GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2006
Lab #4: Synthesis of Sinusoidal Signals—Music Synthesis

Date: 7–13 Feb-06

FORMAL Lab Report: You must write a formal lab report that describes your approach to sound synthesis (Section 4). ***This lab report will be worth 150 points.***

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. You **MUST** complete the online Pre-Post-Lab exercise on Web-CT at the beginning of your scheduled lab session. You can use MATLAB and also consult your lab report or any notes you might have, but you cannot discuss the exercises with any other students. You will have approximately 20 minutes at the beginning of your lab session to complete the online Pre-Post-Lab exercise. The Pre-Post-Lab exercise for this lab includes some questions about concepts from the previous Lab report as well as questions on the Pre-Lab section of this lab.

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** by one of the laboratory instructors. After completing the warm-up section, turn in the verification sheet to your TA.

It is only necessary to turn in Section 4 as this week's lab report. More information on the lab report format can be found on Web-CT under the "Information" link. You should *label* the axes of your plots and include a title and Figure number for every plot. Every plot should be referenced by Figure number in your text discussion. In order to make it easy to find all the plots, include each plot *inlined* within your report. For more information on how to include figures and plots from MATLAB to your report file, consult the "Information" link on Web-CT. If you still do not know how to do so, ask your TA.

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 the submitted work should be original and it should be your own work.

The report will be **due during the period 15–21-Feb at the start of your lab.**

---

1 Introduction

This lab includes a project on sound synthesis with sinusoids. The project requires an extensive programming effort and should be documented with a complete **formal** lab report.[1] 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 sounds, the criteria for judging a good result are given at the end of this lab description.

The sound synthesis will be done 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 the challenge of trying to add other features to the synthesis in order to improve the subjective quality for listening. Many of these additional features are modifications to the spectrum, so it is necessary to learn more about the spectral representation of signals—a topic that underlies this entire course.

[1] Refer to the ECE-2025 Web-CT page for more details on the required format.

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 voltage waveform that will be amplified for 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

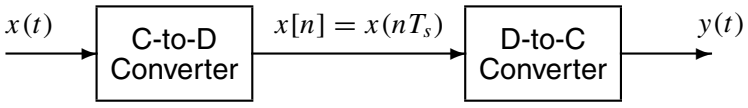

Figure 1: Sampling and reconstruction of a continuous-time signal.

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 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 input samples and interpolates a smooth curve between them. The *Sampling Theorem* tells us that if the input signal $x(t)$ is a sum of sine waves, then 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., $f_s > 2f_{\max}$. 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 analog-to-digital (A-to-D) converter and a 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 if the hardware on the machine has such capability. A convenient choice for the D-to-A conversion rate is 11025 samples per second,[2] 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 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`.

(a) 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.

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

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/second, and compute a total number of samples equivalent to a time duration of 0.5 seconds. 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 should use the `multi_sines()` function from a previous lab for this part (with modification of the sampling rate).

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 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/second. Describe how the *duration* and *pitch* of the signal were affected. Explain.

2.3 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.

2.4 Debugging Skills

Testing and debugging code is a big part of any programming job, as you know if you've 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. Nonetheless, many programmers still 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.

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;
```

2.5 Piano Keyboard

Section 4 of this lab will consist of synthesizing notes in one octave of a musical scale[3] 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 the frequencies of all the keys, one key must be designated as the reference. Usually,

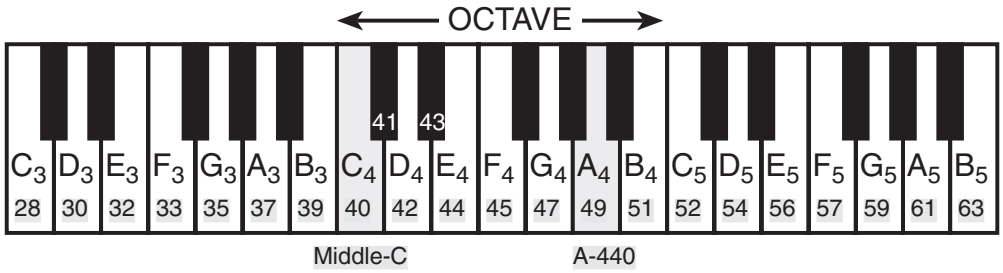

Figure 2: Layout of a piano keyboard. Key numbers are shaded. The notation C_4 means the C-key in the fourth octave.

the reference note is the A above middle-C, called A-440 (or A_4) because its frequency is 440 Hz. (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). 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.

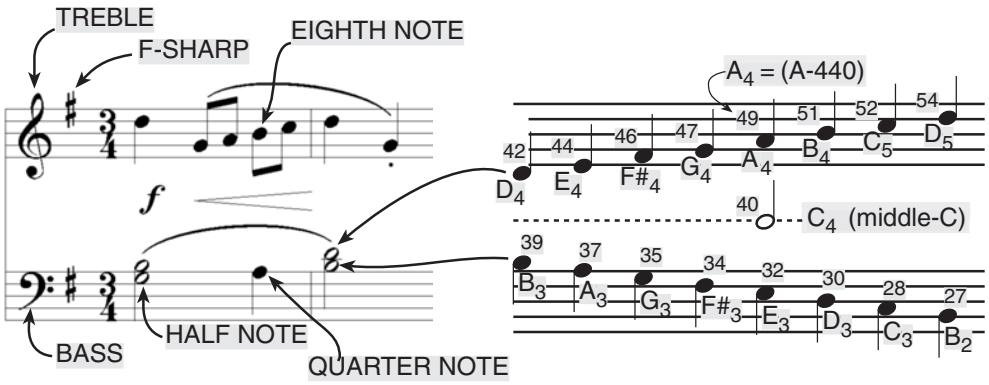

Figure 3: Musical notation is a time-frequency diagram where vertical position indicates which note is to be played. Notice that the shape of the note defines it as a half, quarter or eighth note, which in turn defines the duration of the sound.

Another interesting relationship is the ratio of fifths and fourths as used in a chord. Strictly speaking the

[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 new 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.

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.

2.6 Gaussian Forms

The Gaussian form is used in many different fields, but most often in probability where it is called the Gaussian distribution.[4] In this lab, we will use the Gaussian to control the amplitude weighting of sinusoids.

$$g(v) = \alpha e^{-(v-\mu)^2/2\sigma^2}$$

where v is the independent variable. The plot of a Gaussian is the well-known "bell curve" where μ determines the peak location and σ the width of the bell-shaped peak. The plots in Fig. 4 show the general shape for three values of σ. Notice that width of the bell when measured at the $e^{-1/2} = 0.607$ level is equal to 2σ,

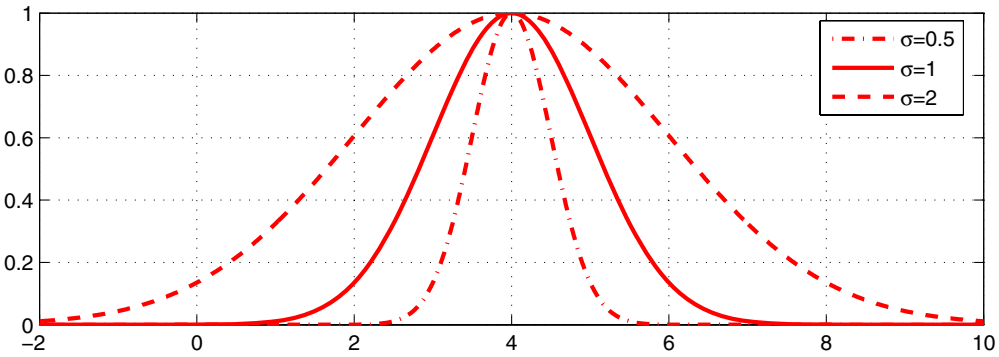

Figure 4: Plot of a Gaussian with parameters $\alpha = 1$, $\mu = 4$, and $\sigma = \frac{1}{2},\ 1,\ 2$.

which confirms the role of σ as the width-control parameter. In our application we will use $\alpha = 1$, but for a probability distribution α is chosen so that the total area under the Gaussian is equal to one, which leads to the value of α being $\alpha = 1/(\sigma\sqrt{2\pi})$.

Write a few lines of MATLAB code to make a plot of $10e^{-(v-1)^2/2(3)^2}$ over the range $-10 \le v \le 10$.

3 Warm-up

3.1 Note Frequency Function

Now 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:

[4] Here's a link for more info: `http://mathworld.wolfram.com/NormalDistribution.html`.

```
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)

3.2 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 scales where each tone is the sum of two notes separated by an octave:

```
%--- play_scale_octave.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 *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 `play_scale_octave.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)

3.3 Spectrogram: Two M-files

In this part, you must display the spectrogram of the scale synthesized in the previous section. 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 the signal for the scale with `play_scale_octave.m`.

(b) Use the function `specgram(xx,512,fs)`. Zoom in to see the progression of notes up the scale (`help zoom`), and also the notes one octave higher. In addition, identify the note A-440 in your spectrogram. The second argument[5] 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.[6]

Instructor Verification (separate page)

(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 is part of the *SP-First* toolbox that can be downloaded from Web-CT. 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: A Musical Illusion

The objective in this lab project is to reproduce a musical illusion called *Shepard's Scale* in which the tones played seem to be continually rising forever, yet seem to stay within one octave. If you familiar with M. C. Escher's drawing of a perpetual staircase[7] called "Ascending and Descending," then this audio illusion is analogous. The illusion is referred to as "circularity in pitch judgement" in an article on the web site of the Acoustical Society of America (`http://asa.aip.org/demo27.html`). You can also find a demonstration audio file at the ASA web site.

The basis of Shepard's scale is to create notes as the sum of many individual sinusoids, all separated by octaves. By cleverly controlling the amplitudes of each sinusoid, the illusion is created as the notes in a scale (or within an octave) are played in succession. For example, to produce the note A-440 you would add many sinusoids at the frequencies of 220 Hz, 880 Hz, 110 Hz, 1760 Hz, 55 Hz, 3520 Hz, and so on. In addition, the amplitude of the sinusoids would be largest at 440 Hz, a little smaller at 220 and 880 Hz, even smaller at 110 and 1760 Hz, and continually smaller for frequencies farther away from 440 Hz. In other words, the *amplitudes are frequency dependent*.

If we followed the guide of the ASA web site, we would use a cosine amplitude weighting. However, we will use a Gaussian instead, because we can get nearly the same amplitude dependence, and we will have the σ parameter available to control the width of the Gaussian. For Shepard's scale, the Gaussian should be centered somewhere between 260 and 500 Hz, and σ should be fixed to get a fixed Gaussian shape for all notes synthesized. One last point is that the Gaussian must also be a function of $\log_2(f)$, instead of f.

The following steps should help you produce a working illusion:

(a) First of all, forget about the amplitude weighting. Synthesize a C-major scale, starting at middle-C, in which each note in the scale is accompanied by eight other notes separated by octaves, extending

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

[6] 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.

[7] The idea of the perpetual staircase is attributed to Roger Penrose and his father Lionel (1958).

four octaves below and four octaves above. For example, to synthesize A-440, you would create the sum of nine sinusoids at the frequencies 27.5, 55, 110, 220, 440, 880, 1760, 3520 and 7040 Hz. Use a sampling frequency of 22050 Hz, so that you have f_s greater than twice the highest frequency.

(b) Playing once through a scale requires that you play seven notes. However, we will need to play the scale over and over, so modify your code to play the C-Major scale five times (35 total notes). Adjust the length of the notes and introduce silence between the notes so that you can easily identify each one.

(c) Now introduce amplitude weighting using the Gaussian form centered somewhere between 260 and 500 Hz. The trick here is that the weighting must be a function of $\log_2(f)$, not f. Make a plot of the weighting function that you are going to use. Make the plot versus $\log_2(f)$ for the case when $\sigma = 2$.

(d) You should now be able to produce the illusion by using the weight function to modify the amplitudes of the nine octave-related sinusoids that are added together for each note. Experiment with the σ parameter to get a good sounding illusion.

(e) Make a spectrogram of playing the scale three times and explain the illusion from the features that you can identify in the spectrogram.

(f) *Variations:* Play every note within the octave; all twelve of them and repeat. Does the illusion sound better in this case?

Lab #4
EE-2025
Spring-2006
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 Complete and demonstrate the function `key2note.m`:

Verified:_____ Date/Time:_____

Part 3.2 Complete and demonstrate the script file `play_scale_octave.m`:

Verified:_____ Date/Time:_____

Part 3.3 Demonstrate the spectrogram of the octave scale generated by `play_scale_octave.m`:

Verified:_____ Date/Time:_____

Sound Evaluation Criteria

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

Overall Impression: _____

Good: Correct illusion; proper use of Gaussian envelope; all notes synthesized and in sync.

OK: Basic sinusoidal synthesis, but the illusion is not obvious

Poor: Synthesis does not work.; incorrect illusion.