GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2001
Lab #4: Synthesis of Sinusoidal Signals—Music Synthesis

Date: 5–8-Feb-01

This is *the official* Lab #4 description; it is similar to the one in Appendix C.3 of the text, but the piece *Funeral March of a Marionette* has been chosen for the synthesis.

This lab report will be worth 150 points.

The Warm-up section of each lab must be completed in Lab and the steps marked *Instructor Verification* must also be signed off **during the lab time**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. Turn in the completed Verification sheet **before** you leave the lab.

FORMAL Lab Report: You must write a formal lab report that describes your approach to music synthesis (Section 4).

The report will **due during the week of 12–15 Feb. at the start of your lab**.

---

1 Introduction

This lab includes a project on music synthesis with sinusoids. The piece *Funeral March of a Marionette* by Gounod[1] has been selected for doing the synthesis program. The project requires an extensive programming effort and should be documented with a complete **formal** lab report.[2] A good report should include the following items: a cover sheet, commented MATLAB code, explanations of your approach, conclusions and any additional tweaks that you implemented for the synthesis. Since the project must be evaluated by listening to the quality of the synthesized song, the criteria for judging a good song are given at the end of this lab description. In addition, it may be convenient to place the final song on a web site so that it can be accessed remotely by a lab instructor who can then evaluate its quality.

2 Overview

We have spent a lot of time learning about the properties of sinusoidal waveforms of the form

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

In this lab we will synthesize waveforms composed of sums of sinusoidal signals (1), sample them, and then reconstruct them for listening. Specifically, we will create a version of *Funeral March of a Marionette* using sinusoidal synthesis. If you would like to try other songs, the CD-ROM includes information about alternative tunes: *Minuet in G*, *Für Elise*, *Beethoven's Fifth*, *Jesu, Joy of Man's Desiring* and *Twinkle, Twinkle, Little Star*.

The primary objective of the lab is to establish the connection between musical notes, their frequencies, and sinusoids. A secondary objective is the challenge of trying to add other features to the synthesis in order

[1] See http://www.hnh.com/composer/gounod.htm.

[2] Refer to the Web-CT page for more details on the required format.

to improve the subjective quality for listening. Students who take this challenge will be motivated to learn more about the spectral representation of signals—a topic that underlies this entire course.

3 Warm-up: Music Synthesis

The instructor verification sheet is included at the end of this lab.

3.1 Debugging Skills

Testing and debugging code is a big part of any programming job, as you know if you been staying up late on the first few labs. Almost any modern programming environment provides a *symbolic debugger* so that break-points can be set and variables examined in the middle of program execution. Of course, many programmers insist on using the old-fashioned method of inserting print statements in the middle of their code (or the MATLAB equivalent, leaving off a few semi-colons). This is akin to riding a tricycle to commute around Atlanta.

In order to learn how to use the MATLAB tools for debugging, 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)`. Exhibit to the TA that you can:

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

4. Lastly, show that you can 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;
```

Instructor Verification (separate page)

3.2 D-to-A Conversion

The digital-to-analog conversion process has a number of aspects, but in its simplest form the only thing we need to worry about at this point 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 variable sampling rate if the hardware on the machine has such capability. A convenient choice for the D-to-A conversion rate is 8000 samples per second, so $T_s = 1/8000$ seconds. Another common choice is 11,025 Hz which is one-quarter of the rate used for audio CDs. Both of these rates should 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 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.

3.3 Piano Keyboard

Section 4 of this lab will consist of synthesizing the notes of a well known piece of music.[3] Since these signals require sinusoidal tones to represent piano notes, a quick introduction to the frequency 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. For example, the reference note is the A above

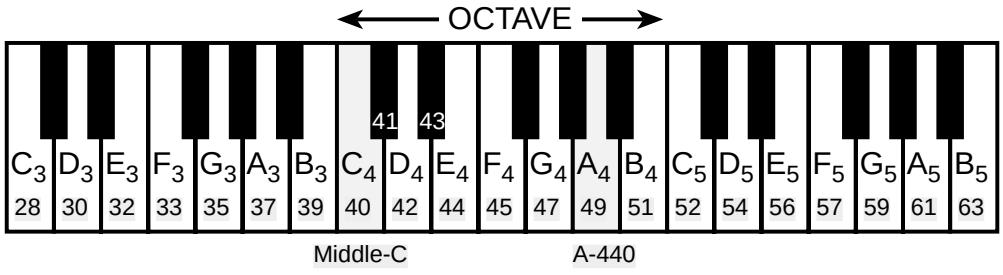


Figure 1: Layout of a piano keyboard. Key numbers are shaded. The notation C_4 means the C-key in the fourth octave.

[3] 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 knowledge is 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.

middle-C which is usually called A-440 (or A_4) because its frequency is 440 Hz. 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 chapter 9 for even more details.

Musical notation shows which notes are to be played and their relative timing (half, quarter, or eighth). Figure 2 shows how the keys on the piano correspond to notes drawn in musical notation. The white keys are all 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., $A_4^{\flat}$ is key number 48.

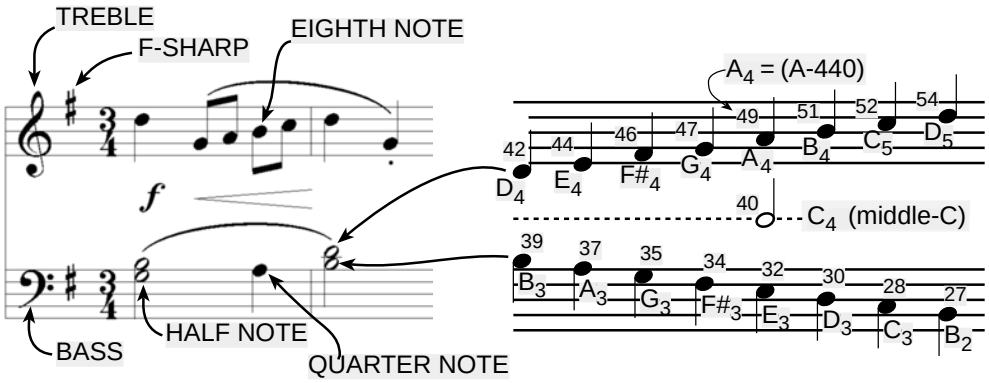


Figure 2: Musical notation is a time-frequency diagram where vertical position indicates which note is to be played. Notice that the shape of the note defines it as a half, quarter or eighth note, which in turn defines the duration of the sound.

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 about 392 Hz which is not exactly 1.5 times 261.6. 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 Clavichord."

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) 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.timeInterval = 0:(1/8000):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.

(b) From the previous lab you should have a `key2note.m` function that will synthesize the correct sinusoidal signal for a particular key number. It will be useful to have that function for doing the rest of the warm-up. The following is an incomplete M-file that will play scales:

```
%--- play_scale.m
%---
scale.keys =   [ 40  42  44  45  47  49  51  52 ];
%--- NOTES: C   D   E   F   G   A   B   C
%  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);

   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 )
```

For the `tone =` line, generate the actual sinusoid for `keynum` by making a call to the function `key2note()` written previously. It is important to point out that the code in `play_scale.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`.

Instructor Verification (separate page)

3.4 Spectrogram: Two M-files

In this part, you must synthesize an F-major chord consisting of the notes F_4, A_4 and C_5 and then display the spectrogram computed from the resulting time signal. Remember that the spectrogram displays an image that shows the *frequency* content of the synthesized *time* signal. Its horizontal axis is time and its vertical axis is frequency.

(a) Generate a signal for the F-major chord that is at least 0.8 seconds long. Use $f_s = 11025$ Hz. This can be done quickly with `add_cos()` or `fsynthesis()` from the previous labs.

(b) Use the function `specgram(xx,1024,fs)`. Zoom in to see the three separate frequencies (`help zoom`). The second argument[4] is the *window length* which could be varied to get different looking spectrograms. The spectrogram is able to "see" the separate spectrum lines with a longer window length, e.g., 1024 or 2048.[5]

Instructor Verification (separate page)

[4] If the second argument is made equal to the "empty matrix" then its default value of 256 is used.

[5] Usually the window length is chosen to be a power of two, because a special algorithm called the FFT is used in the computation. The fastest FFT programs are those where the signal length is a power of 2.

(c) If you are working at home, you might not have the `specgram()` function because it is part of the "Signal Processing Toolbox." In that case, use the function `plotspec(xx,fs)` which can be downloaded from Web-CT (you also need to download `spectgr.m`).[6] Show that you get the same result as in part (b). Explain why the result is correct. If necessary, add a grid so that frequencies can be measured accurately.
• Note: The argument list for `plotspec()` has a different order from `specgram`, because `plotspec()` uses an optional third argument for the *window length* (default value is 256).

4 Lab: Synthesis of Musical Notes

The audible range of musical notes consists of well-defined frequencies assigned to each note in a musical score. Five different pieces are given in the book, but we have chosen a different one for the synthesis program in this lab. Before starting the project, make sure that you have a working knowledge of the relationship between a musical score, key number and frequency. In the process of actually synthesizing the music, follow these steps:

(a) Determine a sampling frequency that will be used to play out the sound through the D-to-A system of the computer. This will dictate the time T_s between samples of the sinusoids.

(b) Determine the total time duration needed for each note, and also determine the frequency (in hertz) for each note (see Fig. 1 and the discussion of the well-tempered scale in the warm-up.) A data file called `marionnotes.mat` will be provided with this information stored in MATLAB structures. A second file called `marionshort.mat` has the same information for the first 3 measures of the piece. Both of these files are contained in a ZIP archive called `marion.zip` which is linked from the lab page.

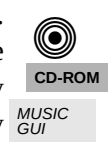
(c) Synthesize the waveform as a combination of sinusoids, and play it out through the computer's built-in speaker or headphones using `soundsc()`.

(d) Make a plot of a few periods of two or three of the sinusoids to illustrate that you have the correct frequency (or period) for each note.

(e) Include a spectrogram image of a portion of your synthesized music—probably about 3 or 4 secs—so that you can illustrate the fact that you have all the different notes. Since the spectrogram M-files will scale the frequency axis to run from zero to half the sampling frequency, it might be useful to "zoom in" on the region where the notes are. Consult `help zoom`, or use the zoom tool in MATLAB-v5.3 figure windows.

4.1 Spectrogram of the Music

Musical notation describes how a song is composed of different frequencies and when they should be played. This representation can be considered to be a *time-frequency* representation of the signal that synthesizes the song. In MATLAB we can can compute a time-frequency representation from the signal itself. This is called the spectrogram, and its implementation with the MATLAB function `specgram()` or `plotspec()`. To aid your understanding of music and its connection to frequency content, a MATLAB GUI is available so that you can visualize the spectrogram along with musical notation. This GUI also has the capability to synthesize music from a list of notes, but these notes are given in "standard" musical notation, not key number. For more information, consult the `help` on `musicgui.m` which only runs in MATLAB version 5.



[6] Actually, you should download the ZIP file with all the new/updated M-files for ECE-2025.

4.2 *Funeral March of a Marionette*

Funeral March of a Marionette is a familiar piece of music by the French composer Gounod. Where have you heard this music before? The first few measures are shown in Fig. 3, and the whole score that you must synthesize can be found on the class website.

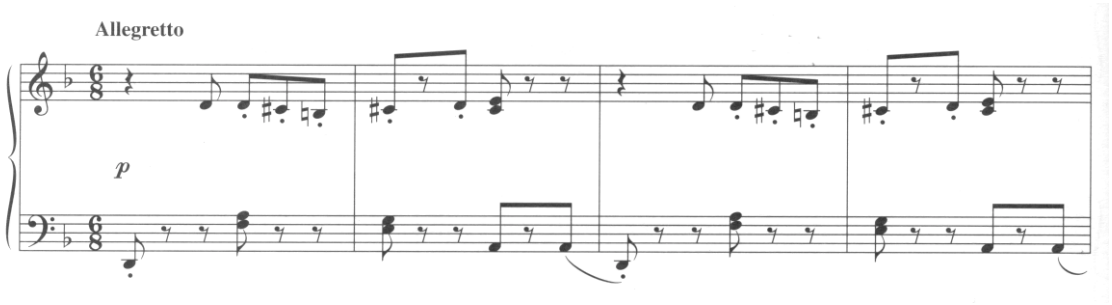


Figure 3: First few measures of the piece *Funeral March of a Marionette*.

You must synthesize the entire piece *Funeral March of a Marionette* by using sinusoids.[7] Therefore, you must determine the notes that are played for the entire passage, map each note to a key number and then synthesize sinusoids to recreate the piece.

4.3 Data File for Notes

Fortunately, a data file called `marionnotes.mat` has been provided with a transcription of the notes and information related to their durations. The data files `marionnotes.mat` and `marionshort.mat` are contained in a ZIP archive called `marion.zip` which is linked from the lab page. The format of a MAT file is not text; instead, it contains binary information that must be loaded into MATLAB. This is done with the `load` command, e.g.,

```
load marionnotes.mat
```

After the load command is executed a new variable will be present in the workspace, called `theMelodies`. Do `whos` to see that you have this new variable.

The variable `theMelodies` is a vector whose elements are structures. Each structure gives information about a single melody in the song. For example, `theMelodies(1)` might contain information about the treble notes (the notes with high key numbers), while `theMelodies(2)` might contain information about the bass notes (the notes with low key numbers). Some melodies contain only a few notes; they add harmony at a few locations in the the song, but are otherwise silent. You can determine the number of melodies in the song by calculating the length of the vector `theMelodies` with the command `length(theMelodies)`. This number will also equal the maximum number of notes that are ever simultaneously played in the song.

Each structure `theMelodies(i)` has three fields: `Keys, StartBeats, and Durations`. A typical structure `theMelodies(i)` looks like

```
theMelodies(i).Keys       =  [ # # # # ...  ]  % Key number
theMelodies(i).StartBeats =  [ # # # # ...  ]  % Starting beat
theMelodies(i).Durations  =  [ # # # # ...  ]  % Duration in beats
```

[7] Use sinusoids sampled at 11025 samples/sec (a lower sampling rate could be used if you have a computer with limited memory).

The value of `theMelodies(i).Keys(j)` is a single note's key number. The note's starting beat and duration in beats is given by the corresponding element in the other two fields.

Measures and *beats* are the basic time intervals in a musical score. A *measure* is denoted in the score by a vertical line that cuts from the top to the bottom of one line in the score. For example, in Fig. 3 there are three such vertical lines dividing that part of the musical score into four measures. Each measure contains a fixed number of *beats* which, in this case, equals six — the number of eighth notes in a measure. The label "6-8" at the left of Fig. 3 describes this relationship and is called the *time signature* of the song. It says that there are six beats per measure and that a single beat is the length of one eighth note.

For example, typing `theMelodies(1).Keys(6)` at the MATLAB command prompt returns the number 42, which describes the D in the second measure. Because the note is an eighth note and an eighth note is one beat, its duration, given by `theMelodies(1).Durations(6)`, equals one. The value of `theMelodies(1).StartBeats(6)` is nine because this note begins at the ninth beat from the beginning of the song.

4.3.1 Timing

From your recollection of this music, estimate the time duration (in seconds) needed for each note. You can define a time duration for quarter notes, eighth notes, and so on, but you still may need to make adjustments in the timing. A reasonable guess for the time of a quarter note might be one quarter of a second (0.25 sec.), but that is likely to make the piece sound very slow. You should write the code so that note duration is a global parameter that can be changed easily. For example, you might let the duration of a quarter note (see Fig. 2) be defined with the statement:

```
q_dur = 0.20;
```

where `q_dur` is in seconds. Then calculate all other durations in terms of `q_dur`. Half notes are twice as long as quarters; eighth notes are half as long. If the quarter note duration is defined only once, then it could be changed: for example, setting `q_dur = 0.10` would make the whole piece play twice as fast.

Another timing issue is related to the fact that when a musical instrument is played, the notes are not continuous. Therefore, inserting very short pauses between notes usually improves the musical sound because it imitates the natural transition that a musician must make from one note to the next. An envelope (discussed below) can accomplish the same thing.

4.4 Musical Tweaks

The musical passage is likely to sound very artificial, because it is created from pure sinusoids. Therefore, you might want to try improving the quality of the sound by incorporating some modifications. For example, you could multiply each pure tone signal by an envelope $E(t)$ so that it would fade 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 because it does not turn on quickly enough, so simultaneous notes of different durations no longer appear to begin at the same time. 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.

Some other issues that affect the quality of your synthesis include relative timing of the notes, correct durations for tempo, rests (pauses) in the appropriate places, relative amplitudes to emphasize certain notes

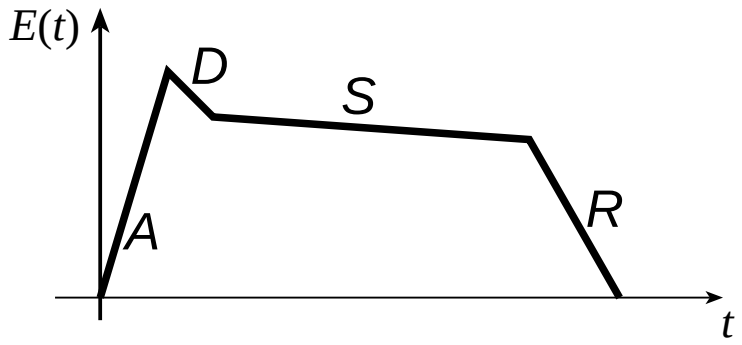


Figure 4: ADSR profile for an envelope function $E(t)$.

and make others soft, and harmonics. Since true piano sounds have a second and third harmonic content, and we have been studying harmonics, this modification would be simple, but be careful to make the amplitudes of the harmonics smaller than the fundamental frequency component. Furthermore, if you include too many higher harmonics, you might violate the sampling theorem and cause *aliasing.* You should experiment to see what sounds best.

4.5 Programming Tips

You may want to modify your `key2note()` function to accept additional parameters describing amplitude, duration, etc. In addition, you might choose to add an envelope and/or harmonics. Chords are created on a computer by simply adding the signal vectors of several notes. Although we have provided a MATLAB file containing the note values and durations for *Funeral March of a Marionette*, you are free to modify the duration values or add notes if you think it will improve the quality of the synthesized sound.

Lab #4
EE-2025
Spring-2001
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 3.1 Show that you can use the debugger on the text file `coscos.m`:

Verified:_____ Date/Time:_____

Part 3.3 Complete and demonstrate the script file `play_scale.m`:

Verified:_____ Date/Time:_____

Part 3.4 Demonstrate and explain the spectrogram of the F-major chord:

Verified:_____ Date/Time:_____

Sound Evaluation Criteria

Does the file play notes? All Notes _____ Most _____ Treble only _____

Overall Impression: _____

Excellent: Enjoyable sound, good use of extra features such as harmonics, envelopes, etc.

Good: Bass and Treble clefs synthesized and in sync, few errors, one or two special features.

OK: Basic sinusoidal synthesis, including the bass, with only a few errors.

Poor: No bass notes, or treble and bass not synchronized, many wrong notes.