GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 2009

Lab #4: Synthesis of Sinusoidal Signals—Music Synthesis

Date: 15–21 Sept-09

You should read the Pre-Lab section of the lab and do all the exercises in the Pre-Lab section before your assigned lab time.

ITS: When you come to the lab, you **must** answer the online ITS questions. You can use MATLAB or any notes you might have but you cannot discuss the exercises with any other students.

Verification: The Warm-up section of each lab must be completed **during your assigned Lab time** and the steps marked *Instructor Verification* must also be signed off **during the lab time**. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the TA or instructor. After completing the warm-up section, turn in the verification sheet to your TA *before leaving the lab.*

It is only necessary to turn in Section 5 as the lab report for this lab. More information on the lab report format can be found on `t-square` under the **INFO** link. Please ***label*** the axes of your plots and include a title and Figure number for every plot. In order to reduce *orphan plots*, include each plot as a figure *embedded* within your report. This can be done easily with MATLAB's `notebook` capability. For more information on how to include figures and plots from MATLAB in your report file, consult the **INFO** link on `t-square`, or ask your TA for details.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students and you are allowed to consult old lab reports, but you cannot give or receive written material or electronic files. Your submitted work should be original and it should be your own work.

The report will be **due during the period 22–28-Sept. at the start of your lab.**

1 Introduction

This lab includes a project on sound synthesis with sinusoidal waveforms of the form

$$x(t) = \sum_k A_k \cos(\omega_k t + \phi_k) \tag{1}$$

so it will be necessary to establish the connection between musical notes, their frequencies, and sinusoids. A secondary objective of the lab is to document the relationship between the synthesized signal, its spectrogram and the musical notes.

2 Pre-Lab

In this lab, the periodic waveforms and music signals will be created with the intention of playing them out through a speaker. Therefore, it is necessary to take into account the fact that a conversion is needed from the digital samples, which are numbers stored in the computer memory to the actual continuous waveform that will be amplified and heard through the speakers.

2.1 Theory of Sampling

Chapter 4 treats sampling in detail, but this lab is usually done prior to lectures on sampling, so we provide a quick summary of essential facts here. The idealized process of sampling a signal and the subsequent reconstruction of the signal from its samples is depicted in Fig. 1. This figure shows a continuous-time input signal $x(t)$, which is sampled by the continuous-to-discrete (C-to-D) converter to produce a sequence of samples $x[n] = x(nT_s)$, where n is the integer sample index and T_s is the sampling period. The sampling

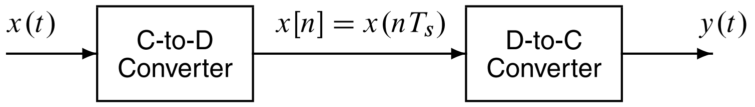

Figure 1: Sampling and reconstruction of a continuous-time signal.

rate is $f_s = 1/T_s$ where the units are samples per second. As described in Chapter 4 of the text, the ideal discrete-to-continuous (D-to-C) converter takes the signal samples $x(nT_s)$ and interpolates a smooth curve through them. The *Sampling Theorem* tells us that when the input signal $x(t)$ is a sum of sine waves, the output $y(t)$ will be equal to the input $x(t)$ if the sampling rate is *more than twice the highest frequency* $f_{\max}$ in the input, i.e., we need $f_s > 2f_{\max}$. In other words, if we *sample fast enough* then there will be no problems synthesizing the continuous audio signals from $x[n]$.

2.2 D-to-A Conversion

Most computers have a built-in hardware for the analog-to-digital (A-to-D) converter and the digital-to-analog (D-to-A) converter (usually on the sound card). These hardware systems are physical realizations of the idealized concepts of C-to-D and D-to-C converters respectively, but for purposes of this lab we will assume that the hardware A/D and D/A are perfect realizations.

The digital-to-analog conversion process has a number of aspects, but in its simplest form the only thing we need to worry about (in this lab) is that the time spacing (T_s) between the signal samples must correspond to the rate of the D-to-A hardware that is being used. From MATLAB, the sound output is done by the `soundsc(xx,fs)` function which does support a variable D-to-A sampling rate `(fs)` if the hardware on the machine has such capability. A convenient choice for the D-to-A conversion rate is 11025 samples per second,[1] so $T_s = 1/11025$ seconds; another common choice is 8000 samples/sec. Both of these rates satisfy the requirement of *sampling fast enough* as explained in the next section. In fact, most piano notes have relatively low frequencies, so an even lower sampling rate could be used. If you are using `soundsc()`, the vector `xx` will be scaled automatically so that its maximum value equals the maximum for the D-to-A converter, but if you are using `sound.m`, you must scale the vector `xx` so that it lies between ± 1. Consult `help sound`.

3 Sampling Sinusoids

The ideal C-to-D converter is, in effect, being implemented whenever we take samples of a continuous-time formula, e.g., $x(t)$ at $t = t_n$. We do this in MATLAB by first making a vector of times, and then evaluating the formula for the continuous-time signal at the sample times, i.e., $x[n] = x(nT_s)$ if $t_n = nT_s$. This assumes perfect knowledge of the input signal, but we have already been doing it this way in previous labs.

(a) To begin, create a vector `x1` of samples of a sinusoidal signal with $A_1 = 100$, $\omega_1 = 2\pi(800)$, and $\phi_1 = -\pi/3$. Use a sampling rate of 11025 samples/sec, and compute a total number of samples equivalent to a time duration of 0.5 secs. You may find it helpful to recall that a MATLAB statement such as `tt=(0:0.01:3);` would create a vector of numbers from 0 through 3 with increments of 0.01. Therefore, it is only necessary to determine the time increment needed to obtain 11025 samples in one second. You could use the `myaddcos()` function from a previous lab for this part *with a modification of the sampling rate.*

[1]This sampling rate is one quarter of the 44,100-Hz rate used in audio CD players.

Use `soundsc()` to play the resulting vector through the D-to-A converter of the your computer, assuming that the hardware can support the $f_s = 11025$ Hz rate. Listen to the output.

(b) Now create another vector `x2` of samples of a second sinusoidal signal (0.8 secs. in duration) for the case $A_2 = 80$, $\omega_2 = 2\pi(1200)$, and $\phi_2 = +\pi/4$. Listen to the signal reconstructed from these samples. How does its sound compare to the signal in part (a)?

(c) ***Concatenate*** the two signals `x1` and `x2` from the previous parts and put a short duration of 0.1 seconds of silence in between. You should be able to concatenate by using a statement something like:

```
xx = [ x1, zeros(1,N), x2 ];
```

assuming that both `x1` and `x2` are row vectors. Determine the correct value of **`N`** to make exactly 0.1 seconds of silence. Listen to this new signal to verify that it is correct.

(d) To verify that the concatenation operation was done correctly in the previous part, make the following plot:

```
tt = (1/11025)*(1:length(xx));   plot( tt, xx );
```

This will plot a huge number of points, but it will show the "envelope" of the signal and verify that the amplitude changes from 100 to zero and then to 80 at the correct times. Notice that the time vector `tt` was created to have exactly the same length as the signal vector `xx`.

(e) Now send the vector `xx` to the D-to-A converter again, but change the sampling rate parameter in `soundsc(xx, fs)` to 22050 samples/second. *Do not recompute the samples in* `xx`, just tell the D-to-A converter that the sampling rate is 22050 samples/sec. Describe how the *duration* and *pitch* of the signal were affected. Explain.

3.1 Structures in MATLAB

MATLAB can do structures. Structures are convenient for grouping information together. For example, run the following program which plots a sinusoid:

```
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 )
```

Notice that the members of the structure can contain different types of variables: scalars, vectors or strings.

3.2 Debugging Skills

Testing and debugging code is a big part of any programming job. Almost any modern programming environment provides a *symbolic debugger* so that break-points can be set and variables examined while a program is running. Nonetheless, many programmers still insist on using the old-fashioned method of inserting print statements in the middle of the code (or the MATLAB equivalent, leaving off a few semi-colons). This is akin to riding a tricycle to commute around Atlanta.

There are two ways to use debugging tools in MATLAB: via buttons in the edit window or via the command line. For help on the edit-window debugging features, access the menu `Help->Using the M-File`

`Editor` which will pop up a browser window at the help page for editing and debugging. For a summary of the command-line debugging tools, try `help debug`. Here is part of what you'll see:

```
dbstop     - Set breakpoint.
dbclear    - Remove breakpoint.
dbcont     - Resume execution.
dbstack    - List who called whom.
dbstatus   - List all breakpoints.
dbstep     - Execute one or more lines.
dbtype     - List M-file with line numbers.
dbquit     - Quit debug mode.

When a breakpoint is hit, MATLAB goes into debug mode.  On the PC
and Macintosh the debugger window becomes active and on UNIX and VMS
the prompt changes to a K>.  Any MATLAB command is allowed at the
prompt.  To resume M-file function execution, use DBCONT or DBSTEP.
To exit from the debugger use DBQUIT.
```

One of the most useful modes of the debugger causes the program to jump into "debug mode" whenever an error occurs. This mode can be invoked by typing:

```
dbstop if error
```

With this mode active, you can snoop around inside a function and examine local variables that probably caused the error. You can also choose this option from the debugging menu in the MATLAB editor. It's sort of like an automatic call to 911 when you've gotten into an accident. Try `help dbstop` for more information.

Download the file `coscos.m` and use the debugger to find the error(s) in the function. Call the function with the test case: `[xn,tn] = coscos(2,3,20,1)`. Use the debugger to:

1. Set a breakpoint to stop execution when an error occurs and jump into "Keyboard" mode,
2. display the contents of important vectors while stopped,
3. determine the size of all vectors by using either the `size()` function or the `whos` command.
4. and, lastly, modify variables while in the "Keyboard" mode of the debugger.

```
function [xx,tt] = coscos( f1, f2, fs, dur )
% COSCOS   multiply two sinusoids
%
t1 = 0:(1/fs):dur;
t2 = 0:(1/f2):dur;
cos1 = cos(2*pi*f1*t1);
cos2 = cos(2*pi*f2*t2);
xx = cos1 .* cos2;
tt = t1;
```

3.3 Piano Keyboard

Section 5 of this lab will consist of synthesizing a simple musical piece.[2] Since these signals require sinusoidal tones to represent piano notes, a quick introduction to the layout of the piano keyboard is needed. On a piano, the keyboard is divided into octaves—the notes in one octave being twice the frequency of the notes in the next lower octave. The white keys in each octave are named A through G. In order to define

[2]If you have little or no experience reading music, don't be intimidated. Only a little music knowledge is needed to carry out this lab. On the other hand, the experience of working in an application area where you must quickly acquire new knowledge is

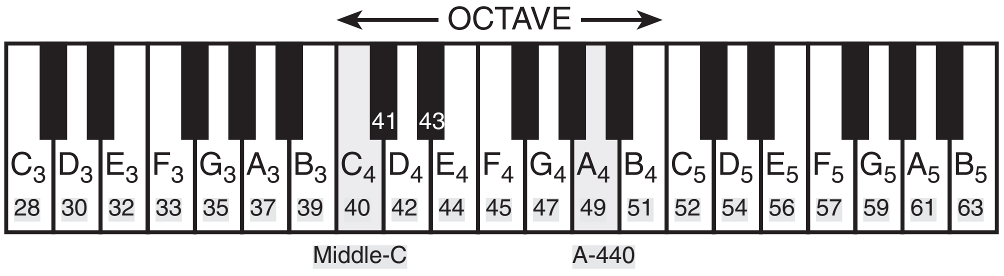

Figure 2: Layout of a piano keyboard. Key numbers are shaded. The notation C_4 means the C-key in the fourth octave, which is middle-C.

the frequencies of all the keys, one key must be designated as the reference. Usually, the reference note is the A above middle-C, called A-440 (or A_4) because its frequency is 440 Hz.[3] Each octave contains 12 notes (5 black keys and 7 white) and the ratio between the frequencies of the notes is constant between successive notes. As a result, this ratio must be $2^{1/12}$. Since middle C is 9 keys below A-440, its frequency is approximately 261 Hz. Consult the text for even more details.

Musical notation shows which notes are to be played and their relative timing (half, quarter, or eighth). Figure 3 shows how the keys on the piano correspond to notes drawn in musical notation. The white keys are labeled as A, B, C, D, E, F, and G; but the black keys are denoted with "sharps" or "flats." A sharp such as $A^{\#}$ is one key number larger than A; a flat is one key lower, e.g., $A_4^{\flat}$ (A-flat) is key number 48.

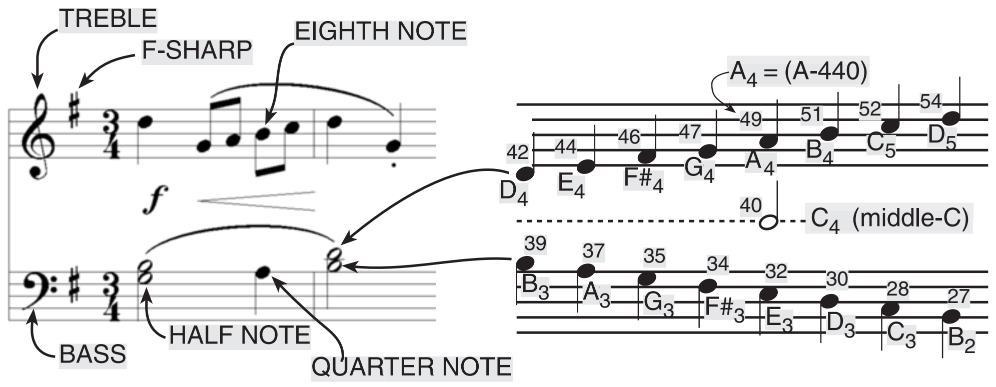

Figure 3: Musical notation is a time-frequency diagram where vertical position indicates which note is to be played. Notice that the shape of the note defines it as a half, quarter or eighth note, which in turn defines the duration of the sound.

Another interesting relationship is the ratio of fifths and fourths as used in a chord. Strictly speaking the fifth note should be 1.5 times the frequency of the base note. For middle-C the fifth is G, but the frequency of G is 391.99 Hz which is not exactly 1.5 times 261.63. It is very close, but the slight detuning introduced by the ratio $2^{1/12}$ gives a better sound to the piano overall. This innovation in tuning is called "equally-tempered" or "well-tempered" and was introduced in Germany in the 1760's and made famous by J. S. Bach in the "Well Tempered Clavier."

Thus, you can use the ratio $2^{1/12}$ to calculate the frequency of notes anywhere on the piano keyboard. For example, the E-flat above middle-C (black key number 43) is 6 keys below A-440, so its frequency should be $f_{43} = 440 \times 2^{-6/12} = 440/\sqrt{2} \approx 311$ Hertz.

a valuable one. Many real-world engineering problems have this flavor, especially in signal processing which has such a broad applicability in diverse areas such as geophysics, medicine, radar, speech, etc.

[3] In this lab, we are using the number 40 to represent middle C. This is somewhat arbitrary; for instance, the Musical Instrument Digital Interface (MIDI) standard represents middle C with the number 60.

4 Warm-up

4.1 Debug coscos.m

Download the file `coscos.m` and use the debugger to find the error(s) in the function. Call the function with the test case: `[xn,tn] = coscos(2,3,20,1)`. Use the debugger to:

1. Set a breakpoint to stop execution when an error occurs and jump into "Keyboard" mode,
2. display the contents of important vectors while stopped,
3. determine the size of all vectors by using either the `size()` function or the `whos` command.
4. and, lastly, modify variables while in the "Keyboard" mode of the debugger.

```
function [xx,tt] = coscos( f1, f2, fs, dur )
% COSCOS   multiply two sinusoids
%
t1 = 0:(1/fs):dur;
t2 = 0:(1/f2):dur;
cos1 = cos(2*pi*f1*t1);
cos2 = cos(2*pi*f2*t2);
xx = cos1 .* cos2;
tt = t1;
```

4.2 Note Frequency Function

Write an M-file to produce a desired note for a given duration. Your M-file should be in the form of a function called `key2note.m`. Your function should have the following form:

```
function xx = key2note(X, keynum, dur, fsamp)
% 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 = sampling frequency, use 8000, 11025 or 22050 Hz
%
tt = 0:(1/fsamp):dur;
freq =            %<=============== fill in this line
xx = real( X*exp(j*2*pi*freq*tt) );
```

For the `freq =` line, use the formulas 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. Notice that the `xx = real( )` line generates the actual sinusoid as the real part of a complex exponential at the proper frequency.

Instructor Verification (separate page)

4.3 Synthesize a Scale with Octaves

In a previous section you completed the `key2note.m` function which synthesizes the correct sinusoidal signal for a particular key number. Now, use that function to finish the following incomplete M-file that will play a minor scale where each tone is the sum of two notes separated by an octave. For the `tone =` line, generate the *two sinusoids*, one for `keynum`, the other for a key that is one octave higher. The sinusoids would be generated by making calls to the function `key2note()` written previously. It is important to point out that the code in `minorScaleWithOctaves.m` allocates a vector of zeros large enough to hold the entire scale then **inserts** each note into its proper place in the vector `xx`.

```
%--- minorScaleWithOctaves.m
%---
scale.keys =   [ 37  39  40  42  44  45  48  49 ];
%--- NOTES: A   B   C   D   E   F   G#   A
%  key #40 is middle-C
%
scale.durations  = 0.25 * ones(1,length(scale.keys));
fs  = 11025;                    %-- or 8000 Hz
xx  = zeros(1, sum(scale.durations)*fs+length(scale.keys) );
n1  = 1;
for kk = 1:length(scale.keys)
   keynum = scale.keys(kk);

   twoNotes =                      %<============= FILL IN THIS LINE

   n2 = n1 + length(twoNotes) - 1;
   xx(n1:n2) = xx(n1:n2) + twoNotes;   %<=== Insert the sound which is the sum of two notes
   n1 = n2 + 1;
end
soundsc( xx, fs )
```

Show that you have created the correct signals by making a spectrogram in which you can see the individual notes. Listening to the signal will also help while testing the code.

Instructor Verification (separate page)

5 Lab: Sinusoidal Chop Sticks

For this project, a very simple song must be synthesized. Two outputs must be produced for the lab report: a sound file for listening that can be evaluated, and a spectrogram of the synthesized signal in which the individual notes can be identified. Before starting the project, make sure that you have a working knowledge of the relationship between musical notes, piano key numbers and their frequencies (from the Pre-Lab).

5.1 Chop Sticks

"Chop Sticks" is one piano piece that everyone can play. A sequence of key-pairs is played, using the white keys in the octave starting at middle-C. The actual notes are given below:

> The keys to be played are F_4 and G_4 (simultaneously) six times, then E_4 and G_4 six times, then D_4 and B_4 six times, then C_4 (middle-C) and C_5 four times, and finally, D_4 and B_4 once and E_4 and A_4 once. Then these 24 key-pairs make one cycle which can be repeated.

5.2 Sinusoidal Synthesis of the Music Signal

The synthesis should be carried out with sinusoids. Two operations will be needed to create the song: addition of two sinusoids to handle the case where two notes are played simultaneously, and concatenation of signals to make the sequence of 24 notes for one cycle of Chop Sticks.

In the process of actually synthesizing the music, keep the following things in mind:

(a) The sampling frequency that will be used to play out the sound through the D-to-A system of the computer should be 11025 Hz.

(b) The time duration needed for each note-pair should be in the range of 150–250 msec. All note-pairs can have the same duration.

(c) There should be a short interval of silence between successive note-pairs; 50–100 msec should be sufficient to hear separate notes in time.

(d) To determine the frequency (in hertz) for each note see Fig. 2 and the discussion of the well-tempered scale in the Pre-Lab.

(e) Set up your MATLAB synthesis program so that you can generate one cycle of Chop Sticks and play it for a listening evaluation by your TA at the beginning of the next lab. Vectorize to make it run fast.

5.3 Spectrogram of the Music

Musical notation describes how a song is composed of different frequencies and when they should be played. This representation is, in effect, a *time-frequency* representation of the signal that synthesizes the song. In MATLAB we can obtain a time-frequency representation from the synthesized signal itself by making a spectrogram with the MATLAB functions `spectrogram()`, `specgram()`, or `plotspec()`.

(a) Include a spectrogram image of your synthesized music—one cycle of Chop Sticks. Since the spectrogram M-files will scale the frequency axis to run from zero to half the sampling frequency, it will be necessary to "zoom in" on the region where the notes are. The frequency region should be 0 to 600 Hz. Consult `help axis` or `help zoom`, or use the zoom tool in the MATLAB figure window, to display and print this region.

(b) In order to see all the notes in the spectrogram, you might have to adjust the window length from its default value of 256 (used in most of the spectrogram M-files. Longer windows will give better frequency resolution, but if the window length is too long the spectrogram plot will become blurred along the time axis.

(c) Provide annotations on your spectrogram plot to show that you have the correct frequencies, and the correct number of notes.

5.4 Miscellaneous Comments

Nothing in this section has to be done for the lab report; these are just relevant comments.

5.4.1 Playing in a Different Key

The big innovation of the well-tempered keyboard is that songs could be played in different keys (C-major, G-major, etc.) by just shifting all the notes up (or down) by a fixed amount. If Chop Sticks is in C-Major, shows that you can play it in another key by adding a fixed integer to all the key numbers in the song. Does it still sound OK?

5.4.2 Musical Tweaks

The musical passage is likely to sound very artificial, because it is created from pure sinusoids. One annoying quality is a clicking sound that is caused by an abrupt beginning or end of a sinusoid. An envelope can be used to smooth the edges of the sinusoids. In addition, some harmonics could be added to enrich the sound. None of these are needed for the lab report. However, if you find the project interesting, these suggestions might lead you to further investigation.

5.4.3 Envelope

One major improvement comes from using an "envelope," where you multiply each pure tone signal by an envelope $E(t)$ so that it fades in and out.

$$x(t) = E(t)\cos(2\pi f_{\text{key}} t + \phi) \tag{2}$$

If an envelope is used it should "fade in" quickly and "fade out" more slowly. An envelope such as a half-cycle of a sine wave $\sin(\pi t/\text{dur})$ is simple to program, but it sounds poor.

A standard way to define the envelope function is to divide $E(t)$ into four sections: attack (A), delay (D), sustain (S), and release (R). Together these are called ADSR. The attack is a quickly rising front edge, the delay is a small short-duration drop, the sustain is more or less constant and the release drops quickly back to zero. Figure 4 shows a linear approximation to the ADSR profile. Consult help on `linspace()` or `interp1()` for functions that create linearly increasing and decreasing vectors.

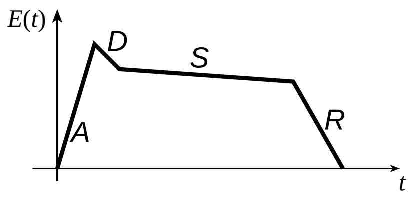

Figure 4: ADSR profile for an envelope function $E(t)$.

Lab #4
ECE-2025
Fall-2009
Instructor Verification Sheet

For each verification, be prepared to explain your answer and respond to other related questions that the lab TA's or professors might ask. Turn this page in at the end of your lab period.

Name: ______________________________ Date of Lab: __________

Part 4.1 Show that you can use the debugger on the text file `coscos.m`:

Verified:____________________ Date/Time:__________

Part 4.2 Complete and demonstrate the function `key2note.m`:

Verified:____________________ Date/Time:__________

Part 4.3 Complete and demonstrate the script file `minorScaleWithOctaves.m`. Demonstrate the spectrogram of the octave scale generated by `minorScaleWithOctaves.m`:

Verified:____________________ Date/Time:__________

Answered ITS questions:

Verified:____________________ Date/Time:__________

Sound Evaluation Criteria (When the report is turned in next week)

Does the file play the correct notes? All Notes _____ Most _____ Missing Pairs _____