GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 2009

Lab #5: Synthesis of Sinusoidal Signals—Speech Synthesis

Date: 22-28 Sept-09

FORMAL Lab Report: You must write a formal lab report that describes your system for speech 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.

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.*

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 after Fall break.

1 Introduction

This lab includes a project on speech synthesis with sinusoids. The speech synthesis will be done with sinusoidal waveforms of the form

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

where each sinusoid will have *short duration* on the order of the pitch period of the speaker. One objective of this lab is to study how many sinusoids are needed to create speech utterances that sounds good. A secondary objective of the lab is the challenge of putting together the short duration sinusoids without introducing artifacts at the transition times. Finally, much of the understanding needed for this lab involves the spectrum representation of signals—a topic that underlies this signal processing course.

2 Pre-Lab

In this lab, the periodic waveforms and speech 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 D-to-A Conversion with `soundsc`

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). In MATLAB the `soundsc(xx,fs)` function will play out the sound stored in the vector `xx` and a sampling rate equal to `fs`. 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) 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 8000 samples/second, and compute a total number of samples equivalent to a time duration of 2 secs. Use `soundsc()` to play the resulting vector through the D-to-A converter of the your computer, and listen to the output.

(b) Now send the vector `xx` to the D-to-A converter again, but change the sampling rate parameter in `soundsc(xx, fs)` to 16000 samples/second. *Do not recompute the samples in* `xx`, just tell the D-to-A converter that the sampling rate is 16000 samples/second. Describe how the *duration* and *pitch* of the signal were affected. Explain.

2.2 Synthesizing a Signal from Short Frames

In speech synthesis, we will create the overall signal one *frame* at a time. A *frame* is a short section of the time-domain signal, e.g., $x_k(t) = x(t + kT)$ for $0 \le t \le T$. In this case, we can synthesize $x(t)$ from all the short-frame signals by adding them together in the correct order; this requires time-shifting. A mathematical notation for adding many short signals that are time-shifted is given by

$$x(t) = \sum_{k=0}^{K-1} x_k(t - kT)$$

where each short-frame signal $x_k(t)$ is shifted by the amount kT. If each short-frame signal has a duration of T, then the shifted signals $x_k(t - kT)$ do not overlap, and we actually create $x(t)$ by *concatenating* the short frames $x_k(t)$ one after the other. With concatenation, the total duration of $x(t)$ would be KT.

Consider the case where the short-frame signals are sinusoids

$$x_k(t) = \cos(2\pi(411 + 100k)t) \qquad \text{for } 0 \le t < T$$

In order to concatenate eight of these sinusoids each with a duration of $T = 0.22$ secs, run the MATLAB code below to make the signal which will have a duration of 1.76 s.

```
fs = 8000;
tt = 0:1/fs:0.22;
M = 8;
L = length(tt);
xx = zeros(1,M*L);
for k = 1:M
   jkl = (k-1)*L + (1:L);
   xx(jkl) = xx(jkl) + cos(2*pi*(411+100*k)*tt);
end
```

The synthesized signal will have changing frequency content (vs. time) which can be verified by displaying a spectrogram. One problem with this synthesis by concatenation is that the transition from one section to the next might not be smooth. Examine a plot of $x(t)$ and zoom in to see the jumps at multiples of T. We can join short-frame signal segments together smoothly if we use "windowing."

2.3 Windowing

In signal processing a *window* is a finite-duration signal $w(t)$ that multiplies another signal $x(t)$ to show the behavior of $x(t)$ over a short duration. The simplest example is the *rectangular window* defined by

$$w_r(t) = \begin{cases} 1 & 0 \le t \le T \\ 0 & \text{elsewhere} \end{cases}$$

If we multiply the signal $x(t)$ by $w_r(t)$ we get a "windowed segment" of $x(t)$ that has the signal values over the interval $[0,T]$, and is zero everywhere else. Furthermore, we can time-shift the window to extract other windowed segments of $x(t)$, e.g.,

$$w_r(t-13)x(t) = \begin{cases} x(t) & 13 \le t \le T+13 \\ 0 & \text{elsewhere} \end{cases}$$

The window can also have values other than zero and one, in which case it does weighting in addition to isolating a segment of the signal. For example, the triangular window is defined via

$$w_\Delta(t) = \begin{cases} 1-2|t/T-\frac{1}{2}| & 0 \le t \le T \\ 0 & \text{elsewhere} \end{cases} \tag{2}$$

where the subscript Δ is used because it looks like a triangle. The advantage of a window with weighting is that it will taper the edges of the windowed segment. This strategy is used in the spectrogram where many successive windowed segments are Fourier analyzed. Tapering the window edges produces a better time-frequency spectrogram image.

3 Warm-up

The objective of the warm-up is to show how we can join short-frame signal segments together smoothly by using "windowing."

3.1 Triangular Window

Sometimes it is necessary to modify the values of a signal to taper the ends. This can be accomplished with what is called a *window function.* One of the simplest window functions is the *triangular window* $w_\Delta(t)$ defined for duration T in (2).

(a) Draw a sketch (by hand) of $w_\Delta(t)$ for the case $T = 50$ ms.

(b) Write a MATLAB function that will generate (samples of) a triangular window. Since sampling at $t = 0$ or $t = T$ would give a zero value, generate the time vector at $t = 0.5/f_s, 1.5/f_s, 2.5/f_s, \ldots$. Use the following MATLAB code as a template:

```
function [win,tt] = myTri( T, fs )
% MYTRI make a triangular window
%
%   T = duration of the window
%   fs = sampling rate
%  win = values of the window at the times in the tt vector
tt = 0.5/fs : 1/fs : T;
win = ???              <==== add code here
```

(c) Test your `myTri` function by making a MATLAB plot of a triangular window for the case $T = 50$ ms, using a sampling rate of 8000 samples/s. How long is the window in number of samples?
Note: it is best if the length is even.

Instructor Verification (separate page)

3.2 Overlapped Signal Segments

In Section 2.2, you created a long signal by concatenating short segments called *frames*. A second method of forming the long signal is to overlap the frames before adding. First of all, we must study how to extract such overlapped frames from a long signal. Suppose that we start with a long signal $y(t)$ and we extract short segments from $y(t)$ in the following manner:

$$y_k(t) = \begin{cases} y(t+kT/2) & 0 \le t < T \\ 0 & \text{elsewhere} \end{cases} \qquad k = 0, 1, \ldots \tag{3}$$

In other words, the k^{th} frame, $y_k(t)$, starts at $t = kT/2$ and ends at $t = kT/2 + T$. In this manner, successive signal segments, such as $y_k(t)$ and $y_{k+1}(t)$, will have 50% overlap.

(a) ***Number of Segments:*** If the duration of $y(t)$ is 0.8 s and $T = 50$ ms, how many segments would be produced by the overlap method in (3)?

(b) ***Segment Length:*** If we are using MATLAB to represent the signal $y(t)$, then we would sample $y(t)$ at a rate f_s to produce a vector containing the samples, i.e., $y[n] = y(n/f_s)$. For example, if $f_s = 8000$ samples/s and the duration of $y(t)$ is 0.8 s, then the entire "y" vector would contain 6400 samples. How many samples are contained in each frame created by the overlap method in (3) if $T = 50$ ms and $f_s = 8000$ Hz?

(c) Write a short MATLAB function that will perform the segmentation according to (3). The function's output should be a matrix whose column length is the number of samples in one frame, and whose row length is the number of frames. Here is a template:

```
function yframes = frame50overlap( yy, T, fs )
% FRAME50OVERLAP   Fill matrix columns with signal frames
%                     overlapped by 50%
%   yy = (long) input signal
%   T = duration of the frame window
%   fs = sampling rate
% yframes = matrix containing segmented signal
%
LThalf = round(fs*T/2);  LT = round(fs*T);
Ly = length(yy);
n1 = 1;
n2 = LT;
frameCnt = 0;
while( n2<Ly)
   frame = yy(n1:n2);
   frameCnt = frameCnt+1;
   yframes(:,frameCnt) = frame(:);  %--make sure frame is a column
   n1 = ???? ; %  <----- FILL IN CODE for updating n1 and n2
   n2 = ???? ; %  <-----
end
```

(d) Test your overlap function for $T = 50$ ms and $f_s = 8000$ samples/s on the following signal:

```
ytest = cos( 95*pi*(0:(1/8000):0.8) );
```

Use the MATLAB plotting function `strips(yframes(:,1:6))` to plot the first six columns (as rows in the plot). Compare the last half of each row with the first half of the next row to explain the 50% overlap.

$\boxed{\textbf{Instructor Verification} \text{ (separate page)}}$

3.3 Overlapped Windows

The triangular window has the following interesting property: when you add shifted windows, the result is one, except for the ends. This property can be stated mathematically as

$$\sum_{k=0}^{K-1} w_{\Delta}(t-kT/2) = \begin{cases} \text{Rising} & 0 \le t < \frac{1}{2}T \\ 1 & \frac{1}{2}T \le t \le \frac{1}{2}KT \\ \text{Falling} & \frac{1}{2}KT < t < \frac{1}{2}(K+1)T \\ 0 & \frac{1}{2}(K+1)T \le t \end{cases} \tag{4}$$

The important line in equation (4) is the second one which says that the sum equals one in the intervals where the windows overlap, see Fig. 1.

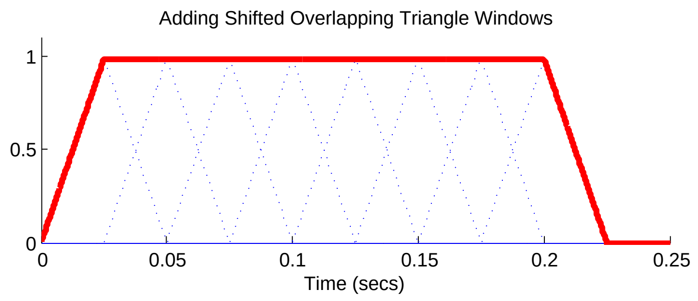

Figure 1: Sum of shifted and overlapped triangular windows. The continuous-time window duration is 50 ms. When the window is sampled at f_s, the length of the resulting discrete-time window must be an even integer for this property to hold exactly.

(a) Complete the code fragment below that will add together eight shifted triangular windows. It should produce a plot something like Fig. 1.

```
fs = 8000;
win = myTri( 0.050, fs);   %- T = 50 ms
win8 = zeros(1,5*length(win));
n1 = 1;
Lw = length(win);
ttx = (1:length(win8))/fs;
plot( ttx, win8 );
hold on
for k = 1:8
    n2 = ?????   %%<==== add code here
    win8(n1:n2) = win8(n1:n2) + win; %??? <==== FILL in n1 & n2
    plot((n1:n2)/fs,win,'--b')   %%<==== same range as above
    pause
    n1 = n1 + ??????????????? %%<==== add code here
end
hold on, plot( ttx, win8, '-r' ), hold off
```

Instructor Verification (separate page)

3.4 Add Overlapped and Windowed Segments

We can use the overlap property of the triangular window from eq. (4) in Section 3.3 to add back together the segments from the 50% overlap method of eq. (3). First of all, we would apply the triangular window to each segment, i.e., multiply the signal segment by the window $w_\Delta(t) y_k(t)$, and then we would add all of the segments together *with the correct shift.*

$$\sum_{k=0}^{K-1} w_\Delta(t-kT) y_k(t-kT)$$

The result will be equal to $y(t)$ except for the regions at the ends.

(a) Here is a code fragment that does the windowing on each frame:

```
% yframes = matrix containing the frames of the segmented signal
for k = 1:size(yframes,2)
  yframesWin(:,k) = yframes(:,k).*myTri(size(yframes,1)/fs,fs)';
end
```

(b) Write a `for` loop that will add the windowed segments back together to form `yy` over most of the time interval, except for the first and last $\frac{1}{2}T$ secs. Recall that the windowed frames are stored in the columns of a matrix, called `yframesWin` in the previous part. Then the code in Section 3.3(a) can be modified to perform the overlapping additions

(c) Now test the entire process with the signal

```
xx = cos(2*pi*440*tt);
```

with a frame duration of 10 ms, 50% overlap, and $f_s = 8000$ Hz. Perform the three processing steps as: (1) break it into segments (refer to Section 3.2(c)), (2) window each segment with a triangular window (part (a) above), and (3) add the segments back together (part (b) above). Show that the result is a 440-Hz cosine, except for the first 5 ms and the last 5 ms.

4 Lab: Synthesis of Speech with Sinusoids

Speech signals are often “quasi-periodic” especially in vowel regions. Thus it is reasonable to expect that speech utterances might be synthesized from a few sinusoids. On the other hand, fricatives such as /s/ and /sh/ sounds are not sinusoidal, so there are probably regions where the sinusoidal synthesis would do a poor job. Fortunately, the *intelligibility* of an utterance depends mostly on the vowels and less on the fricatives.

Speech can be synthesized by adding together a (relatively) small number of short-duration sinusoids. One important factor in the sinusoidal synthesis of speech is the *frame duration,* which is the time interval during which one set of sinusoids is used. From frame to frame the sinusoids can change. In speech, there are two time durations that would be relevant to picking the frame duration: the speaker’s pitch period and the average (or maximum) articulation rate of most humans The *pitch period* varies with individuals and with sex—adult males generally have a lower pitch (frequency) than adult females and hence the pitch period is longer for an adult male speaker. Typical values for the pitch period are approximately 5–10 ms. The *articulation rate* is a measure of how fast a speaker can form different sounds and is dictated by how fast the muscles in the vocal tract can move to form different sounds. For example, try saying the alphabet (A-B-C-D-E-F) as fast as you can. It is generally accepted that the individual sounds can change no faster than every 40–50 ms. Taken together the pitch frequency and articulation rate determine how often we should try changing the sinusoids for the speech synthesis.
Note: longer frames give excellent frequency resolution, but shorter frames track temporal changes better.

Data Format for Analyzed Signals

Any (speech) signal can be encoded with the MATLAB function `extractSinus` (i.e., the analysis function that extracts sinusoidal components from a signal) which has the following calling format:

```
function [Camps,Freqs] = extractSinus(xFrames,fs,numSines)
%extractSinus   will produce sinusoidal components per frame for a signal
%
% usage:  [Camps,Freqs] = extractSinus(xFrames,fs,numSines)
%
%     xFrames = input signal broken into 50% overlapped frames (by frame50overlap)
%     fs = sampling rate  (samples per sec)
%     numSines = # of sinusoids to find (only the positive freqs are counted)
% outputs:
%     Camps = array of complex amps (numSines by number of frames)
%     Freqs = array of freqs, one for each complex amp
```

The output of `extractSinus` gives the frequencies and complex amplitudes needed in each frame; the inputs are the number of sinusoidal components to extract, the sampling rate, and the *framed signal matrix* produced by the your `frame50overlap` M-file which depends on the frame duration, frame overlap, and sampling rate. In the (output) complex-amplitude and frequency arrays, the value of `Freqs(j,n)` is the j^{th} frequency (Hz) in the n^{th} frame of the signal; the corresponding complex amplitude is `Camps(j,n)`.
Note-1: `numSines` is the *maximum* number of frequency-domain peaks that will be found; if the actual number is less, then zeros will be inserted in the `Camps` and `Freqs` arrays to fill out the columns.
Note-2: If the outputs need to be ordered by amplitude or frequency, see `help sort` for an M-file that can sort a matrix.
Note-3: The analysis function `extractSinus` is provided as “p-code”, so it can be run like an M-file even though the actual code cannot be viewed.
Challenge: Write your own `extractSinus` M-file. It requires one call to the FFT function on each frame of the signal, followed by a peak picking function that finds the biggest spectrum components.

A variety of input (speech) signals will be needed during this lab project, including recordings of your own voice. There are at least three ways to get a speech signal into MATLAB, depending on the format:

1. Use the `wavread` function to get the signal from a `WAV` file, e.g., `[xx,fs]=wavread('catsdogs.wav');`.

2. Use the `load` command to load in data from a `MAT` file, which is MATLAB's binary format. For example, `load s7.mat`

3. Use `audiorecorder` to record directly from a microphone into MATLAB.

In addition, you can write a `WAV` file from MATLAB with the `wavwrite` function. Use `help wavwrite` for more info, and be careful that you scale the signal's amplitude to be between ± 1. Scaling of an arbitrary vector of numbers can be accomplished by finding the maximum absolute value, e.g.,

```
xxScaled = xx*( newMax/max(abs(xx)) );
```

4.1 Speech Analysis

The function `extractSinus` can be used to carry out two interesting measurements on your own voice(s): pitch frequency and vowel analysis. For this purpose it will be necessary to make one or two recordings of your own speech.

(a) For all team members, make a recording of each member speaking one isolated vowel, e.g., "aah", "ee", "u" as in hut, "a" as in bat, etc.

(b) To determine the pitch frequency, make two complementary measurements: (1) in the time waveform isolate a few periods in the middle of the vowel sound and measure the period in milliseconds, and (2) from a spectrogram of the vowel sound measure the fundamental frequency by examining the harmonic line spectrum. Pick the window length for the spectrogram to be much longer than one pitch period so that you get sufficient resolution of the spectral lines. Compare the values from these two measurements, and discuss whether or not they are consistent with one another.

(c) For the vowel sound, determine how many sinusoids would be needed to resynthesize just the vowel. Tabulate the frequencies and complex amplitudes needed; keep the number of components to a minimum. The frequency components should be more or less constant, but if they move during the vowel, make a note of that, and give an average value.

(d) Then, using the small number of sinusoids determined in the previous part, resynthesize a few periods of the vowel waveform (vs. t) and compare to the original time-domain waveform. Discuss how closely you are able to match the original waveform—it doesn't have to be perfect. Recall the examples of the "bat" signal from lecture.
Note: the function `extractSinus` gives you the complex amplitudes and frequencies of the "N" largest components, so you can use it to get the information needed to resynthesize the vowel.

4.2 Speech Synthesis Function

The next step is to write a general signal synthesis function and evaluate its use on a few different signals. The input comes from the analysis function `extractSinus` which extracts a set of complex amplitudes and frequencies for sinusoids in each short frame of a signal. The synthesis step requires that you write a function to create frames of the signal from a limited number of sinusoidal components given as a list of complex amplitudes and frequencies, and then add the frames together. For the synthesis process, the following general comments are relevant to the work that will have to be done in the next few subsections.

1. The analysis function `extractSinus` is provided as "p-code", so it can be run like an M-file.

2. 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 $T_s = 1/f_s$ between samples of the sinusoids.

3. Synthesize the speech waveform as a combination of overlapped (or concatenated) short-duration frames, and play it out through the computer's built-in speaker or headphones using `soundsc()`. The duration of the short sinusoids in each frame is fixed by the frame duration.

4. Consider the possibility that the phase of the `Camps` array may not be needed. You would expect to use the complex values from `Camps` to make the short sinusoids, but you can experiment with using only the magnitude of `Camps` to form the sinusoids. The rationale for this is the statement that "the ear is insensitive to phase." If you want to take this one step further, replace the phases of `Camps` with random phases, see `help rand`.

5. In order to evaluate the analysis-synthesis process, it is useful to make a spectrogram image of a portion of each synthesized utterance to compare with the spectrogram of the original. One or two seconds of the signal should be sufficient, so that you can illustrate whether or not you have used the correct number of sinusoids. In effect, you can use the spectrogram to confirm the correctness of your synthesis. The window length in the spectrogram might have to be adjusted, but start with an initial value of 256 for the window length in `spectrogram()` or `plotspec()`, and then adjust it (most likely by increasing it).

   Note: by default, the spectrogram M-files will scale the frequency axis to run from zero to half the sampling frequency, so it might be useful to "zoom in" on the region where the signal's frequencies lie. Consult `help zoom,` or use the zoom tool in MATLAB figure windows.

4.2.1 Write and Test a Synthesis Function in MATLAB

Now, it's time to write a general MATLAB function that can the actual signal synthesis:

(a) Write a `mySynth` function that will take the outputs from `extractSinus` and produce an output signal in which the frames are overlapped and added with the triangular window developed in the Warm-up. You can use a function like `myaddcos` written in a previous lab to sum the complex exponentials within one frame (make sure it was vectorized for speed). However, if you wrote `myaddcos` with a time increment that depends on the highest frequency, then you will have modify it so that the time increment depends on the sampling rate, and the sampling rate should become one of the inputs.
Note: If the signal was analyzed with an overlap in `extractSinus`, then you must use a segmenting strategy like `overlap50` inside of `mySynth`. For the best sounding results, overlapping will be needed.

(b) Apply `extractSinus` to a recording[1] of your own voice which is sampled at $f_s = 8000\,\text{Hz}$. Use a frame duration of 30 ms, with a 15 ms overlap, and extract at least ten sinusoidal frequency components (per frame).[2] One suggested utterance is "We were away a year ago" which consists entirely of voiced sounds, so it should work well.

(c) Synthesize the speech using all the frequencies components obtained in the analysis of part (b). Compare spectrograms of the original and the resynthesized signal, and comment on the differences that you hear in the sounds, and see in the spectrograms.

4.3 Modifying Speech

In this section, you will implement signal modifications that depend on the fact that you have the signal decomposed into "atoms" that are concentrated in time and frequency. The atoms can be modified prior to synthesizing a new modified signal: frequencies can be raised or lowered, or the time duration stretched or squeezed. it will not be possible to make the modified signal sound perfect. Therefore, you will have to

[1]MATLAB (version 7) has a function called `audiorecorder` that can acquire sound from a microphone via a sound card. Consult the `help` on `audiorecorder` and read the examples. Make sure to save the data as "double" instead of "int16."

[2]The file `catdogs.wav` is also available, but you are not required to process it for your lab report.

be creative in figuring out the best choice of parameters to make the effect sound good. For example, you should experiment with different frame durations and different number of frequency components to get the best possible sound—it won't be perfect.

4.3.1 Time-Scale Modification of Speech

It is possible to resynthesize the speech signal so that it plays faster (or slower) without changing the apparent pitch; in other words, the result will sound like the same speaker talking very fast, or very slow.

(a) Describe your strategy for writing a time-scale-modified synthesis function. This can probably be done by adding one input parameter to your MATLAB synthesis function from the previous section.

(b) Illustrate that this works on two of your recordings; try to make the signal duration twice as long, and also half as long.

(c) Show that this sounds better than just playing the original faster or slower. Give spectrograms to verify that the frequencies have not moved.

4.3.2 Frequency Modification of Speech

It is possible to resynthesize the speech signal so that it plays at the same rate, but the speaker's pitch appears to be higher or lower. In other words, a female speaker could be transformed into a lower-pitch (male-sounding?) speaker, and vice versa.

(a) Describe your strategy for writing a frequency-scale-modified synthesis function. Your MATLAB function will need one more input parameter if all the frequencies are multiplied by a common factor. *Note:* When increasing frequencies, you should avoid aliasing the high frequency components.

(b) Illustrate that this works on two of your recordings; try to raise all the frequencies by a factor of 1.5 or more, and then try to reduce them all by a factor of 0.6 or more.

(c) Make a spectrogram of the synthesized signal to verify that the frequencies were changed correctly.

4.4 Testing

One more data set, `sigUnknownF09.mat`, will be posted as a test case for synthesis. This is a `.mat` file containing the arrays `Camps` and `Freqs` extracted from a long speech signal with $f_s = 8000\,\text{Hz}$, a frame duration of 30 ms, and 50% overlap. You should run your synthesis algorithm with the objective of trying to understand what is being said.

Bring your working synthesis program to the lab when your report is due. A couple of test signals will be run to verify that you have a successful synthesis program, and that it can perform the time and frequency modifications described above. The format of the test signals will be the same as `sigUnknownF09.mat`, so make sure that your program can handle such inputs quickly and easily. In addition, you will be asked some questions about the inner workings of your MATLAB synthesis function(s).

4.5 Lab Report Suggestions

Your lab report should include spectrograms of the original and the synthesized signals. The write-up should point out features in the spectrograms that indicate the notable differences between the original and synthesized signals for the different cases. Figures should be annotated to point out these features; hand-written annotation are acceptable if written neatly.

Lab #5
EE-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 3.1 Make a triangular window M-file `myTri.m`:

Verified:____________________ Date/Time:__________

Part 3.2 Test overlapped signal segments:

Verified:____________________ Date/Time:__________

Part 3.3 Illustrate overlapped triangular windows:

Verified:____________________ Date/Time:__________

$\Longrightarrow$ Answered ITS questions:

Verified:____________________ Date/Time:__________ Completed Peer Evlauation?

Speech Evaluation Criteria

Are the sentences intelligible? All _____ Most _____ Only one _____

Do the time (and frequency) modifications work well?

Mystery sentence correct?

In-Lab test sentence correct?

Overall Impression: __________________________

Good: Good sound quality. Works well for both 4 and 8 sinusoids.

OK: Basic sinusoidal synthesis, but not smooth at the boundaries. Possible problem with windowing.

Poor: Synthesis does not work properly. Poor sound quality.