Table of Contents

PREAMBLE

DO NOT REMOVE THE LINE BELOW

```
clear;
```

Connor Dupuis, Section: 28944, TA: Naoki Sawahashi

QUESTION 1: COMMENTING

```
========================

type('eel3135_lab02_comment.m')


%% ACKNOWLEDGEMENTS / REFERENCES:
% This code uses functions written by Ken Schutte in 2019, which is
 used to
% read and decode midi files. The code is under a GNU General
% Public License, enabling us to run, study, share, and modify the
% software.
%
% More info can be found at: http://www.kenschutte.com/midi


%% INITIAL SETUP
clear
close all
clc

%% DEFINE MUSIC
```

```
% INTITIAL VARIABLES
Fs = 44100;             % ==> The sampling freqeuncy <==

%  ===> Executes the fucntion midiInfo and puts the results into
 variables Notes and endtime <===
[Notes, endtime] = midiInfo(readmidi('gym.mid'), 0, 2);
L = size(Notes,1);      % ==> Assigns L to the length of the first
 dimension of Notes <==

%  ===> Passes the function build_song a column vector of ones with
 length L, column 3 of Notes, column 6 - 5 of notes, and Fs <===
x = build_song(ones(L,1), Notes(:,3), Notes(:,6)-Notes(:,5), Fs);

%  ===> Assigns tot_samples the ceiling of the ((sum of all the
 elements from the result of columns 6 - 5 of Notes) multiplied by
 Fs). <===
tot_samples = ceil(sum(Notes(:,6)-Notes(:,5))*Fs);

%  ===> Creates a figure with the plots x vs t one with defualt axis
 values and one with configured axis value<===
t = 0:1/Fs:(tot_samples-1)/Fs;  % ==> Sets t (the time) equal to the
 vector of 0 to Fs with increments of tot_samples-1 <==
figure(1);
subplot(211)
plot(t, x);
xlabel('Time [s]')
ylabel('Amplitude')
subplot(212)
plot(t, x);
xlabel('Time [s]')
ylabel('Amplitude')
axis([0 0.1 -1 1])              % ==> Specifies the limits for the
 current axes x axis is from 0 to 0.1 and the y axis is form -1 to
 1<==

%  ===> Takes any keyboard input to advance to the nextline to exeute
 soundsc <===
input('Click any button to play sound')
soundsc(x, Fs);



% ========
% YOU DO *NOT* NEED TO DESCRIBE THESE LINES (your free to figure it
 out though)
W = 0.1;    % Window size
tic;
for mm = 1:ceil(tot_samples/Fs/W)
    % PAUSE UNTIL NEXT FRAME
    xlim([(mm-1)*W+[0 W]]); % Set limits of plot
    tm = toc;                       % Check current time
    if mm*W < tm, disp(['Warning: Visualization is ' num2str(mm*W-tm)
 's behind']); end
```

```
    drawnow; pause(mm*0.1-tm);       % Synchronize with clock
end
% =======



%%
% ==========================================
% SUPPORTING FUNCTIONS FOUND BELOW
% Add comments appropriately below
% ==========================================


function x = key_to_note(A, key, dur, fs)
% key_to_note: ========> Takes in a complex amplitude, key, duration,
 and sampling rate and outputs the sinusoidal waveform of the note
 <=========
%
% Input Args:
%     A: complex amplitude
%   key: number of the note on piano keyboard
%   dur: duration of each note (in seconds)
%    fs: A scalar sampling rate value
%
% Output:
%     x: sinusoidal waveform of the note

    %  ===> Sets N equal to the floor of the duraion multiplied by the
 sampling rate.
    % Sets t eqaul to the vector 0 to N-1, then divided by fs. Sets
 freq value usign the key  <===
    N    = floor(dur*fs);
    t    = (0:(N-1)).'/fs;
    freq = (440/32)*2^((key-9)/12);

    %  ===> Takes the real value of he complex equation <===
    x    = real(A*exp(1j*2*pi*freq*t));


end


function x = build_song(As, keys, durs, fs)
% build_song:  ========> Takes in arguments As, keys, durs, fs to
 produce a raw audio signalof length N*fs  <=========
%
% Input Args:
%   As: A length-N array of complex amplitudes for building notes
% keys: A length-N array of key numbers (which key on a keyboard) for
 building notes
%   durs: A length-N array of durations (in seconds) for building
 notes
%     fs: A scalar sampling rate value
%
```

```
% Output Args:
%      x: A length-(N*fs) length raw audio signal
%
    %  ===> Sets x equal to a column vector of zeros using the
 duration and sampling rate <===
    x = zeros(ceil(sum(durs)*fs), 1);
    for k = 1:length(keys)

        %  ===> Sets note equal to the converted key using the helper
 function key_to_note <===
        note       = key_to_note(As(k), keys(k), durs(k), fs);
        start_time = sum(durs(1:k-1));

        %  ===> Sets n1, n2, and x(n1:n2) to corresponding values
 using the start time and sampling rate  <===
        n1         = floor(start_time*fs) + 1;
        n2         = floor(start_time*fs) + floor(durs(k)*fs);
        x(n1:n2)   = x(n1:n2) + note;

    end

end
```

QUESTION 2

```
========================
```

2(a) PLOT FIRST FOUR PERIODS

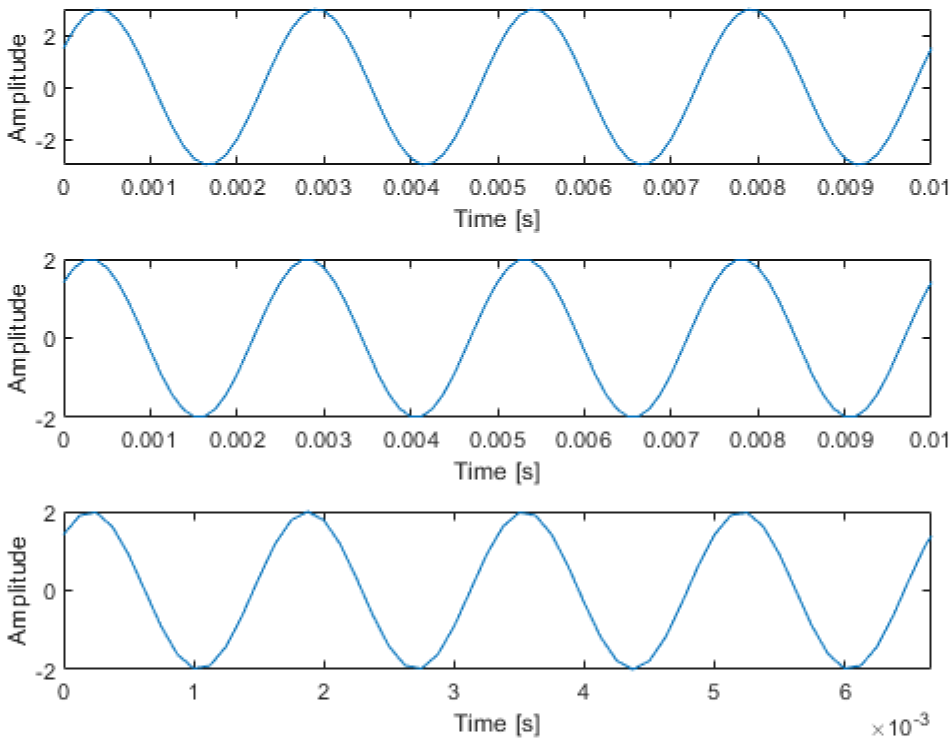
```
fs = 8000; % Sampling frequency
Ts = 1/fs; % Sampling period
t = 0:Ts:0.01; % Time

figure(1);

f1 = 400; % Wave frequency
T1 = 1/f1; % Wave Period
s1 = 3*cos(2*pi*f1*t - pi/3);
subplot(311);
plot(t,s1)
xlabel('Time [s]')
ylabel('Amplitude')
axis([0 T1*4 -3 3]) % Sets the axis values for 4 periods

f2 = 400; % Wave frequency
T2 = 1/f2; % Wave Period
s2 = 2*cos(2*pi*f2*t - pi/4);
subplot(312);
plot(t,s2)
xlabel('Time [s]')
ylabel('Amplitude')
axis([0 T2*4 -2 2]) % Sets the axis values for 4 periodsods
```

```
f3 = 600; % Wave frequency
T3 = 1/f3; % Wave Period
s3 = 2*cos(2*pi*f3*t - pi/4);
subplot(313);
plot(t,s3)
xlabel('Time [s]')
ylabel('Amplitude')
axis([0 T3*4 -2 2]) % Sets the axis values for 4 periods
```



2(b) CREATE AND SUBMIT .WAV FILE

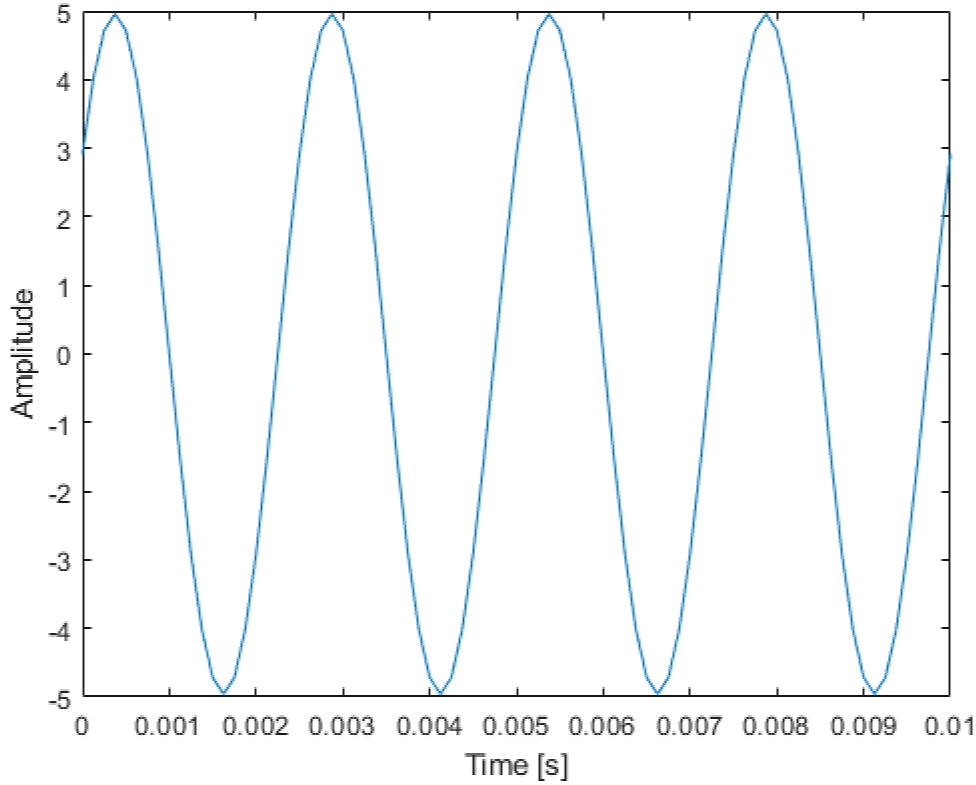
```
s1_scaled = s1/max(abs(s1));
s2_scaled = s2/max(abs(s2));
s3_scaled = s3/max(abs(s3));

audiowrite('s1.wav',s1_scaled,fs);
audiowrite('s2.wav',s2_scaled,fs);
audiowrite('s3.wav',s3_scaled,fs);
close all
```

2(c) PLOT FIRST FOUR PERIODS

```
x1 = s1 + s2; % Sums s1 and s2
T4 = 1/400; % Wave period
```

```
plot(t,x1)
xlabel('Time [s]')
ylabel('Amplitude')
axis([0 T4*4 -5 5]) % Sets the axis values for 4 periods
```

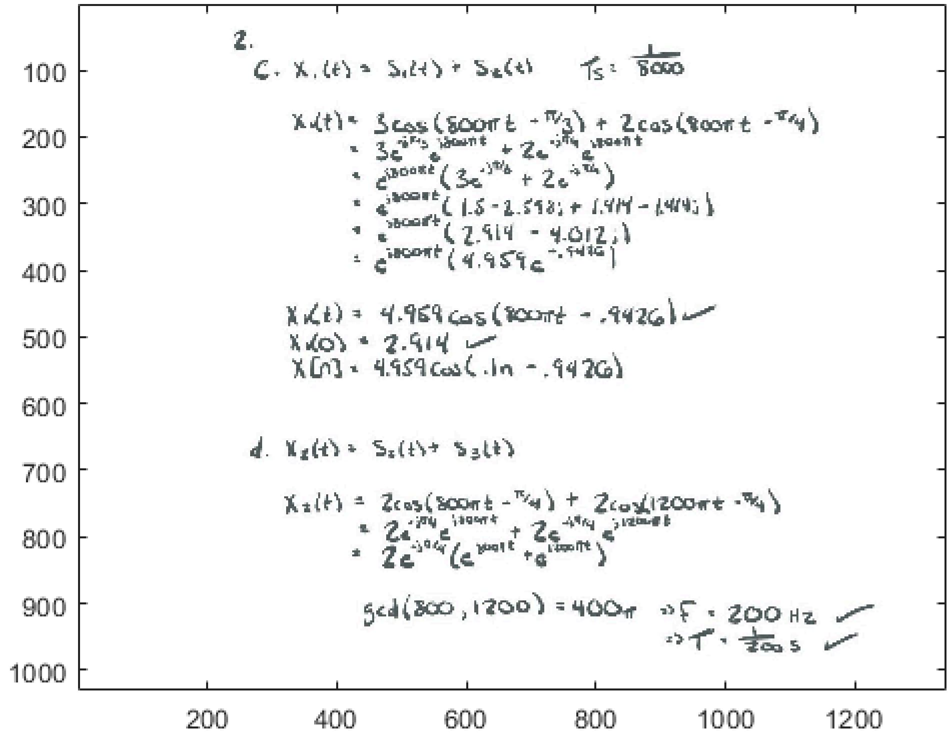


2(d) PLOT FIRST FOUR PERIODS

```
T5 = 1/200; % Wave Period
t1 = 0:Ts:T5*4; % Time
s2 = 2*cos(2*pi*f2*t1 - pi/4); % Need s2  to be redifined in terms of
 t2
s3 = 2*cos(2*pi*f3*t1 - pi/4); % Need s3  to be redifined in terms of
 t2
x2 = s2 + s3; % Sums s2 and s3

plot(t1,x2)
xlabel('Time [s]')
ylabel('Amplitude')

img = imread('hand.jpg');
image(img)
```



QUESTION 3

==========================

3(a) CREATE SOUND

```
fs = 8000; % Sampling freqeuncy

% Defining values
As = [1 1 1 1 1 1 1 1 1 1 1 1 1];
keys = [44 42 40 42 44 44 44 42 42 42 44 47 47];
durs = [1 1 1 1 1 1 2 1 1 2 1 1 2]*1/4;

mary_defualt = build_song(As, keys, durs, fs); % Sets x equal to the
 created song
soundsc(mary_defualt, fs); % Plays the song


mary_defualt_scaled = mary_defualt/max(abs(mary_defualt)); % Scales
 the value prior to writing
audiowrite('mary_defualt.wav', mary_defualt_scaled, fs); %Writes to an
 audio file
```

3(b) MODIFY FUNCTION (key_to_note_trumpet function is at end of file)

Only need to modify function -- this area can be empty

3(c) CREATE SOUND (build_song_trumpet function is at end of file)

```
mary_trumpet = build_song_trumpet(As, keys, durs, fs); % Sets x equal
 to the created song
soundsc(mary_trumpet, 8000); % Plays the song

mary_trumpet_scaled = mary_trumpet/max(abs(mary_trumpet)); % Scales
 the value prior to writing
audiowrite('mary_trumpet.wav', mary_trumpet_scaled, fs); %Writes to an
 audio file
```

3(d) CREATE SOUND (build_song_time function is at end of file)

```
start_time = [0 1 2 3 4 5 6 7 8 9 10 11 12]*1/4;
end_time = ([0 1 2 3 4 5 7 8 9 11 12 13 15]+0.2)*1/4;

mary_time = build_song_time(As, keys, start_time, end_time, fs);
soundsc(mary_time, fs);

mary_time_scaled = mary_time/max(abs(mary_time)); % Scales the value
 prior to writing
audiowrite('mary_time.wav', mary_time_scaled, fs); %Writes to an audio
 file
```

3(e) PLOT COMPARISONS

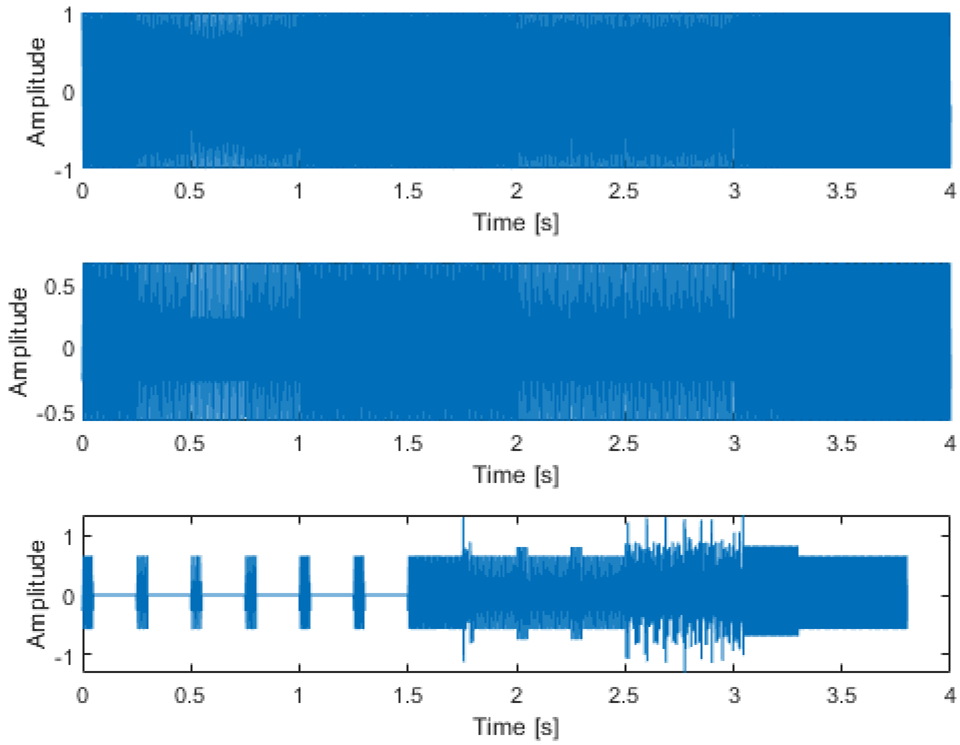
```
t = 0:1/fs:(length(mary_defualt)-1)/fs;

figure(1);
subplot(311);
plot(t, mary_defualt);
xlabel('Time [s]')
ylabel('Amplitude')

subplot(312);
plot(t, mary_trumpet);
xlabel('Time [s]')
ylabel('Amplitude')

t = 0:1/fs:(length(mary_time)-1)/fs;
```

```
subplot(313);
plot(t, mary_time);
xlabel('Time [s]')
ylabel('Amplitude')
```



3(f) ANSWER QUESTION

```
% Due to the silent gaps that build_song_time produces, there is empty
% space in the graph where no sound is being produced. There is also
% overlap near the end of the graph due to certain notes starting
 before
% another has stopped.
```

== SUPPORTING FUNCTIONS FOUND BELOW ==

```
function x = key_to_note(A, key, dur, fs)
% key_to_note: Produces a sinusoidal waveform corresponding to a
%  given piano key number
%
% Input Args:
%     A: complex amplitude
%   key: number of the note on piano keyboard
%   dur: duration of each note (in seconds)
%    fs: A scalar sampling rate value
%
```

```
% Output:
%     x: sinusoidal waveform of the note

    N    = floor(dur*fs);
    t    = (0:(N-1)).'/fs;
    freq = (440/32)*2^((key-9)/12);
    x    = real(A*exp(1j*2*pi*freq*t));


end


function x = key_to_note_round(A, key, dur, fs)
% key_to_note: Produces a sinusoidal waveform corresponding to a
%  given piano key number this time with rounding specifically for
%   build_song_time
%
% Input Args:
%     A: complex amplitude
%   key: number of the note on piano keyboard
%   dur: duration of each note (in seconds)
%    fs: A scalar sampling rate value
%
% Output:
%     x: sinusoidal waveform of the note

    N    = round(dur*fs);
    t    = (0:(N-1)).'/fs;
    freq = (440/32)*2^((key-9)/12);

    Ak = [0.1155, 0.3417, 0.1789, 0.1232, 0.0678, 0.0473, 0.0260,
 0.0045, 0.0020]; % Harmonic amplitudes
    phi = [-2.1299, 1.6727, -2.5454, 0.6607, -2.0390, 2.1597, -1.0467,
 1.8581, -2.3925]; % Harmonic phase shifts

    % For loop iterating through and summing harmonics
    x = 0;
    for k = 1:length(Ak)
        x = x + Ak(k)*cos(2*pi*k*freq*t + phi(k));
    end
end

function x = build_song(As, keys, durs, fs)
% build_song: Uses key_to_note and the inputted duration to create an
 output
%   of notes for a specified amount of time.
%
% Input Args:
%   As: A length-N array of complex amplitudes for building notes
% keys: A length-N array of key numbers (which key on a keyboard) for
 building notes
%   durs: A length-N array of durations (in seconds) for building
 notes
%     fs: A scalar sampling rate value
```

```
%
% Output Args:
%      x: A length-(N*fs) length raw audio signal
%

    x = zeros(floor(sum(durs)*fs), 1);
    for k = 1:length(keys)
        note       = key_to_note(As(k), keys(k), durs(k), fs);
        start_time = sum(durs(1:k-1));
        n1         = floor(start_time*fs) + 1;
        n2         = floor(start_time*fs) + floor(durs(k)*fs);
        x(n1:n2)   = x(n1:n2) + note;
    end
end


function x = key_to_note_trumpet(A, key, dur, fs)
% key_to_note: Produces a sinusoidal waveform corresponding to a
%  given piano key number
%
% Input Args:
%     A: complex amplitude
%   key: number of the note on piano keyboard
%   dur: duration of each note (in seconds)
%    fs: A scalar sampling rate value
%
% Output:
%     x: sinusoidal waveform of the note

    N    = floor(dur*fs);
    t    = (0:(N-1)).'/fs;
    freq = (440/32)*2^((key-9)/12);

    Ak = [0.1155, 0.3417, 0.1789, 0.1232, 0.0678, 0.0473, 0.0260,
 0.0045, 0.0020]; % Harmonic amplitudes
    phi = [-2.1299, 1.6727, -2.5454, 0.6607, -2.0390, 2.1597, -1.0467,
 1.8581, -2.3925]; % Harmonic phase shifts

    % For loop iterating through and summing harmonics
    x = 0;
    for k = 1:length(Ak)
        x = x + Ak(k)*cos(2*pi*k*freq*t + phi(k));
    end

% Different solutions for the same problem

%     % Creating freqeuncy vector for harmoncs
%     freqk = [1*freq, 2*freq, 3*freq, 4*freq, 5*freq, 6*freq, 7*freq,
 8*freq, 9*freq];

%     for k = 1:length(Ak)
%         x = x + Ak(k)*cos(2*pi*freqk(k)*t + phi(k));
%     end
```

```
%     % Manually summing all harmonics
%     x   = A*((0.1155*cos(2*pi*1*freq*t-2.1299))
 + (0.3417*cos(2*pi*2*freq*t+1.6727)) +
 (0.1789*cos(2*pi*3*freq*t-2.5454))...
%             + (0.1232*cos(2*pi*4*freq*t+0.6607)) +
 (0.0678*cos(2*pi*5*freq*t-2.0390)) + (0.0473*cos(2*pi*6*freq*t
+2.1597))...
%             + (0.0260*cos(2*pi*7*freq*t-1.0467))
 + (0.0045*cos(2*pi*8*freq*t+1.8581)) +
 (0.0020*cos(2*pi*9*freq*t-2.3925)));
end


function x = build_song_trumpet(As, keys, durs, fs)
% build_song_trumpet: Uses key_to_note and the inputted duration to
 create an output
%   of notes for a specified amount of time. Uses the harmonics of a
%   trumpet to change the timbre.
%
% Input Args:
%   As: A length-N array of complex amplitudes for building notes
% keys: A length-N array of key numbers (which key on a keyboard) for
 building notes
%   durs: A length-N array of durations (in seconds) for building
 notes
%     fs: A scalar sampling rate value
%
% Output Args:
%      x: A length-(N*fs) length raw audio signal
%

    x = zeros(floor(sum(durs)*fs), 1);
    for k = 1:length(keys)
        note       = key_to_note_trumpet(As(k), keys(k), durs(k), fs);
        start_time = sum(durs(1:k-1));
        n1         = floor(start_time*fs) + 1;
        n2         = floor(start_time*fs) + floor(durs(k)*fs);
        x(n1:n2)   = x(n1:n2) + note;
    end
end



function x = build_song_time(As, keys, start_time, end_time, fs)
% build_song: Uses key_to_note and the inputted start and end time to
 create an output
%   of notes for a specified amount of time.
%
% Input Args:
%          As: A length-N array of complex amplitudes for building
 notes
%        keys: A length-N array of key numbers (which key on a
 keyboard) for building notes
%  start_time: A length-N array of start times (in seconds) for notes
```

```
%    end_time: A length-N array of end times (in seconds) for notes
%          fs: A scalar sampling rate value
%
% Output Args:
%      x: A length-(N*fs) length raw audio signal
%
    durs = end_time - start_time;
    x = zeros(end_time(length(end_time))*fs, 1);
    for k = 1:length(keys)
        note       = key_to_note_round(As(k), keys(k), durs(k), fs);
        n1         = floor(start_time(k)*fs) + 1;
        n2         = ceil(end_time(k)*fs);
        x(n1:n2)   = x(n1:n2) + note;
    end
end
```

Published with MATLAB® R2020a