Taylor Rembos
EEL 3135
Lab 3 Part 1

Lab 3: Music Synthesis with Sinusoidal Signals

Lab Part One

1.1 A Function to Play a Note

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 key2note.m

```
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 = 440*(2^((keynum-49)/12));%<========= fill in this line
    xx = real(X*exp(j*2*pi*freq*tt)); %<======= fill in this line
end
```

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

Use glottalkey2note()to write a script, play_glottallamb.m, that plays a series of notes.

checked off

```
% --------------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 = key2note(1,keynum,0.38); %amplitude 1, keynum, 0.38s
   % <------- Fill in this line
        n2 = n1 + length(tone) - 1;
        xx(n1:n2) = tone; %<------- Insert the note
        n1 = n2 + 1;
     end
soundsc(xx,fs)
```

```
>> specgram(xx,512,fs)
>> title('Marys Lamb');
>> specgram(xx,812,fs)
>> title('Marys Lamb');
>> specgram(xx,11025,fs)
>> title('Marys Lamb');
>> specgram(xx,12,fs)
>> title('Marys Lamb');
```

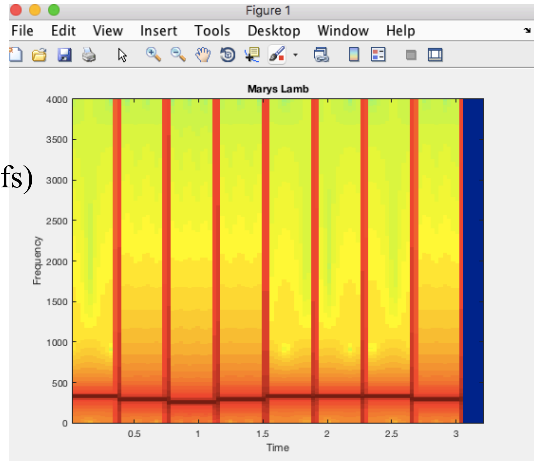


Figure 2 - 512

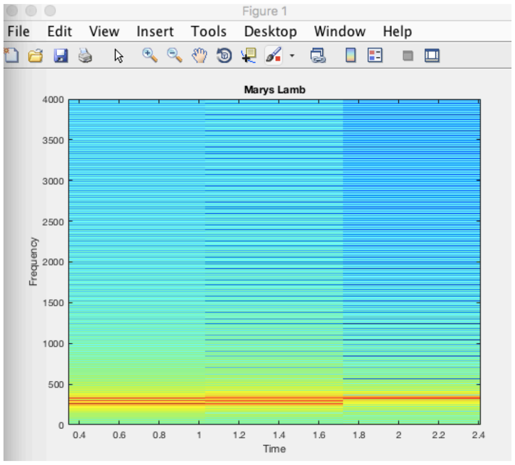


Figure 3 - 11025

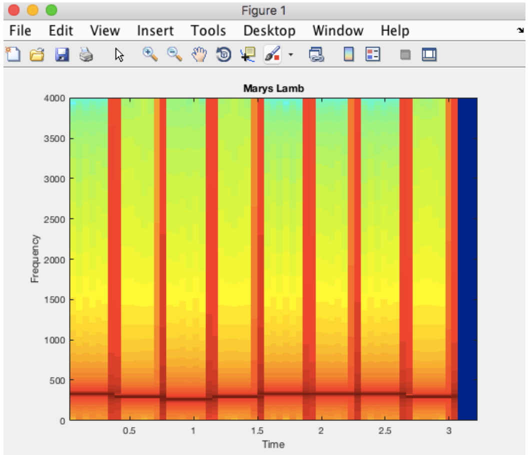


Figure 2 - 812

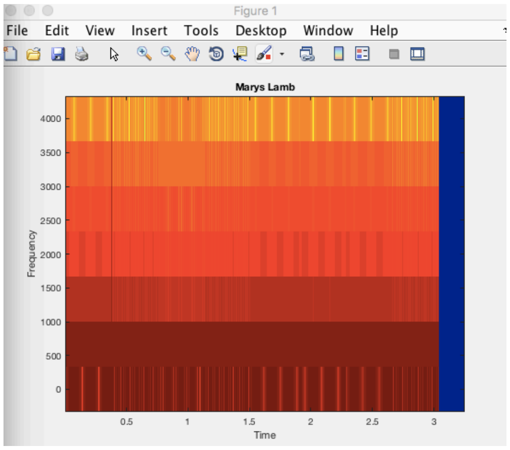
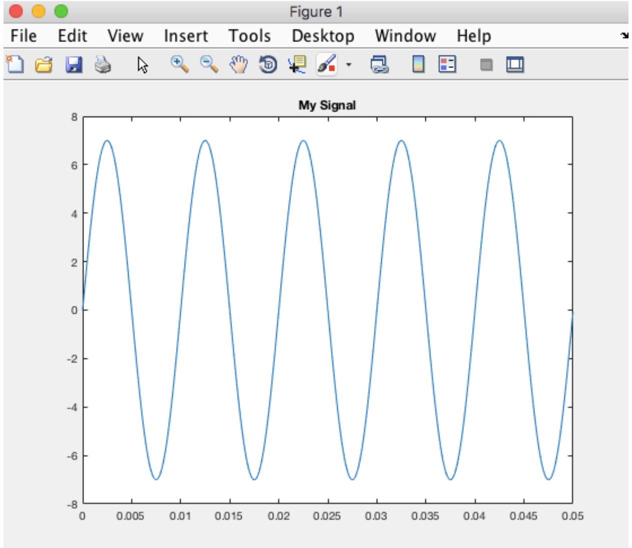


Figure 4 - 12

1.3 Structures

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 =
  struct with fields:

      Amp: 7
    phase: -1.5708
     freq: 100
       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

x =
  struct with fields:

             Amp: 7
           phase: -1.5708
            freq: 100
              fs: 11025
    timeInterval: [1×552 double]
          values: [1×552 double]
            name: 'My Signal'

>> plot(x.timeInterval, x.values)
>> title(x.name)
```



1.4 The Evenly-Timed First Voice

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.

checked off

Script play_firstvoice_even.m

```
% --------------play_firstvoice_even.m-------------- %
firstvoice_even.keys = [theVoices(1).noteNumbers];
% NOTES: C D E F G
% Key #40 is middle-C

firstvoice_even.durations = 0.25 *
ones(1,length(firstvoice_even.keys));
fs = 8000; % 11025 Hz also works
xx = zeros(1, sum(firstvoice_even.durations)*fs +
length(firstvoice_even.keys));
n1 = 1;
     for kk = 1:length(firstvoice_even.keys)
          keynum = firstvoice_even.keys(kk);
        tone = key2note(1,keynum,0.5); %amplitude 1, keynum, 0.38s % <-
------ Fill in this line
        n2 = n1 + length(tone) - 1;
        xx(n1:n2) = tone; %<------- Insert the note
        n1 = n2 + 1;
     end
soundsc(xx,fs)
```

```
>> load('barukh_fugue.mat')
>> play_firstvoice_even
```

1.5 The Correctly-Timed First Voice

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.

‘play_firstvoice.wav’ file

Script play_firstvoice.m

```
% --------------play_firstvoice_even.m-------------- %
load('barukh_fugue.mat');
firstvoice_even.keys = [theVoices(1).noteNumbers];
% NOTES: C D E F G
% Key #40 is middle-C

firstvoice_even.durations = 0.25 *
ones(1,length(theVoices(1).noteNumbers));
fs = 8000; % 11025 Hz also works
xx = zeros(1,
0.25*fs*(theVoices(1).durations(length(theVoices(1).durations)) +
theVoices(1).startPulses(length(theVoices(1).durations))));
%n1 = 1;
     for kk = 1:length(theVoices(1).noteNumbers)
          keynum = firstvoice_even.keys(kk);
        tone = key2note(1,keynum,theVoices(1).durations(kk)./4); % <---
---- Fill in this line
        n1 = theVoices(1).startPulses(kk)*fs/4 +1;
        n2 = n1 + length(tone) - 1;
        xx(n1:n2) = tone; %<------- Insert the note
     end
     xx = xx/(max (abs (xx))); %% anti clipping xx
soundsc(xx,8000)
audiowrite('song.wav',xx,fs);
```

```
>> load('barukh_fugue.mat')
>> play_firstvoice
```

1.6 Silence and startPulses: Construction of the Better Fugue

Produce the Better Fugue, save it as a *.wav* file, and submit it on Canvas alongside your lab document.

‘song.wav’ file

function playSong.m

```
function song = playSong(theVoices)
% PLAYSONG Produce a sinusoidal waveform containing the combination
ofthe different notes in theVoices
% usage: song = playSong ()
% song = the output sinusoidal waveform
load barukh_fugue.mat
fs = 8000;
spp = 0.25 %%% 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 =
zeros(1,spp*fs*(theVoices(3).durations(length(theVoices(3).durations))
+ (theVoices(3).startPulses(length(theVoices(3).durations))))); %%%
vector of zeros

% Then add in the notes
    for i = 1:length(theVoices) % Cycle through each set of notes
        % Convert data arrays to appropriate units
        for j = 1:length(theVoices(i).noteNumbers) % Cycle throughеach
note in aset
            keynum = theVoices(i).noteNumbers(j);
            note = key2note(1,keynum,theVoices(i).durations(j).*spp);
%%% create sinusoid of correct length to represent a single note
            locstart = theVoices(i).startPulses(j)*(fs/4)+1; %%% 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 (abs (song))); %% anti clipping xx
    soundsc(song,fs);
    audiowrite('song.wav',song,fs);
% Use audiowrite() to generate WAV file
end
```