GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Fall 2003
Lab #4: Synthesis of Sinusoidal Signals—Speech Synthesis

Date: 16–22-Sept-03

FORMAL Lab Report: You must write a formal lab report that describes your approach to music 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 23–29 Sept. at the start of your lab.**

---

1 Introduction

This lab includes a project on speech synthesis with sinusoids. Several sentences have been selected for doing the synthesis program. 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 speech, the criteria for judging a good result are given at the end of this lab description.

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 a sentence 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 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 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 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

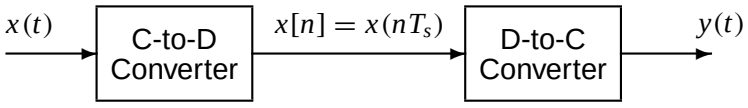

Figure 1: Sampling and reconstruction of a continuous-time signal.

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, human speech has 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.

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 Synthesizing a Signal from Sections

In speech synthesis, we will create the overall signal one section at a time. One way to do this is to add together a number of short signals

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

where each signal $x_k(t)$ has a duration of T. If the shifted signals $x_k(t - kT)$ do not overlap, then we would be creating $x(t)$ by concatenating the sections $x_k(t)$ one after the other and the total duration of $x(t)$ would be KT.

Consider the case where

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

In order to concatenate six sinusoids with $T = 0.3$ secs, run the MATLAB code below to make the signal which will have a duration of 1.8 s. Then the frequency content (vs. time) of the synthesized signal can be verified by displaying a spectrogram.

```
fs = 8000;
tt = 0:1/fs:0.3;
M = 6;
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
```

One problem with this synthesis is that the transition from one section to the next might not be smooth. Examine a plot of $x(t)$ to see the jumps at multiples of T.

3 Warm-up

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* defined[3] for duration T as

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

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

(b) Write a MATLAB function that will generate a triangular window. Since sampling at $t = 0$ or $t = 2T$ 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 comments as a template:

```
function [win,tt] = triwin( T, fs )
% TRIWIN   make a triangular window
%
%   T, so that 2T = duration of the window
%   fs = sampling rate
%  win = values of the window at the times in the tt vector
```

[3]The subscript Δ in $w_\Delta(t)$ is meant to convey the shape of the window.

(c) Test your `triwin` function by making a MATLAB plot of a triangular window for the case $T = 50$ millisec, using a sampling rate of 8000 samples/sec.

Instructor Verification (separate page)

3.2 Overlapped Signal Segments

In the Pre-Lab, you created a long signal by concatenating short segments. In this section, a second method of forming the long signal from short segments will be studied. Suppose that we start with a long signal $y(t)$ and we extract short segments from $y(t)$ in the following manner:

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

Thus, the k^{th} segment, $y_k(t)$, starts at $t = kT$ and ends at $t = (k+2)T$. Furthermore, successive signal segments, such as $y_k(t)$ and $y_{k+1}(t)$, have 50% overlap.

(a) If the duration of $y(t)$ is 1.5 sec. and $T = 50$ msec., how many segments would be produced by overlap method in (3)?

(b) 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/sec and the duration of $y(t)$ is 1.5 sec., then the entire "y" vector would contain 12000 samples. How many samples are contained in each segment created by the overlap method in (3)?

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

```
function yseg = overlap50( yy, T, fs )
% OVERLAP50   Fill matrix columns with signal segments
%                overlapped by 50%
%   yy = (long) input signal
%   T, so that 2T = duration of the window
%   fs = sampling rate
% yseg = matrix containing segmented signal
%
L = round(fs*T);  L2 = round(fs*T*2);
Ly = length(yy);
n = L2;
k=1;
while( n<Ly)
   ychop = yy(n1:n2);  ????<----- FILL IN CODE for n1 and n2
   yseg(:,k) = ychop(:);  %--make sure it's a column
   k = k+1;
   n = n + L;
end
```

(d) Test your function for $T = 25$ msec. and $f_s = 8000$ samples/sec. on the following signal:

```
yy = cos( 40*pi*(0:(1/8000):1.5) );
```

Explain why every other column of the `yseg` matrix is identical for this test case. Use the MATLAB plotting function `strips(yseg(:,1:6))` to plot the first six columns (as rows in the plot).

Instructor Verification (separate page)

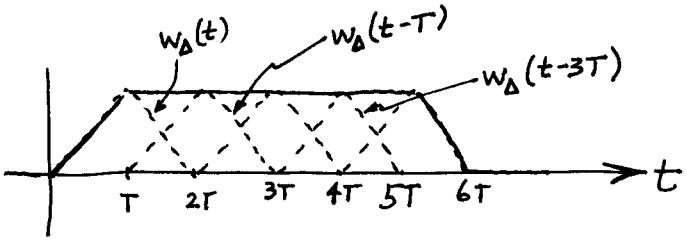

Figure 2: Sum of shifted and overlapped triangular windows.

3.3 Overlapped Windows

The triangular window has the following interesting property: when you add shifted triangular windows, the result is one, except for the ends, i.e.,

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

The important line in this equation is the second one which says that the sum equals one in the intervals where the triangles overlap, see Fig. 2.

(a) Complete the code fragment below that will add together six shifted triangular windows. It should produce a figure something like Fig. 2.

```
win = triwin( 0.05, 8000);
winsix = zeros(1,4*length(win));
n = 1;
Lw = length(win);
plot( winsix );
for k = 1:6
    winsix(n1:n2) = winsix(n1:n2) + win; %??? <==== FILL in n1 & n2
    hold on
    plot( winsix, '-r' )
    plot(n1:n2,win,'--b')   %%<==== same range as above
    hold off
    pause
    n = n + ?????????????? <==== add code here
end
```

Instructor Verification (separate page)

3.4 Add Overlapped and Windowed Segments

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

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

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:

```
% yseg = matrix containing segmented signal
for k = 1:size(yseg,2)
  ysegwin(:,k) = yseg(:,k).*triwin(size(yseg,1),1);
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 T secs.

(c) Now load the speech signal `catsdogs.wav` with the command:

```
yy = wavread('catsdogs.wav');
```

and perform the same three processing steps as above: (1) break it into segments, (2) window each segment with a triangular window, and (3) add the segments back together.

3.5 Spectrogram: Two M-files

In this part, you must display the spectrogram of the speech sound `catsdogs.wav` used 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) Use the function `specgram(xx,400,fs).` Zoom in to see details of the speech sounds (`help zoom`). The second argument[4] is the *window length* which could be varied to get different looking spectrograms.[5]

(b) 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: 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 bunch of short-duration sinusoids. Thus, an important factor in the sinusoidal synthesis of speech is the *frame length*, 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 length: the speaker's pitch period and the articulation rate of humans in general. The *pitch period* varies with individuals and with sex—adult males generally have a lower pitch than adult females and hence the pitch period is longer for an adult male speaker. The *articulation rate* is a measure or 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

[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.

(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 millisec. Taken together the pitch frequency and articulation rate determine how often we should try changing the sinusoids for the speech synthesis. We will use a frame length of 10 millisec. which is close to the pitch period.

In the process of actually synthesizing the speech, keep in mind the following general ideas:

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

(b) The total time duration needed for each sinusoid is fixed by the frame length and the percentage overlap. In the instructions below, two cases will be investigated: no overlap and 50% overlap.

(c) Data files will be provided with all the information stored in MATLAB structures. These files are contained in a ZIP archive called `speech4Lab.zip` which is linked from the lab page.

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

(e) Include a spectrogram image of a portion of each synthesized utterance—probably about 1 or 2 secs—so that you can illustrate the fact that 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 up, but start with an initial value of 256 for the window length in `specgram()`, or `plotspec()`.

Note: 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 frequencies are. Consult `help zoom`, or use the zoom tool in MATLAB figure windows.

4.1 Data File for Speech Signals

The speech data is encoded in a format that captures the frequencies and complex amplitudes needed in each frame; in addition, there are global parameters such as frame duration, frame overlap, and sampling rate. Several files are provided for different sentences. The data files are contained in a ZIP archive called `speech4Lab.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 sentence01.mat
```

After the load command is executed a new variable will be present in the workspace, called `SpSines`. Use `whos` at the command prompt to see that you have this new variable. The variable `SpSines` is a structures whose elements are either scalars or structures. The fields of the structure are

```
SPSines.name       =  identifier for the utterance
SPSines.fs         =  sampling rate used in the analysis
SPSines.framedur   =  duration of the frame (in sec.)
SPSines.overlap    =  percentage of frame overlap
SPSines.numframes  =  total number of frames in the sentence
SPSines.numsines   =  number of sines for each frame
SPSines.freqs      =  matrix of frequencies (in Hz); one column has
                        all the frequencies for one frame
SPSines.camps      =  matrix of complex amplitudes; one column has
                        all the complex amplitudes for one frame
```

The value of `SpSines.freqs(n,j)` is the n^{th} frequency (in Hz) in the j^{th} frame of the signal; the corresponding complex amplitude is `SpSines.camps(n,j)`.

Note: You can determine the length of the sentence by calculating the number of frames times the frame duration times the percentage overlap. Adding the frame duration will give the total duration in seconds.

4.2 Activities

In this lab, you will be given a set of complex amplitudes and frequencies of sinusoids for each small frame of speech signal, and your task is to use these coefficients to re-synthesize the speech signal. Speech processing involves a lot of parameter tweaking since a good speech processing algorithm has to accommodate for variability of human's speech. In this lab however, we will limit the variability to female and male speaker, frame overlap, and windowing.

The concept of frame overlap is crucial in this assignment. Since speech is a time-varying signal, it doesn't make sense to use a constant set of sinusoids to represent the whole speech signal. Instead, we are going to segment the speech into small frames, and synthesize sinusoids for each frame. When we shift to the next frame, we have to decide how much overlap we need.

In this assignment, for each speech signal you will be provided with two sets Fourier Series Coefficients, one with no-overlap and the other one with 50% overlap (half-overlap). If your algorithm works correctly, you will be able to determine which approach works better, 50% overlap or no-overlap.

(a) Load the file `Sp_msa1_no` (male speaker up to 8 frequencies, no overlap).

(b) Synthesize the speech using all 8 frequencies.

(c) Synthesize the speech using the 3 frequencies with the largest amplitudes. See `help sort` to find the largest ones. It might be useful to write a MATLAB function called `reducedCoefficients` that returns the largest amplitude components.

(d) Load the file `Sp_msa1_half` (male speaker up to 8 frequencies, half overlap).

(e) Synthesize the speech using all 8 frequencies. Employ a triangular window for the 50% overlap.

(f) Synthesize the speech using the 3 frequencies with the largest amplitudes.

(g) Compare the results by showing spectrograms. Also, describe the quality of the synthesis.

(h) Repeat the steps above for the following two files which were made for a female speaker:

- `Sp_fsa1_no` (female speaker up to 8 frequencies, no overlap).
- `Sp_fsa1_half` (female speaker up to 8 frequencies, half overlap).

(i) Compare the quality of synthesis for the male and female speakers.

4.2.1 Mystery Signal

One more dataset, `Sp_mystery`, is included with the objective of trying to understand what is being uttered. Furthermore, the frame duration is different, so your algorithm will be tested for its generality.

4.2.2 Additional Questions

Here are some questions to consider when you write your report. Is windowing a crucial factor in speech synthesis ? Which approach to speech synthesize is better, 50% overlap or no-overlap?

After reducing the Fourier Series Coefficients to only 3 coefficients per frame, is the speech still intelligible? What is the quality compared to the ones with 8 coefficients? When reducing to only 3 coefficients, which speech signal, female speech or male speech, is affected more?

All data files were created with the same frame duration, but could you make a recommendation for shortening (or lengthening) the frame for female speakers?

Lab #4
EE-2025
Fall-2003
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 `triwin.m`:

Verified:_____ Date/Time:_____

Part 3.2 Test overlapped signal segments:

Verified:_____ Date/Time:_____

Part 3.3 Overlapped triangular windows:

Verified:_____ Date/Time:_____

Speech Evaluation Criteria

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

Mystery sentence correct?

Test sentence correct?

Overall Impression: _____

Good: Good sound quality. Works well for both 3 and 8 sinusoids. Correct choice of top 3 sinusoids.

OK: Basic sinusoidal synthesis, but 3 sinusoid case could be better.

Poor: Synthesis does not work. Poor sound quality.