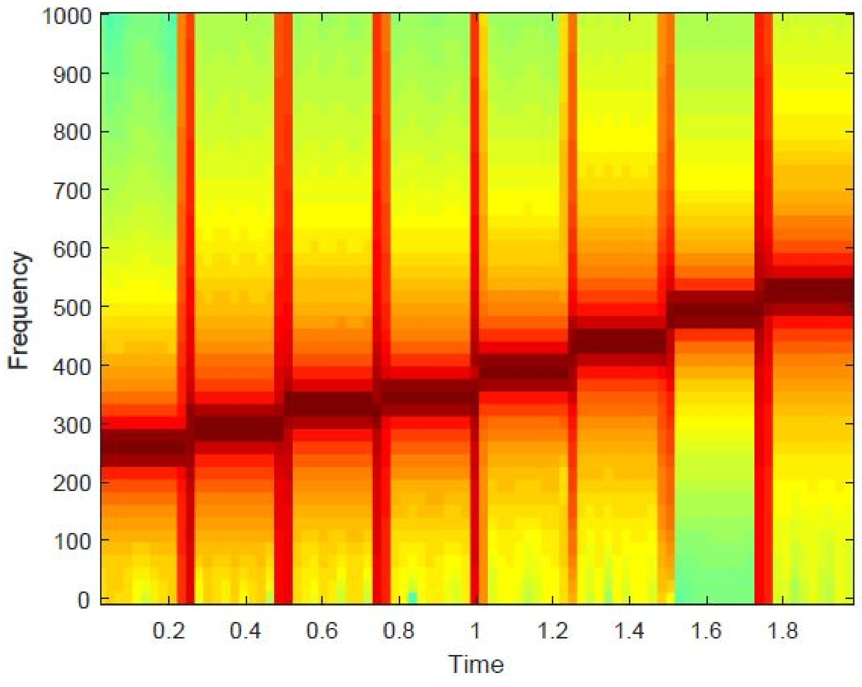

Figure 1. Spectrogram of a C major scale

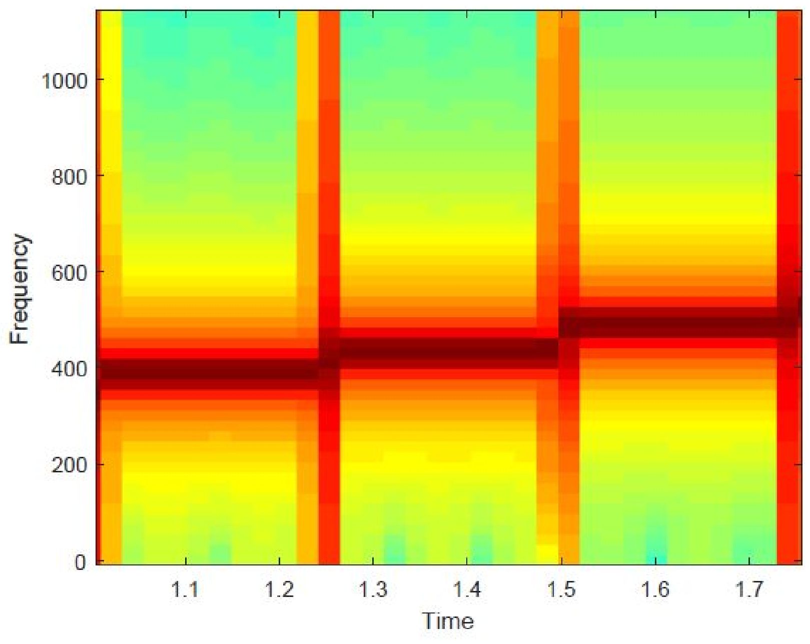

Figure 2. Spectrogram of 3 notes of a C major scale

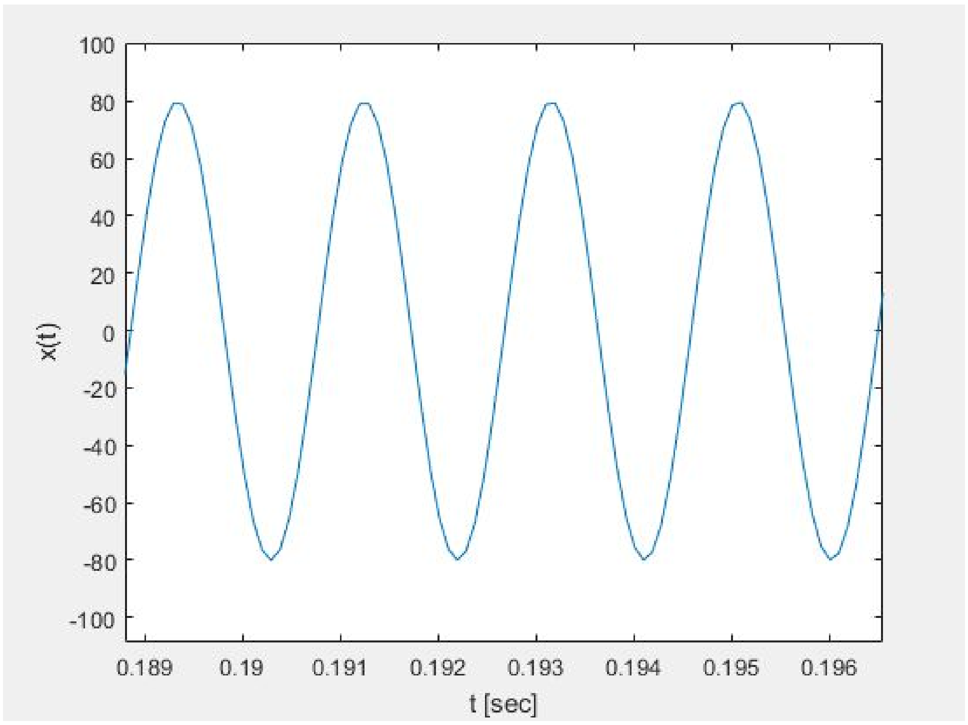

Figure 3. Sinusoid Plot for C_5 note

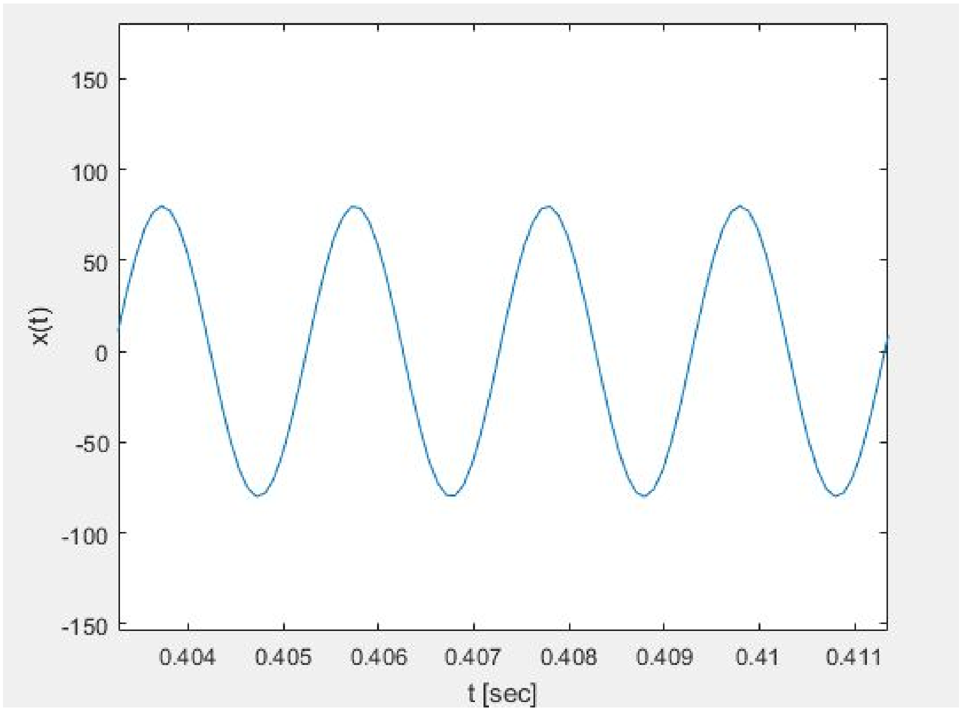

Figure 4. Sinusoid Plot for B_4 note

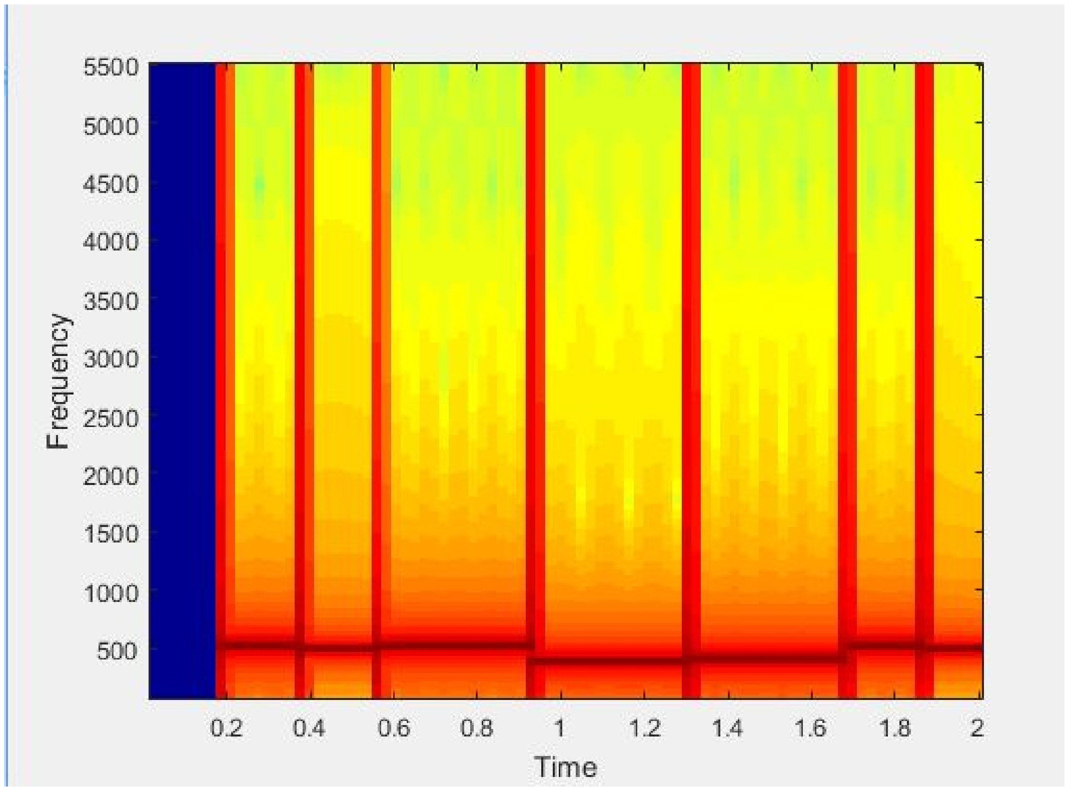

Figure 5. Spectrogram for 2 seconds without harmonics

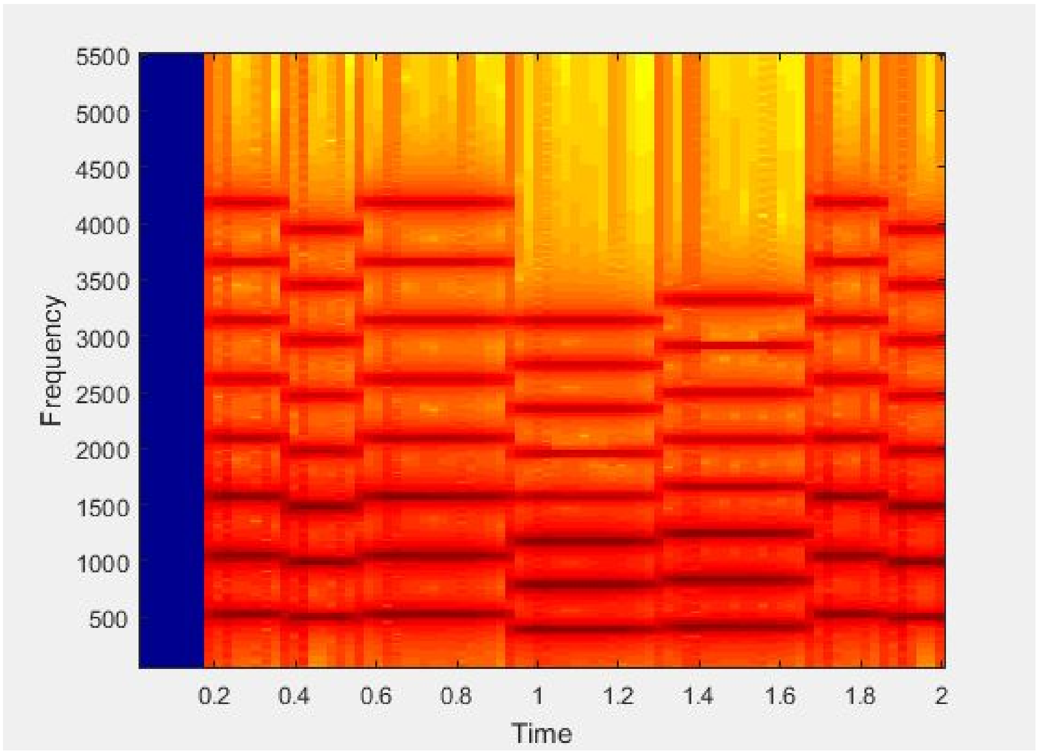

Figure 6. Spectrogram for 2 seconds with harmonics

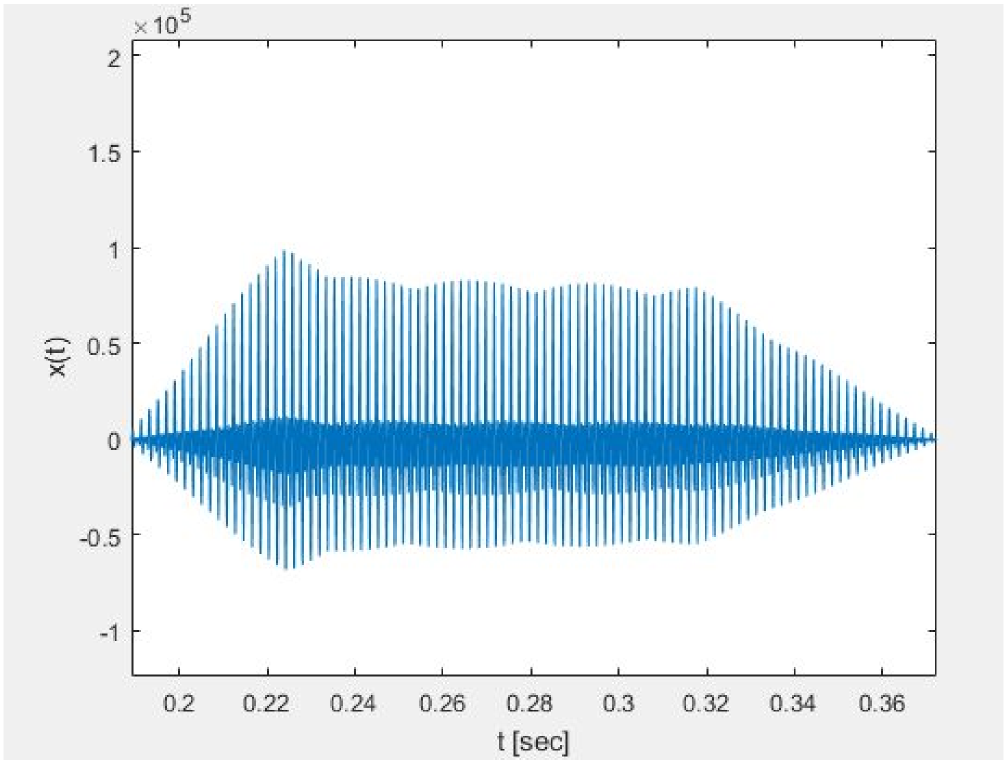

Figure 7. Sinusoid Plot for C_5 note with envelope

Figures 1 and 2 show the spectrogram of a C major scale. The C major scale starts from C_4 and goes up to C_5. C_4 has a frequency of approximately 216.63 Hz which can clearly be seen as the first red line in figure 1. The root note for key2notewt is A_4 which has a frequency of 440 Hz which can be seen from the middle red line in figure 2. Figures 3 and 4 are the sinusoid plots for the notes C_5 and B_4 which happen to be the first two notes of the fugue. Their respective frequencies are 525.35 Hz and 490.55-493.88 Hz (depending on tuning). The periods of these signals can be found by dividing 1 by their frequencies which come out to be .0019 and .002 seconds. Since these are very close, it is quite hard to see the difference of the two signals but the period does match up with what they should be. Figures 5 and 6 show a span of 2 seconds of the synthesized music where one has harmonics and one does not. In figure 5, it is clearly seen that there are no added harmonics by the way that there is only one stream of red lines. While when you look at figure 6, there are multiple streams of red lines that correlate with the different harmonics being added. Finally in figure 7, it displays a sinusoid plot for C_5 when an envelope is used. The shape of the envelope is clearly seen with the attack being the first rising edge, the delay being the slight drop, the sustain being the almost completely horizontal portion and the release being the final drop back to 0.

Appendix:

```
function xx = key2notewt(X, keynum, dur)
% KEY2NOTEWT Produce a sinusoidal waveform corresponding to a
% given piano key number in the well-tempered tuning
%
% usage: xx = key2notewt(X, keynum, dur)
%
% xx = the output sinusoidal waveform
% X = complex amplitude for the sinusoid, X = A*exp(j*phi).
% keynum = the piano keyboard number of the desired note
% dur = the duration (in seconds) of the output note
%
fs = 11025;
tt = 0:(1/fs):dur;
freq = 440*2^((keynum-49)/12);

% Adding in harmonics
xx =real((X*3*exp(j*2*pi*freq*tt)) + (X*5*exp(j*2*pi*freq*tt*2)) ...
   + (X*5*exp(j*2*pi*freq*tt*3)) + (X/2*exp(j*2*pi*freq*tt*4)) + ...
   (X/6*exp(j*2*pi*freq*tt*5)) + (X/6*exp(j*2*pi*freq*tt*6)) + ...
   (X/8*exp(j*2*pi*freq*tt*7)) + (X/8*exp(j*2*pi*freq*tt*8)));
```

Listing 1. Key2notewt.m Code

```
function xx = key2notejt(X, keynum, dur)
% KEY2NOTEWT Produce a sinusoidal waveform corresponding to a
% given piano key number in the well-tempered tuning
%
% usage: xx = key2notewt(X, keynum, dur)
%
% xx = the output sinusoidal waveform
% X = complex amplitude for the sinusoid, X = A*exp(j*phi).
% keynum = the piano keyboard number of the desired note
% dur = the duration (in seconds) of the output note
%
newnum = 0;
modnum = 0;
i = 1;
ratio = 0;
fs = 11025;
tt = 0:(1/fs):dur;
newnum = keynum - 40;
```

```
if(newnum >= 0)
    modnum = mod(newnum,12);
    while(newnum >= 12)
        modnum = mod(newnum,12);
        newnum = newnum - 12;
        i = i + 1;
    end

    switch modnum
    case 0
        ratio = 1;
    case 1
        ratio = (25/24);
    case 2
        ratio = (9/8);
    case 3
        ratio = (6/5);
    case 4
        ratio = (5/4);
    case 5
        ratio = (4/3);
    case 6
        ratio = (45/32);
    case 7
        ratio = (3/2);
    case 8
        ratio = (25/16);
    case 9
        ratio = (5/3);
    case 10
        ratio = (9/5);
    case 11
        ratio = (15/8);
    case 12
        ratio = 2;
    end

    freq = 261.63*ratio*i;
    xx =real((X*3*exp(j*2*pi*freq*tt)) + (X*5*exp(j*2*pi*freq*tt*2)) ...
    + (X*5*exp(j*2*pi*freq*tt*3)) + (X/2*exp(j*2*pi*freq*tt*4)) + ...
    (X/6*exp(j*2*pi*freq*tt*5)) + (X/6*exp(j*2*pi*freq*tt*6)) + ...
    (X/8*exp(j*2*pi*freq*tt*7)) + (X/8*exp(j*2*pi*freq*tt*8)));
else
```

```
  i = 2;
  modnum = mod(abs(newnum),12);
  while(abs(newnum) >= 12)
  modnum = mod(abs(newnum),12);
  newnum = abs(newnum) - 12;
  i = i + 1;
  end

  switch modnum
  case 0
      ratio = 2;
  case 1
      ratio = (15/8);
  case 2
      ratio = (9/5);
  case 3
      ratio = (5/3);
  case 4
      ratio = (25/16);
  case 5
      ratio = (3/2);
  case 6
      ratio = (45/32);
  case 7
      ratio = (4/3);
  case 8
      ratio = (5/4);
  case 9
      ratio = (6/5);
  case 10
      ratio = (9/8);
  case 11
      ratio = (25/24);
  case 12
      ratio = 1;
  end

  freq = 261.63*ratio*(1/i);
  xx =real((X*3*exp(j*2*pi*freq*tt)) + (X*5*exp(j*2*pi*freq*tt*2)) ...
  + (X*5*exp(j*2*pi*freq*tt*3)) + (X/2*exp(j*2*pi*freq*tt*4)) + ...
  (X/6*exp(j*2*pi*freq*tt*5)) + (X/6*exp(j*2*pi*freq*tt*6)) + ...
```

```
    (X/8*exp(j*2*pi*freq*tt*7)) + (X/8*exp(j*2*pi*freq*tt*8)));
end
```

Listing 2. Key2notejt.m Code

```
function xx = synthandplay(songfile, fs, tuning)

% Synthesize and Play a song encoded in the theVoices
% given piano key number in the well-tempered tuning or
% just-tempered tuning
%
% usage: xx = synthandplay(songfile, fs, tuning)
%
% xx = the output sinusoidal waveform
% songfile = the name of the *.mat file to read from
% fs = sampling frequecy
% tuning = 1 for well-tempered tuning, 0 for just-tempered tuning
%

% Set default values
if nargin == 0
    songfile = 'BWV847_Fugue2_Cminor'; % Default song
    fs = 11025; % default sampling frequency
    tuning = 1;
elseif nargin == 1
    fs = 11025; % default sampling frequency
    tuning = 1;
elseif nargin == 2
    tuning = 1;
end
```

Playback parameters

```
bpm = 80;
beats_per_second = bpm/60;
seconds_per_beat = 1/beats_per_second;
seconds_per_pulse = seconds_per_beat/4;
samples_per_pulse = fs * seconds_per_pulse;
```

Load music file and find number of voices

```
eval(['load' ' ''' songfile '.mat''']);
number_of_voices = size(theVoices,2);
```

Process theVoices to generate sinusoids

1. Find last pulse and total duration of piece

```
for vloop = 1:number_of_voices
  % find # of notes per voice
  number_of_notes(vloop) = length(theVoices(vloop).noteNumbers);
  if number_of_notes(vloop)> 0
    last_pulse(vloop) = theVoices(vloop).startPulses(number_of_notes(vloop));
    last_duration(vloop) = theVoices(vloop).durations(number_of_notes(vloop));
  end
end
```

2. Synthesize soundtrack one voice at a time (mono version)

Find size of vector to hold all notes

```
size_of_soundtrack = (sum(theVoices(1).durations*samples_per_pulse*1.08)...
             + number_of_notes(1));

soundtrack = zeros(1, round(size_of_soundtrack));

% Synthesize the notes, one voice at a time
fprintf('synthesizing %d voices\n', number_of_voices);
dur = seconds_per_pulse;
one_pulse_rest_note = key2notewt(0,0,dur);

for vloop = 1:number_of_voices % voices loop
  fprintf('voices %d\n', vloop);
  n1 = 1;
  for nloop = 1:number_of_notes(vloop) % notes loop
    % Find duration and key number
    dur = (theVoices(vloop).durations(nloop))*seconds_per_pulse;
    keynum = theVoices(vloop).noteNumbers(nloop);
```

```
% Generate waveform, include harmonics and ADSR
if tuning == 1
    note = key2notewt(80*exp(j*pi/4),keynum,dur);
elseif tuning == 0
    note = key2notejt(80*exp(j*pi/4),keynum,dur);
end
enveloped_note = envelope(note);

% Compute where to insert tone
place_of_tone = theVoices(vloop).startPulses(nloop);
if nloop == 1
    if place_of_tone ~= 1
        delay = place_of_tone-1;
        for loop = 1:delay
            n2 = n1 + length(one_pulse_rest_note) - 1;
            soundtrack(n1:n2) = soundtrack(n1:n2) + one_pulse_rest_note;
            n1 = n2 + 1;
        end
        n2 = n1 + length(enveloped_note) - 1;
        soundtrack(n1:n2) = soundtrack(n1:n2) + enveloped_note;
        n1 = n2 + 1;
    else
        n2 = n1 + length(one_pulse_rest_note) - 1;
        soundtrack(n1:n2) = soundtrack(n1:n2) + one_pulse_rest_note;
        n1 = n2 + 1;
    end
else
    % insert voices in soundtrack, in the right place
    pulse_difference = place_of_tone - ...
                       theVoices(vloop).startPulses(nloop-1);

    rest_pulses_needed = pulse_difference - ...
                         round(theVoices(vloop).durations(nloop-1));

    if rest_pulses_needed > 0
       for loop = 1:rest_pulses_needed
           n2 = n1 + length(one_pulse_rest_note) - 1;
           soundtrack(n1:n2) = soundtrack(n1:n2) + one_pulse_rest_note;
           n1 = n2 + 1;
       end
       n2 = n1 + length(enveloped_note) - 1;
       soundtrack(n1:n2) = soundtrack(n1:n2) + enveloped_note;
```

```
              n1 = n2 + 1;
           end
%
%           if rest_pulses_needed < 0
%               n1 = n1 - length(one_pulse_rest_note)*rest_pulses_needed;
%               n2 = n1 + length(enveloped_note) - 1;
%               soundtrack(n1:n2) = soundtrack(n1:n2) + enveloped_note;
%               n1 = n2 + 1 + length(one_pulse_rest_note)*rest_pulses_needed;
%           end
%
           if rest_pulses_needed == 0
               n2 = n1 + length(enveloped_note) - 1;
               soundtrack(n1:n2) = soundtrack(n1:n2) + enveloped_note;
               n1 = n2 + 1;
           end

       end

    end
end % for number_of_voices

fprintf('soundtrack done!\n')
```

Playback in speaker

```
soundsc(soundtrack, fs);
```

Listing 3. Synthandplay.m Code

```
function xx = envelope(yy)
% Envelope takes in a signal and applies and envelope to the
% given signal
%
% usage: xx = envelope(yy)
%
% xx = the output sinusoidal waveform
% yy = the signal to be multiplied
%
length_of_input = length(yy);
xx = zeros(1,length_of_input);
A = linspace(0,100,round((2*length_of_input)/10));   % Attack
D = linspace(100,80,round((0.5*length_of_input)/10));  % Delay
```

```
S = linspace(80,75,round((4.5*length_of_input)/10));    % Sustain
R = linspace(75,0,round((3*length_of_input)/10));     % Release

% Combine all parts
count = 1;
for vloop = 1:round((2*length_of_input)/10)
   xx(count) = A(vloop);
   count = count + 1;
end

count = round((2*length_of_input)/10);
for vloop = 1:round((0.5*length_of_input)/10)
   xx(count) = D(vloop);
   count = count + 1;
end

count = round((2.5*length_of_input)/10);
for vloop = 1:round((4.5*length_of_input)/10)
   xx(count) = S(vloop);
   count = count + 1;
end

count = round((7*length_of_input)/10);
for vloop = 1:round((3*length_of_input)/10)
   xx(count) = R(vloop);
   count = count + 1;
end
xx = xx.*yy;
```

Listing 4. Harmonics.m Code