Lab 3: Music Synthesis with Sinusoidal Signals

Introduction

Sinusoids are used in a wide variety of applications. Perhaps the most obvious one is in music. A musical note is a simple sinusoid of the form $Acos(2\pi ft + \varphi)$, with amplitude A, frequency f, and phase φ. A melody, or musical voice, is composed of an ordered list of notes and rests. A musical composition is, simply put, a sum of melodies. Every pulse in a musical composition is therefore a sum of sinusoids of the form:

$$x(t) = \sum_k A_k cos(\omega_k t + \varphi_k)$$

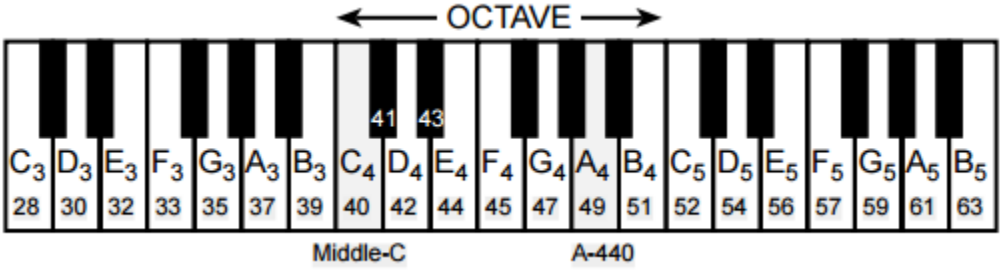
In this lab, you will synthesize some musical pieces, including *Mary Had a Little Lamb*, *Fugue #2 for the Well-Tempered Clavier* by Bach, *Hava Nashira* by Barukh, and *Treat You Better* by Shawn Mendes. Because this is a rather complicated programming assignment, you should start as soon as possible and take advantage of all class sessions and TA office hours.

Debugging tools can be very helpful when troubleshooting larger programs like the one you will write in this lab. To learn about MATLAB debugging tools (you don't need to submit your help results), type `help debug` in the command window.

Lab Part One

1.1 A Function to Play a Note

Piano keyboards are laid out as illustrated by the following image:



- C_4 refers to the C-key in the fourth octave. C_4 is also called middle-C.
- A_4 is also called A-440, because its frequency is 440 Hz.
- Every key is given a key number. The key number of middle-C is 40.
- The frequency of any key can be found by substituting its key number into the formula:

$$f = 440(2^{\frac{key\ number-49}{12}})$$

Remember that when we construct a sinusoid through MATLAB, we are in fact constructing a sampled sinusoid

$$x[n] = x(nT_s)$$

The Sampling Theorem tells us that to recover a continuous signal from its samples without aliasing, f_s must be at least two times greater than the largest frequency component. `soundsc(xx, fs)` will output sound recovered from the sample vector `xx` at a rate of f_s samples per second. Use this to judge how your synthesis is doing as you work on the lab. In music synthesis, common choices for sampling frequency f_s are 8000 Hz, 11025 Hz, or multiples of 11025.

Write a MATLAB function to produce a desired note for a given duration at a given complex amplitude. Use the following skeleton code to write your function:

```
function xx = key2note(X, keynum, dur)
      % KEY2NOTE Produce a sinusoidal waveform corresponding to a given
        piano key number
      % usage: xx = key2note (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 = 8000;
      tt = (1/fs):(1/fs):dur;
      freq = %<================ fill in this line
      xx = real(    ); %<================ fill in this line
end
```

For the `freq=` line, use the formula given above to determine the frequency for a sinusoid in terms of its key number. You should start from a reference note (middle-C or A-440 is recommended) and solve for the frequency based on this reference. Fill in the `xx = real()` line to generate the actual sinusoid as the real part of a complex exponential at the proper frequency.

1.2 Synthesize a Song – Mary Had a Little Lamb that NEVER grew up!

Multiple notes can be played in order by concatenating their row vectors as follows:

```
xx = [x1 x2];
```

Use `glottalkey2note()` to write a script, `play_glottallamb.m`, that plays a series of notes. Use the following skeleton code to write your script:

```
% --------------play_lamb.m-------------- %
mary.keys = [44 42 40 42 44 44 44 42 42 42 44 47 47];
% NOTES: C D E F G
% Key #40 is middle-C
mary.durations = 0.25 * ones(1,length(mary.keys));
fs = 8000;  % 11025 Hz also works
xx = zeros(1, sum(mary.durations)*fs + length(mary.keys));
n1 = 1;
for kk = 1:length(mary.keys)
      keynum = mary.keys(kk);
      tone =  % <------- Fill in this line
      n2 = n1 + length(tone) - 1;
      xx(n1:n2) = xx(n1:n2) + tone; %<------- Insert the note
      n1 = n2 + 1;
end
soundsc(xx, fs)
```

a. Generate the sound and play it for a TA. Have your TA check you off for this section. (If you can't finish this in time for the TA to check it off, submit it as a *.wav* file.)
b. Plot the frequency-time spectrogram of Mary using the function `specgram(xx, 512, fs)`. Notice that you can zoom in to show the order of notes. Try varying the window length (the second function argument) to get different looking spectrograms if needed. (Note; specgram doesn't work for some students; if this is the case, use "plotspec" instead).

1.3 Structures

Now that you've received an introduction to musical notes, and you've played a scale, you're going to play something a little more complicated. MATLAB allows for structures, which group information together. A structure, which is a data type, is used to represent information about something more complicated than what can be held by a single number, character, or boolean or an array of any one of them. For example, a Student can be defined by his or her name (an array of characters), GPA (a double), age (an integer), UFID (a long integer), and more. Each of these pieces of information can be labeled with an easily understood descriptive title, and then combined to form a whole (the structure).

Structures give us a way to "combine" multiple types of information under a single variable. The nice thing about structures is that they allow us to use human-readable descriptions for our data.For example, every note in a melody can be characterized as a list of notes, each one of

which starts at a specified pulse, and is played for a specified duration. We can define three separate arrays for one melody:

```
melody_noteNumbers    = [40 42 44 45 47 49 51 52];
melody_durations      = [1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5];
melody_startPulses    = [1 3 5 7 9 11 13 15];
```

But it is generally better to define a melody as a structure containing three fields, denoted as

```
structurename.fieldname:
melody.noteNumbers    = [40 42 44 45 47 49 51 52];
melody.durations      = [1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5];
melody.startPulses    = [1 3 5 7 9 11 13 15];
```

The structure is now a type, of which there can now be vectors:

```
melody(1).noteNumbers = [40 42 44 45 47 49 51 52];
melody(1).durations   = [1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5];
melody(1).startPulses = [1 3 5 7 9 11 13 15];
melody(2).noteNumbers = [40 42 44 45 47 49 51 52];
melody(2).durations   = [1.5 1.5 1.5 1.5 1.5 1.5 1.5 1.5];
melody(2).startPulses = [1 3 5 7 9 11 13 15];
```

This is almost ALWAYS a better way to go then to use a group of arrays or even a cell array.To learn more about structures, examine and run the following code:

```
x.Amp = 7;
x.phase = -pi/2;
x.freq = 100;
x.fs = 11025
x.timeInterval = 0:(1/x.fs):0.05;
x.values = x.Amp*cos(2*pi*(x.freq)*(x.timeInterval) + x.phase);
x.name = 'My Signal';
x     %---- echo the contents of the structure "x"
plot(x.timeInterval, x.values)
title(x.name)
```

1.4 The Evenly-Timed First Voice

Now that you've received an introduction to musical notes, and you've played a scale, you're going to play something a little more complicated.

You have been given a data file called `barukh_fugue.mat`, which contains an array of structures called `theVoices`, a structure array of the same form as melody above. `theVoices` is composed of three voices, each voice of which is characterized by three vectors: `durations`, `noteNumbers`, and `startPulses`.The first voice's notes are accessed as the vector

```
theVoices(1).noteNumbers
```

Modify your `play_mary.m`code to create a new function, `play_firstvoice_even.m`, to play each note in the first voice of the BarukhFugue for 0.5 seconds each. Have your TA check this off. If you can't finish this in time, submit as a *.wav* file on Canvas.

1.5 The Correctly-Timed First Voice

Something sounded a little bit off with that, didn't it? That's because in the melody, the notes aren't meant to all be played for the exact same amount of time. Each note is meant to be played for a specified duration, or number of pulses. Provided that there are no intended breaks between notes, a melody can be constructed from a vector of `noteNumbers` and a vector of durations. These two vectors will have the same length; every

```
theVoices(1).noteNumbers(i)
```

will have a matching

```
theVoices(1).durations(i).
```

Modify your `play_firstvoice_even.m`code to create a new function, `play_firstvoice.m`, to play each note in the first voice for its correct duration of pulses, with each pulse being 0.25 seconds long. Again, have your TA check this off. If you can't finish this in time, submit as a *.wav* file on Canvas.

1.6 Silence and `startPulses`: Construction of the Better Fugue

Balance is an important part of music. Sometimes, multiple musical voices are simultaneously playing. But sometimes, a given voice is silent in favor of another voice. Sometimes, both voices are silent. We can denote silent periods as time after one note has ended, but before the next note has begun. Suppose a note of `noteNumber` 49 starts at pulse 13, and has duration 2; and the next note of `noteNumber` 52 starts at pulse 16 and has duration 1. Then the musical voice is to play Middle A for two pulses, it is to be silent for one pulse, and then it is to play High C for one pulse. Thus, by incorporating information about `theVoices(i).startPulses`, in addition to `theVoices(i)`.durations and `theVoices(i).noteNumbers`, we can incorporate silent periods into our musical reconstruction.

Using the `startPulses`, `durations`, and `noteNumbers` properties of each voice in `theVoices`, construct the three melodies in the Barukh Fugue and add them together.

The MATLAB data file `better_fugue.mat` is of a similar structure as `barukh_fugue.mat`, also containing a `theVoices` array of structures, but this time with five voices. Using your `barukh_fugue` code: Add the five voices, from `theVoices` in `better_fugue.mat`, together to produce the Better Fugue. Note that there may have been errors in the way that you produced the `barukh_fuguethat` aren't noticeable in the `better_fugue`; the `better_fugue` is a bit harder. Produce the Better Fugue, save it as a *.wav* file, and submit it on Canvas alongside your lab document. There is no password here - submit the *.wav* file. This ends Part 1 of this lab.

1.8 Some Help with the Above

The function you create will be similar to your `play_scale`file, with the exception that there will be multiple overlapping melodies and the notes will not all be the same duration. Because this is a rather complicated programming assignment, you should start as soon as possible and take advantage of all class sessions and TA office hours.

Remember to reference arrays in the structure. For example, if you wanted to reference the note numbers in the code to find the length, you would have to say

```
length(theVoices(i).noteNumbers)
```

The skeleton code for your function is provided below:

```
function song = playSong(theVoices)
% PLAYSONG Produce a sinusoidal waveform containing the
  combinationofthe different notes in theVoices
% usage: song = playSong ()
% song = the output sinusoidal waveform
loadbarukh_fugue.mat
fs = 8000;
spp = % seconds per pulse, theVoices is measured in pulses with 4
        pulsesper beat

% Create a vector of zeros with length equal to the total number
  ofsamples in the entire song
song = % vector of zeros
% Then add in the notes
     fori = 1:length(theVoices) % Cycle through each set of notes
     % Convert data arrays to appropriate units
          for j = 1:length(theVoices(i).noteNumbers) % Cycle througheach note
                                                       in aset
               note = % create sinusoid of correct length to represent a
                        single note
               locstart = % index of where note starts
               locend = % index of where note ends
               song(locstart:locend) = song(locstart:locend) + note;
          end
     end
% Use audiowrite() to generate WAV file
end
```

Timing (nothing to submit for this subsection)

The time duration of a single pulse will determine the speed of the song. This is typically specified in beats per minute. You can perform a series of conversions to convert beats per minute into seconds per pulse, given that there are 4 pulses per beat and `beats_per_minute = 120`.

```
beats_per_minute = 120;
beats_per_second = beats_per_minute / 60;
seconds_per_beat = 1 / beats_per_second;
seconds_per_pulse = seconds_per_beat / 4;
```

Don't forget to submit your PDF file, too!

Lab Part Two

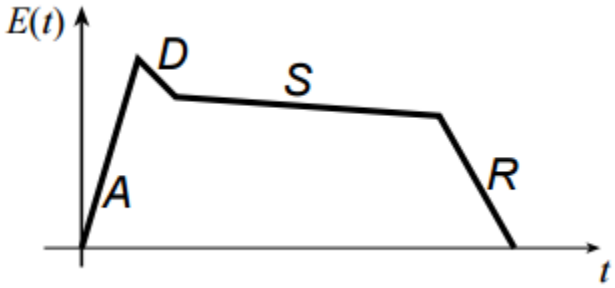
2.1 Construction of the Bach Fugue

The MATLAB data file `bach_fugue.mat` is of the same structure as `barukh_fugue.mat`, also containing a `theVoices` array of structures. Add the three voices from `theVoices` in `bach_fugue.mat`, together to produce the Bach Fugue. Have your TA check this off. (Alternately, if you can't finish this in time, submit as a *.wav* file on Canvas.)

2.2 Musical Tweaks - Enveloping

After creating the `barukh_fugue`, you may notice that the product sounds artificial. This is because it is created from pure sinusoids. In this section, you will incorporate an envelope into your music synthesis process in order to improve the quality of the final product.

When played by an instrument, notes are not single pure evenly-amplified tones. Their amplitudes change over the course of a single note, with *attack*, *decay*, *sustain*, and *release* parts of a note, together making up an 'ADSR' profile. Enveloping creates short pauses between musical notes, as a scaling vector where all values are between 0 and 1. Envelopes can be implemented in the synthesis process by multiplying each note by an enveloping function E(t).



E(t) should be a mathematical representation of the above. You can get such a curve by using the `linspace()` or `hanning()`. The `hanning()` function is useful here, as it gets the job done since it's a bell-shaped curve on a scale from 0 - 1. These correlate to relative magnitudes and durations of each stage. The envelope `E` is multiplied (dot multiplied) by each note in the signal, scaling their amplitudes accordingly. As `E(1)` and `E(length(note))` should both be zero, the values of the vector should fade in-and-out, reducing the roughness produced in synthetic note-to-note transitions. Your job here is to improve the sound quality.

Hint 1:The function `interp1()` may be useful in this section.

Hint 2:You'll want to make sure that your note and `E` are the same length.

Hint 3:Previous generations of students have found the Hanning Window to be an effective envelope.

Implement an envelope to improve the sound of your Bach Fugue. Save it as *bach_envelope.wav* and submit it on Canvas.

Note: grading here is based on your ability to improve the sound quality. This part is intentionally left vague; we want you to be creative here!

2.3 Musical Tweaks – Fourier Series of aTrumpet

Remember that any real-world periodic signal can be written as a sum of sinusoids whose frequencies (in Hertz) are integer multiples of the fundamental frequency. The amplitudes and phases of the sinusoids are all that is needed to specify the Fourier series expansion. Musical signals are a natural application of Fourier series, since musical signals (specifically, an instrument playing a specific note) are periodic. The sinusoid with the same period as the music signal is called the fundamental; it is the pure tone that sounds most like the signal. To obtain the richer sound of an instrument playing a note, we add in harmonics; musicians call them overtones. Harmonics are sinusoids at frequencies double, triple, etc. the frequency of the fundamental. The amplitudes of the harmonics determine the timbre (sound) of the instrument playing the note. The reason a violin playing note B sounds different from a trumpet playing note B is that the amplitudes of the harmonics are different.

The Fourier series representation of a musical signal is specified by only a few numbers (the amplitudes and phases of the harmonics, and the note):

$$x(t) = \sum_{k}^{\infty} Re\{A_k e^{j\omega_0 kt} e^{-j\varphi_k}\} = \sum_{k}^{\infty} A_k \cos(\omega_0 kt - \varphi_k)$$

In practice, the infinite series is truncated to a finite number (k) of terms, giving a finite Fourier series. Finite Fourier series still give good approximations to most periodic signals.

Since instruments playing musical notes create periodic signals, musical signals have Fourier series expansions. The Fourier series can be truncated to a finite number of terms and still do a good job of representing the musical signal. The timbre (sound) of the trumpet is produced by the amplitudes A_k. The fundamental frequency alone would be a pure tone; the overtones (harmonics) create the richer sound of the trumpet.

Here is a table of A_k and φ_k values for nine harmonics:

Harmonic k	A_k- Relative Amplitude	φ_k - Relative Phase in Radians
1 (fundamental)	0.1155	-2.1299
2	0.3417	+1.6727
3	0.1789	-2.5454
4	0.1232	+0.6607
5	0.0678	-2.0390
6	0.0473	+2.1597
7	0.0260	-1.0467
8	0.0045	+1.8581
9	0.0020	-2.3925

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?

Using a sampling frequency of 44100 Hz, construct your Bach Fugue in trumpet. Submit this as a *.wav* file on Canvas.

Don't forget to submit your PDF file, too!

2.9001 Extra Credit – Synthesize a Musical Piece of your own!

For up to fifteen points of extra credit, out of 100 for the entire lab, synthesize a piece of music into a structure of `theVoices`form, calling it `theVoices_yourname`. Save it as `yourname_fugue.mat`. Submit the *.mat* file, your *.m* code, your PDF explaining it, and the *.wav* of your musical composition. A simple composition might get 5 points. A more complex one might get 10. A really well-done one might earn the full 15 points. Again, be creative!

Some ideas:

How might you create a beat using MATLAB? The Internet has the answer for you.Some melodies are best played using a trumpet. Others might do better with a piano-sounding set of harmonics and envelopes, or a flute-sounding set of harmonics and envelopes.

We look forward to hear your compositions!