Lab 1 Part 2

Table of Contents

2.1 - Construction of the Bach Fugue

```
load bach_fugue.mat
song = playSong(theVoices);
audiowrite('bach_fugue.wav',song/max(song),11024);
```

2.2 - Musical Tweaks - Enveloping

```
type ADSR
type playSongWithADSR

load bach_fugue.mat
song = playSongWithADSR(theVoices);
audiowrite('bach_fugue_ADSR.wav',song/max(song),11024);


function tone = ADSR(tone)
    %{
    ADSR: Apply an envelope to a particular note.
    Input Args:
    -tone: the tone to apply the ADSR envelope to.
    Output:
    -tone: the tone that was supplied as a paramater, with the ADSR
    envelope.
    Usage:
    envelopedTone = ADSR(tone);
    %}

    A = linspace(0.0, 0.9, (length(tone)*0.25)); % rise 25%
    D = linspace(0.9, 0.7, (length(tone)*0.05)); % drop 5%
    S = linspace(0.7, 0.7, (length(tone)*0.40)); % maintain 40%
    R = linspace(0.7, 0.0, (length(tone)*0.30)); % drop 30%
    ADSR = [A D S R];

    x = zeros(size(tone));
    x(1:length(ADSR)) = ADSR;
    tone = tone .* x;
end

function song = playSongWithADSR(theVoices)
    %{
```

```
   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.
   Output:
   -song: vector that represents discrete-time version of a musical
waveform
   Usage: song = playSong()
   %}

   fs = 11024;
   bpm = 120;
   bps = bpm / 60;
   spb = 1 / bps;
   spp = spb / 4; %seconds per pulse, theVoices is measured in pulses
with 4 pulses per beat

   maxIndex = 1;
   for i = 1:length(theVoices)
       lastNoteNumber = length(theVoices(i).noteNumbers);
       lastNote = key_to_note(0.5,
theVoices(i).noteNumbers(lastNoteNumber),
theVoices(i).durations(lastNoteNumber)*spp);
       currentIndex = spp*fs*theVoices(i).startPulses(lastNoteNumber)
+ length(lastNote) - 1;
       if currentIndex > maxIndex
           maxIndex = currentIndex;
       end
   end

   song = zeros(1, maxIndex); %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)
           keynum = theVoices(i).noteNumbers(j);
           dur = theVoices(i).durations(j)*spp;
           note = ADSR(key_to_note(0.5, keynum, dur)); %Create
sinusoid of correct length to represent a single note
           locstart = spp*fs*theVoices(i).startPulses(j); % Index of
where note starts
           locend = locstart + length(note) - 1; % index of where
note ends
           song(locstart:locend) = song(locstart:locend) + note;
       end
   end
   song = song/max(song);
   soundsc(song, fs);
end
```

2.3 - Musical Tweaks - Fourier Series of a Trumpet

```
type playSongWithTrumpet
type key_to_trumpet_note

load bach_fugue.mat
song = playSongWithTrumpet(theVoices);
audiowrite('bach_fugue_trumpet_ADSR.wav',song/max(song),44100);


function song = playSongWithTrumpet(theVoices)
    %{
    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.
    Output:
    -song: vector that represents discrete-time version of a musical
 waveform
    Usage: song = playSong()
    %}

    fs = 44100;
    bpm = 120;
    bps = bpm / 60;
    spb = 1 / bps;
    spp = spb / 4; %seconds per pulse, theVoices is measured in pulses
 with 4 pulses per beat

    maxIndex = 1;
    for i = 1:length(theVoices)
        lastNoteNumber = length(theVoices(i).noteNumbers);
        lastNote =
 key_to_trumpet_note(theVoices(i).noteNumbers(lastNoteNumber),
 theVoices(i).durations(lastNoteNumber)*spp);
        currentIndex = spp*fs*theVoices(i).startPulses(lastNoteNumber)
 + length(lastNote) - 1;
        if currentIndex > maxIndex
            maxIndex = currentIndex;
        end
    end

    song = zeros(1, ceil(maxIndex)); %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)
            keynum = theVoices(i).noteNumbers(j);
            dur = theVoices(i).durations(j)*spp;
```

```
            note = ADSR(key_to_trumpet_note(keynum, dur)); %Create
 sinusoid of correct length to represent a single note
            locstart = spp*fs*theVoices(i).startPulses(j); % Index of
 where note starts
            locend = locstart + length(note) - 1; % index of where
 note ends
            % floor is included here to remove the warning "Integer
 operands are required for colon operator when used as index."
            song(floor(locstart):floor(locend)) =
 song(floor(locstart):floor(locend)) + note;
        end
    end
    song = song/max(song);
    soundsc(song, fs);
end

function xx = key_to_trumpet_note(keynum, dur)
    %{
    KEY_TO_TRUMPET_NOTE: Produce a sinusoidal waveform corresponding
 to a given piano key number
    Input Args:
    -X: amplitude (default = 1)
    -keynum: number of the note on piano keyboard -dur: duration of
 the note (in seconds)
    Output:
    -xx: sinusoidal waveform of the note
    %}
    harmonics.amplitudes = [0.1155, 0.3417, 0.1789, 0.1232, 0.0678,
 0.0473, 0.0260, 0.0045, 0.0020];
    harmonics.phases = [-2.1299, 1.6727, -2.5454, 0.6607, -2.0390,
 2.1597, -1.0467, 1.8581, -2.3925];

    fs = 44100;
    tt = 0:(1/fs):dur-1/fs;
    freq = 440 * (   2^( (keynum-49)/12 )   );
    xx = zeros(1,length(tt));

    for k = 1:length(harmonics.amplitudes)
        A = harmonics.amplitudes(k);
        phi = harmonics.phases(k);
        xx = xx + real( A*exp(j*2*pi*freq*k*tt)*exp(j*phi) );
    end
end
```

```
Question 1: 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?

According to the Nyquist Theorem, we need a sampling frequency of at
least 1200 Hz * 2 + 1 = 2401 Hz to avoice aliasing.
```

Published with MATLAB® R2019a