GEORGIA INSTITUTE OF TECHNOLOGY

SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2005
Lab #5: Synthesis of Sinusoidal Signals—Speech Synthesis

Date: 16–22-Feb-05

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.

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 *informal* lab report.

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

The report will be **due during the period 23 Feb.–1-Mar. at the start of your lab.**

---

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 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.

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

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,[1] so $T_s = 1/11025$ seconds; another common choice is 8000 samples/sec. Both of these rates satisfy the requirement of *sampling fast enough* as explained in the next section. In fact, 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.

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.

2.2 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

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

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.
dbdown     - Change local workspace context.
dbmex      - Enable MEX-file debugging.
dbstack    - List who called whom.
dbstatus   - List all breakpoints.
dbstep     - Execute one or more lines.
dbtype     - List M-file with line numbers.
dbup       - Change local workspace context.
dbquit     - Quit debug mode.

When a breakpoint is hit, MATLAB goes into debug mode, the debugger
window becomes active, and 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;
```

2.3 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

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

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 actually 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 each with a duration of $T = 0.3$ 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 by concatenation 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

We can join signal segments together smoothly if we use "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* defined[2] 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 `linspace` or the colon operator to generate the lines on the sides of the triangle. Use the following comments as a template:

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

```
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
```

(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. How long is the window in number of samples?

Instructor Verification (separate page)

3.2 Overlapped Signal Segments

In Section 2.3, you created a long signal by concatenating short segments. A second method of forming the long signal is to use overlapped short segments. Here we will study how to extract such overlapped segments 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:

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

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) ***Number of Segments:*** 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) ***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/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) if $T = 50$ msecs. and $f_s = 8000$ Hz?

(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);
n2 = L2;
k=1;
while( n2<Ly)
   n1 =   ????   <----- FILL IN CODE for n1 and n2
   ychop = yy(n1:n2);
   yseg(:,k) = ychop(:);  %--make sure it's a column
   k = k+1;
   n2 = n2 + 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)

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. 1.

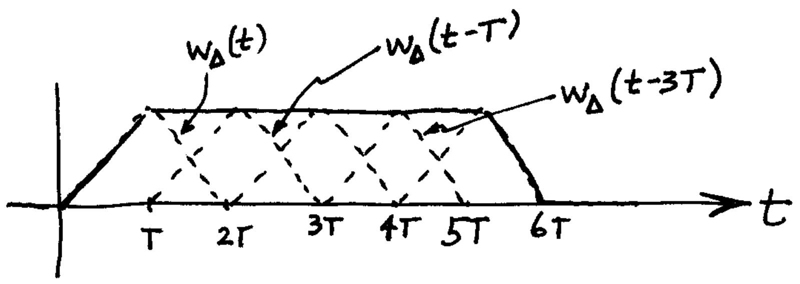

Figure 1: Sum of shifted and overlapped triangular windows. The window length must be even for this property to hold.

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

```
win = triwin( 0.05, 8000);
winsix = zeros(1,4*length(win));
n1 = 1;
Lw = length(win);
plot( winsix );
for k = 1:5
    n2 = ?????   <==== add code here
    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
    n1 = n1 + ?????????????? <==== 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 3.3) 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)/(2*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 T secs. Refer to Section 3.4(a) for sample code.

(c) Now test the entire process with the signal

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

with a segment duration of 10 millisec., 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 millisec. and the last 5 millisec.

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 (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 around 10 millisec. which is close to the pitch period of most speakers.

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.

(c) An analysis function `analyzeSpeech` is provided to extract sinusoidal components from a signal. It is given as "p-code", so it can be run like an M-file even though the actual code cannot be viewed.

(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.

Data Format for Speech Signals
Any speech signal can be encoded with the MATLAB function `analyzeSpeech` which has the following calling format:

```
function [Camps,Freqs] = analyzeSpeech(xx,fs,numSines,frameDur,overlapPct)
%ANALYZESPEECH   produce sinusoidal components for a speech utterance
%
% usage:  [Camps,Freqs] = analyzeSpeech(xx,fs,numSines,frameDur,overlapPct)
%
%   xx = input signal vector
%   fs = sampling rate  (samples per sec)
%   numSines = # of sinusoids to find (only the positives freqs are counted)
%   frameDur = duration of each frame in secs.
%   overlapPct = frame overlap expressed as a percentage
%   Camps = array of complex amps (numSines by number of frames)
%   Freqs = array of freqs, one for each complex amp
```

The output of `analyzeSpeech` gives the frequencies and complex amplitudes needed in each frame; the inputs are global parameters such as frame duration, frame overlap, and sampling rate. In the complex-amplitude and frequency arrays, the value of `freqs(n,j)` is the n^{th} frequency (in Hz) in the j^{th} frame of the signal; the corresponding complex amplitude is `camps(n,j)`.

There are three ways to get a speech signal into MATLAB, depending on the format:

1. Use the `wavread` function to load the speech data in from a `.WAV` file.
   For example, `[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.

4.1 Synthesis

In this lab, you will use the `analyzeSpeech` function to create a set of complex amplitudes and frequencies of sinusoids for each small frame of speech signal, and then write a function to re-synthesize the speech signal from a limited number of sinusoidal components.

(a) Write a `synthSpeech` function that will take the outputs from `analyzeSpeech` and produce a signal containing the speech. You can use a function like `add_cmplx_exp` written in a previous lab to sum the complex exponentials. However, if you wrote `add_cmplx_exp` 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 `analyzeSpeech`, then you must use a segmenting strategy like `overlap50` inside of `synthSpeech`.

(b) Apply `analyzeSpeech` to a recording[3] of your own voice which is sampled at $f_s = 8000$ Hz. Use a frame duration of 10 msec., 50% overlap, and extract 10 sinusoidal frequency components (per frame). One possible utterance is the vowels, i.e., "A-E-I-O-U".

(c) Synthesize the speech using all 10 frequencies components obtained in the analysis of part (b).

(d) Now, extract 3 frequency components and synthesize the speech (these will be the three largest ones).

(e) Your lab report can be relatively short, but show spectrograms of the original and the two synthesized speech signals. Point out features in the spectrograms that indicate the differences between the ten-sinusoid and three-sinusoid cases.

(f) Make subplots of the original signal and the resynthesized signal versus time, and identify where they are different by marking on the plots. The comparison will be easier if you scale the resynthesized speech, and the original signal, so that maximum value is one.

4.1.1 Mystery Signal

One more data set, `Smystery.mat`, is included as a challenge for synthesis. This is a `.MAT` file containing the arrays `Camps` and `Freqs` extracted from a speech signal with $f_s = 8000$ Hz, a frame duration of 8 msec., and 50% overlap. You should run your synthesis algorithm with the objective of trying to understand what is being uttered.

4.1.2 Testing

Bring your working synthesis program to the next lab. A couple of test signals will be run to verify that you have a successful synthesis program. The format of the test signals will the same as `Smystery.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).

[3]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."

Lab #5
EE-2025
Spring-2005
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 10 sinusoids.

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

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