```
diary on
format compact
%Johnny Li
%EEL3135 Fall 2018
%Lab 3 Part 2

%2.1
type play_song
function song = playSong(theVoices)
%PLAY_SONG Construct the three voices in the Barukh Fugue and add them
%together. Add the five voices, from better_fugue.mat, together to produce
%the Better Fugue. Synthesisze the bach_fugue.
%Function based on given code 1.6.

%{
    PLAYSONG: Produce a sinusoidal waveform containing the combination of
    the different notes in theVoices
    Input Args:
        theVoices: structure contains noteNumbers, durations, and
        startpulses vectors for multiple voices of a song.
    Output:
        song: vector that represents discrete-time version of a musical
        waveform
    Usage:
        song = playSong()
%}
%load barukh_fugue.mat
load bach_fugue.mat
%Define variables
%Frequency
fs = 8000;
%Beat per minute->beats per second->second per beats->second per pulse
%Given Code
beats_per_minute  = 120;
beats_per_second  = beats_per_minute / 60;
seconds_per_beat  = 1 / beats_per_second;
%spp = seconds_per_beat / 4;
%seconds per pulse, theVoices is measured in pulses with 4 pulses per beat

%Set spp to 0.15 for better fugue
spp=0.15;

%Length of voices
numV=length(theVoices);
%Length of notes
numN=length(theVoices(numV).noteNumbers);
%Final start pulse
fsp=theVoices(numV).startPulses(numN);
%Final durations
fd=theVoices(numV).durations(numN);
song = zeros(1,ceil((fsp+fd)*spp*fs));

%Get Max value in theVoices
M=0;
for a=1:numV
    for b=1:length(theVoices(a).durations)
```

```
        d=theVoices(a).durations(a);
        st=theVoices(a).startPulses(b);
        if M<(d+st)
            M=d+st+1;
        end
    end
end

%Longest value in better
song = zeros(1,ceil(M*spp*fs));
%Create a vector of zeros with length equal to the total number of samples
%in the entire song

%Then add in the notes
for i = 1:length(theVoices)
    for j = 1:length(theVoices(i).noteNumbers)
         note =
key_to_note(1,theVoices(i).noteNumbers(j),theVoices(i).durations(j)*spp);
         %Create sinusoid of correct length to represent a single note
         locstart = theVoices(i).startPulses(j)*spp*fs;
         %Index of where note starts
         locend = locstart+length(note)-1;
         %Index of where note ends
         song(locstart:locend) = song(locstart:locend) + note;
    end
%For clipping
song=song/(max(abs(song)));
end
%Create autofile
audiowrite('bach_fugue1.wav',song,fs);
end

%Play the bach fugue
play_song(theVoices);

%2.2
type envBach

%EnvBach Synthesisze the bach_fugue and implement an envelope to improve
%the sound quality.
%Script for 2.2.

%From play_song function
%{
    PLAYSONG: Produce a sinusoidal waveform containing the combination of
    the different notes in theVoices
    Input Args:
        theVoices: structure contains noteNumbers, durations, and
        startpulses vectors for multiple voices of a song.
    Output:
        song: vector that represents discrete-time version of a musical
        waveform
    Usage:
        song = playSong()
%}
%load barukh_fugue.mat
```

```
load bach_fugue.mat
%Define variables
%Frequency
fs = 8000;
%Beat per minute->beats per second->second per beats->second per pulse
%Given Code
beats_per_minute  = 120;
beats_per_second  = beats_per_minute / 60;
seconds_per_beat  = 1 / beats_per_second;
%spp = seconds_per_beat / 4;
%seconds per pulse, theVoices is measured in pulses with 4 pulses per beat

%Set spp to 0.15 for better fugue
spp=0.15;

%Get Max value in theVoices
M=0;
for a=1:length(theVoices)
    for b=1:length(theVoices(a).durations)
        d=theVoices(a).durations(a);
        st=theVoices(a).startPulses(b);
        if M<(d+st)
            M=d+st+1;
        end
    end
end

%Longest value
song = zeros(1,ceil(M*spp*fs));
%Create a vector of zeros with length equal to the total number of samples
%in the entire song

%Then add in the notes
for i = 1:length(theVoices)
    for j = 1:length(theVoices(i).noteNumbers)
         note =
key_to_note(1,theVoices(i).noteNumbers(j),theVoices(i).durations(j)*spp);

         %Create sinusoid of correct length to represent a single note
         locstart = theVoices(i).startPulses(j)*spp*fs;
         %Index of where note starts
         locend = locstart+length(note)-1;

        %Create modify notes
        %Length based on estimatation of graph
        A = linspace(0, 1, 0.1286*length(note)); %Rise
        D = linspace(1, 0.77, 0.0714*length(note)); %Drop slightly
        S = linspace(0.77, 0.7, 0.6143*length(note)); %steady decline
        R = linspace(0.7, 0, 0.1857*length(note)); %Rest

        %Concatenate note
        ADSR = [A D S R zeros(1,length(note)-length(A)-length(D)-length(S)-
length(R))];

        %Envelop note
```

```
        note=note.*ADSR;

         %Index of where note ends
         song(locstart:locend) = song(locstart:locend) + note;
    end
%For clipping
song=song/(max(abs(song)));

%Create autofile
audiowrite('envBach.wav',song,fs);
end

%Play the bach fugue with envelope
envBach

%2.3
%Question: Suppose the maximum frequency in the Bach Fugue is 1200 Hz. What
%is the minimum sampling frequency needed to synthesize, without aliasing,
%a trumpet sound containing nine harmonics?
%The minimum sampling frequency needed to synthesize, without aliasing, is
%double of the principle harmonic frequency which is (1200/9)*2= 267 Hz.

%Using a sampling frequency of 44100 Hz, construct your Bach Fugue in
%trumpet.
%Function to convert key to trumpet note
type key_to_trumpet

function x = key_to_trumpet(keynum,dur)
%KEY_TO_TRUMPET Transfer the key value into a trumpet note. For lab 3.
%Using fourier series.
%Function written for 2.3.

%Sampling Frequency
fs=44100;
%Frequency given
F=440*(2^((keynum-49)/12));
%Amplitude
amp=[0.1155 0.3417 0.1789 0.1232 0.0678 0.0473 0.0260 0.0045 0.0020];
%Phase
p=[-2.1299 1.6727 -2.5454 0.6607 -2.0390 2.1597 -1.0467 1.8581 -2.3925];
%Time interval
tt=1/fs:1/fs:dur;

%fourier series of trumpet
x=0;
for w=1:9
    x=x+amp(w)*cos(2*pi*F*w*tt-p(w));
end

end

type TBach

%TBach Using a sampling frequency of 44100 Hz, construct your Bach Fugue in
%trumpet.
```

```
%Script for 2.3.

%From play_song function
%{
    PLAYSONG: Produce a sinusoidal waveform containing the combination of
    the different notes in theVoices
    Input Args:
        theVoices: structure contains noteNumbers, durations, and
        startpulses vectors for multiple voices of a song.
    Output:
        song: vector that represents discrete-time version of a musical
        waveform
    Usage:
        song = playSong()
%}
load bach_fugue.mat

%Frequency
fs = 44100;

%Beat per minute->beats per second->second per beats->second per pulse
%Given Code
beats_per_minute  = 120;
beats_per_second  = beats_per_minute / 60;
seconds_per_beat  = 1 / beats_per_second;
%spp = seconds_per_beat / 4;
%seconds per pulse, theVoices is measured in pulses with 4 pulses per beat

%Set spp to 0.15
spp=0.15;

%Get Max value in theVoices
M=0;
for a=1:length(theVoices)
    for b=1:length(theVoices(a).durations)
        d=theVoices(a).durations(a);
        st=theVoices(a).startPulses(b);
        if M<(d+st)
            M=d+st+1;
        end
    end
end

%Longest value in better
song = zeros(1,ceil((M+1)*spp*fs));
%Create a vector of zeros with length equal to the total number of samples
%in the entire song

%Then add in the notes
for i = 1:length(theVoices)
    for j = 1:length(theVoices(i).noteNumbers)
         %note =
key_to_note(1,theVoices(i).noteNumbers(j),theVoices(i).durations(j)*spp);
         %Use new function of key_to_trumpet)
```

```
note=key_to_trumpet(theVoices(i).noteNumbers(j),theVoices(i).durations(j)*spp
);

         %Create sinusoid of correct length to represent a single note
         locstart = theVoices(i).startPulses(j)*spp*fs;
         %Index of where note starts
         locend = locstart+length(note)-1;

        %Create modify notes
        %Length based on estimatation of graph
        A = linspace(0, 1, 0.1286*length(note)); %Rise
        D = linspace(1, 0.77, 0.0714*length(note)); %Drop slightly
        S = linspace(0.77, 0.7, 0.6143*length(note)); %steady decline
        R = linspace(0.7, 0, 0.1857*length(note)); %Rest

        %Concatenate note
        ADSR = [A D S R zeros(1,length(note)-length(A)-length(D)-length(S)-
length(R))];

        %Envelop note
        note=note.*ADSR;

         %Index of where note ends
         song(locstart:locend) = song(locstart:locend)+ note;
    end
%For clipping
song=song/(max(abs(song)));

%Create autofile
audiowrite('TBach.wav',song,fs);
end
```