GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2011
Lab #4: Synthesis of Sinusoidal Signals—Speech Synthesis

Date: 14–17 Feb-2011

FORMAL Lab Report: You must write a formal lab report that describes your system for speech synthesis (Section 5). ***This formal lab report will be worth 150 points.***

You should read the Pre-Lab section and do all the exercises in the Pre-Lab before your assigned lab time.

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.*

Electronic Submission: Please submit all M-files and electronic documents in a .zip file via **`t-square`**.

Peer Evaluation: Along with the lab report, each member of the team must fill out an on-line Peer Evaluation form that will be found within **`t-square`**.

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, but you cannot give or receive any written material or electronic files. In addition, you are not allowed to use or copy material from old lab reports from previous semesters. Your submitted work must be your own original work.

Two reports will be required for this lab:
An informal report on Section 4 **due during the period 21–24 Feb. at the start of your lab**,
A **Formal report** on Section 5 **due during the period 1–7-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 the number of sinusoids needed to create synthetic speech utterances that sound 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 entire 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 $0 \le t \le T$, where T is the section duration. 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 of the $x_k(t)$ signals. A mathematical notation for adding many short signal segments that are time-shifted is given by

$$x(t) = \sum_{k=0}^{K-1} x_k(t - kS) = x_0(t) + x_1(t - S) + x_2(t - 2S) + \ldots \tag{2}$$

where each short-frame signal $x_k(t)$ is time-shifted by the amount kS secs. There are two special cases: $T = S$, where each short-frame duration T is equal to the time shift S, and $T > S$ where the segments overlap in the summation of (2). When $T = S$ the shifted signals $x_k(t - kS)$ do not overlap, and the summation (2) actually creates $x(t)$ by *concatenating* the short frames $x_k(t)$ one after the other. With concatenation, the total duration of $x(t)$ would be KT, which is the same as KS.

Consider an example 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        %-- This FOR loop does the SUMMATION
   jkl = (k-1)*L + (1:L);
   xx(jkl) = xx(jkl) + cos(2*pi*(411+100*k)*tt);
end
```

This synthesized signal will have changing frequency content (vs. time) which can be verified by displaying a spectrogram of `xx`. 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 `xx` versus time, and zoom in to see the jumps at T, $2T$, $3T$, etc. On the other hand, it is possible to join short-frame signal segments together smoothly if we apply "windowing" to the segments, and the windowed segments are overlapped before adding them together.

2.3 Windowing

In signal processing a *window* is a finite-duration signal $w(t)$ that multiplies another signal $x(t)$ to isolate 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 a signal $x(t)$ by $w_r(t)$ we get a "windowed segment" of $x(t)$ that has the signal values of $x(t)$ over the interval $[0, T]$, and is zero everywhere else. If we want a different segment of $x(t)$, then we must time-shift the (rectangular) window before multiplying to extract other segments of $x(t)$, e.g.,

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

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

$$w_\Delta(t) = \begin{cases} 1 - \frac{2}{T}|t - \frac{1}{2}T| & 0 \le t \le T \\ 0 & \text{elsewhere} \end{cases} \tag{3}$$

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 which produces a better time-frequency spectrogram image. This strategy is usually the default in the spectrogram function where many successive windowed segments are Fourier analyzed.

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* and *overlapped* segments.

3.1 Triangular Window

Sometimes it is necessary to modify the values of a signal segment 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 (3).

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

(b) Write a MATLAB function that will create a structure `winTri` to hold all the information about the triangular window, including the values. Since it is convenient to have a window that is symmetric, 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 winTri= createTriWin( T, fs )
% createTriWin: create struct for a Triangular window with four fields:
% winTri.dur
% winTri.fsamp
% winTri.times
% winTri.values
%
% T = duration of the window
% fs = sampling rate
winTri.times = 0.5/fs :  1/fs :  T;
winTri.values = ?  <==== add code here
....more code...
```

(c) Test your `createTriWin` function by doing a MATLAB plot of a triangular window for the case $T = 20$ ms, using a sampling rate of 8000 samples/s. How long is this window in number of samples? *Note*: it is best if the window length is even.[1]

Instructor Verification (separate page)

3.2 Overlapping Input Signal Segments

In Section 2.2, you created a long signal by concatenating short signal segments called *frames*. A second method of forming the long signal is to *overlap the frames* before adding. We also need to do the reverse—extract overlapped short segments (frames) from a long signal. Suppose that we start with a long signal $s(t)$ and we extract short segments from $s(t)$ in the following manner:

$$s_k(t) = \begin{cases} s(t + kT/2) & 0 \le t < T \\ 0 & \text{elsewhere} \end{cases} \qquad k = 0, 1, \ldots \tag{4}$$

In other words, the k^{th} frame, or k^{th}signal segment, $s_k(t)$, starts at $t = kT/2$ and ends at $t = kT/2 + T$. Using (4), successive signal segments (frames), such as $s_k(t)$ and $s_{k+1}(t)$, will overlap each other by 50%, the second half of the k-th frame is identical to the first half of the $(k+1)$-st frame.

(a) ***Number of Segments:*** If the total duration of the long signal $s(t)$ is 0.8 s and the segment duration is $T = 20$ ms, how many segments would be produced by the overlap method in (4)?

(b) ***Segment Length:*** If we are using MATLAB to represent the signal $s(t)$, then we would sample $s(t)$ at a rate f_s to produce a vector containing the samples, i.e., $s[n] = s(n/f_s)$. For example, if $f_s = 8000$ samples/s and the duration of $s(t)$ is 0.8 s, then the entire $s[n]$ vector would contain 6400 samples. How many samples are contained in each frame created by the 50% overlap method in (4) if $T = 20$ ms and $f_s = 8000$ Hz?

(c) Write a short MATLAB function that will perform the segmentation according to (4). 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 outFrames = overlapFramesBy50( ss, T, fs )
%   Fill matrix columns with signal frames, overlapping by 50%
% ss = (long) input signal
% T = duration of the frame, ie, short segment duration
% fs = sampling rate
% outFrames = matrix containing segmented signal
%
LThalf = round(fs*T/2); LT = 2*LThalf;  %- should be = round(fs*T)
Lss = length(ss);
n1 = 1;
n2 = LT;
frameCnt = 0;
outFrames = [];
while( n2<Lss)    %-- this will ignore/drop an ending frame that is not full
   thisFrame = ss(n1:n2);
   frameCnt = frameCnt+1;
   outFrames(:,frameCnt) = thisFrame(:); %-- (:) makes sure that the frame is a column
   n1 = ????  ; % <----- FILL IN CODE for updating n1 and n2
   n2 = ????  ; % <-----
end
```

[1] In MATLAB it is possible to guarantee that the window length (in samples) will be an even integer: If you happen to choose `T` and `fs` so that `fs*T` is odd, then add one: `L = round(fs*T), L = L+rem(L,2), Tnew = L/fs`. In effect, this will lengthen the window duration (in secs) from `T` to `Tnew`, but the change is very small because `fs` is usually big.
Note: the window length is often called the frame length.

(d) Test your overlap function for $T = 20$ ms and $f_s = 8000$ samples/s on the following signal:

```
sigTest = cos( 95*pi*(0.5/8000:(1/8000):0.8) );  %-- offset the samples by 0.5/fs
```

Use the MATLAB function `strips(outFrames(:,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.

Instructor Verification (separate page)

3.3 Overlapped Windows

The triangular window has the following interesting property: when you shift the window by 50%, and then add the first half to the last half, the result is a constant. This property can be stated mathematically as

$$\sum_{k=0}^{K-1} w_{\Delta}(t - kT/2) = \begin{cases} \text{Rising} & 0 \le t < \frac{1}{2}T \\ c & \frac{1}{2}T \le t \le \frac{1}{2}KT \\ \text{Falling} & \frac{1}{2}KT < t < \frac{1}{2}(K+1)T \\ 0 & \text{elsewhere} \end{cases} \tag{5}$$

where c is a constant. The important line in equation (5) is the second one which says that the sum equals a constant c in the intervals where the windows overlap, see Fig. 1 (for the case of 50-msec. windows).

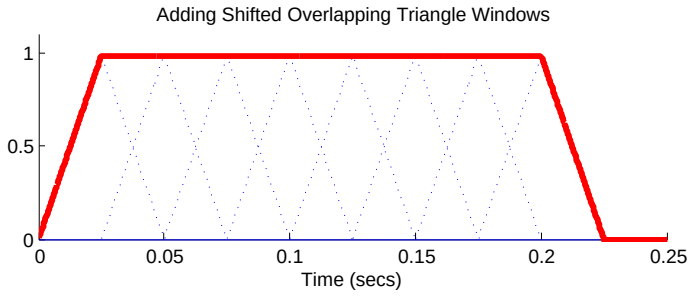

Figure 1: Sum of shifted and overlapped triangular windows has a constant region. The continuous-time window duration is 50 ms. When the window is sampled at f_s, the length of the resulting vector containing samples of the window must be an even integer for this property to hold exactly.

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

```
fs = 8000;
myTriWin = createTriWin( 0.020, fs);  %- T = 20 ms
Lw = length(myTriWin.times);
win8 = zeros(1,5*Lw);
n1 = 1;
ttx = (1:(5*Lw))/fs;
plot( ttx, win8 );
hold on
for k = 1:8
   n2 = ?????   %<==== add code here
   win8(n1:n2) = win8(n1:n2) + myTriWin.values;  %???  <==== FILL in n1 and n2
   plot((n1:n2)/fs,myTriWin.values,'--b')  %<==== same range as above
   pause
   n1 = n1 + ??????????   %<==== add code here
end
hold on, plot( ttx, win8, '-r' ), hold off
```

(b) Determine the value of c in (5) from the mathematics of the window definition. Then determine the value of c in (5) from the plot, and compare.

Instructor Verification (separate page)

3.4 Add Overlapped and Windowed Segments

We can use the overlap property of the triangular window from eq. (5) in Section 3.3 to add back together the segments from the 50% overlap method of eq. (4). First of all, we would apply the triangular window to each segment, i.e., multiply the signal segment by the window $w_{\Delta}(t)s_k(t)$, and then we would add all of the segments together *with the correct time shift.*

$$\sum_{k=0}^{K-1} w_{\Delta}(t-kT)s_k(t-kT) = c\,s(t) \quad ?$$

The result should be equal to $s(t)$ scaled by c except for the regions at the ends.

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

```
% ovFrames = matrix containing the overlapped frames of the segmented signal
myTriWin = createTriWin(size(ovFrames,1)/fs,fs);
for k = 1:size(ovFrames,2)
   ovFramesWin(:,k) = ovFrames(:,k).*myTriWin.values';
end
```

(b) Write a `for` loop that will add the windowed segments back together to form a signal vector `ssOut` that is (a scaled version) of the original 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 `ovFramesWin` in the previous part. Then the code in Section 3.3(a) can be modified to perform the overlapping additions of neighboring segments which are stored in neighboring columns.

(c) Now test the entire process with the signal

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

with a frame duration of 20 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 overlapping segments back together (part (b) above). Show that the result is a 440-Hz cosine, except for the first 10 ms and the last 10 ms. The amplitude should be the original amplitude multiplied by c.

Verified:_____ Date/Time:_____(Optional: makes the lab project easier)

4 Lab-A: Synthesis of Vowels 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 more 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 gender—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.

4.1 Pitch Frequency Measurement

Carry out two measurements on your own voice(s) to determine pitch frequency.

(a) For all team members, make a recording of each member speaking one isolated vowel; choose from vowels such as "aah", "ee", "u" as in hut, "a" as in bat, etc.

(b) To determine the pitch frequency, make two complementary measurements, one in the time domain and the other in the frequency domain. First, in a *plot of the time waveform* isolate three periods in the middle of the vowel sound which look *quasi-periodic*. Then measure the (pitch) period in milliseconds by locating identical features in each period. Convert this measurement to a pitch frequency in hertz.

(c) For the frequency-domain measurement, create a *time-frequency plot* of the signal by making a *spectrogram* of the vowel sound. Pick the window length for the spectrogram to be equivalent to the expected duration of a few pitch periods. The quasi-periodic nature of the vowel should give a *line spectrum* in the frequency domain. Measure the fundamental frequency by examining the spacing of the harmonic lines in the spectrogram image. Try to align this measurement to be at the same time as the measurement in the previous part.
Note: Longer frames give excellent frequency resolution, but shorter frames track temporal changes better. A larger window length would improve the resolution of the spectral lines and make it easier to measure the frequency. However, if the window length is too large the spectrogram loses its resolution because the frequency content of the vowel might be changing slightly over many pitch periods. Therefore, experiment with different window lengths.

(d) Compare the values from these two measurements, and discuss their consistency.

5 Lab-B: Modifying Speech During Synthesis

Any (speech) signal can be encoded with the MATLAB function `getSines2011` (i.e., the analysis function that extracts sinusoidal components from a signal) which has the following calling format:

```
function Xsines = getSines2011(xFrames,fs,numSines)
%getSines2011   will produce sinusoidal components per frame for a signal
%
% usage:  Xsines = getSines2011(xFrames,fs,numSines)
%
%     xFrames = matrix formed from input signal broken into into frames,
%                 e.g., 50% overlapped frames (done by overlapFramesBy50)
%     fs = sampling rate  (samples per sec)
%     numSines = # of sinusoids to extract (only the positive freqs are counted)
% OUTPUT structure (Xsines)
%     Xsines.camps  = array of complex amps (number of frames by numSines)
%     Xsines.freqs = array of frequencies, one per amp (number of frames by numSines)
%     Xsines.T = frame duration in secs used in the Fourier analysis.  (scalar)
%     Xsines.fs =sampling rate in Hz.  (scalar)
```

The output structure from `getSines2011` gives the complex amplitudes and frequencies needed in each frame, *sorted by decreasing magnitude of the complex amplitude.* The inputs are the number of sinusoidal components to extract, the sampling rate, and the *framed signal matrix* produced by the `overlapFramesBy50` M-file which depends on the frame duration, frame overlap, and sampling rate. In the complex amplitude and frequency output arrays, the value of `Xsines.freqs(n,p)` is the p^{th} frequency (Hz) in the n^{th} frame of the signal; the corresponding complex amplitude is `Xsines.camps(n,p)` for the positive frequency component in the spectrum.

Note-1: `numSines` is the *maximum* number of frequency-domain peaks to be found; if the actual number is less, then zeros will be inserted in the `Xsines.camps` and `Xsines.freqs` arrays to fill out the rows.

Note-2: The analysis function `getSines2011` is provided as "p-code", so it can be run like an M-file even though the actual code cannot be viewed.

Note-3: The fields `Xsines.T` and `Xsines.fs` were added to make the synthesis process easier. Their presence does not affect the operation of the `getSines2011` function.

Challenge: Write your own `getSines2011` 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 get data from a `MAT` file, MATLAB's binary format, e.g., `load s7.mat`
3. Use `audiorecorder` to record directly from a microphone into MATLAB. See `help audiorecorder`.

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

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

5.1 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 will come either from a WAV file, or from the analysis function `getSines2011` which extracts a set of complex amplitudes and frequencies in each short frame of a signal. The synthesis step requires that you write a function to create frames of the signal using a limited number of sinusoidal components from

getSines2011, and then add the frames together *smoothly*. 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. `sigOut = mySineSynth(Xsines,nSines,alpha,beta)` is the template for the synthesis function, where `sigOut` is a structure with two fields: `sigOut.vals` and `sigOut.fs`. The input parameters are `Xsines` from `getSines2011`, the number of sines `nSines` to be used in the synthesis, and two parameters for modifying the synthesis `(alpha, beta)` to be discussed later.

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. Although it would be natural to use the same f_s as in `getSines2011`, this is not necessary because you are making the output signal and can choose any valid f_s.

3. The speech waveform will be synthesized as a combination of overlapped (or concatenated) short-duration frames, and played out through the computer's built-in speaker or headphones using `soundsc()`. The duration of the short sinusoids in each frame is specified by the frame duration field in the `Xsines` structure. This field can be changed when longer frames are desired, i.e., when slowing down the speech.

4. Consider the possibility that the phase of the complex amplitudes `Xsines.camps` returned by `getSines2011` may not be needed. You would expect to use the amplitude and phase values to make the short sinusoids, but you can experiment with using only the amplitude 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 in `Xsines.camps` with random phases, see `help rand`.

5. Once you get the analysis-synthesis process working, the output signal should be evaluated. It is best to do the evaluation in the frequency domain, i.e., by using spectrograms. You should make a spectrogram image of a portion of synthesized utterance (the output signal), and then you can compare it to the spectrogram of the original over the same time interval. A few seconds of the signal should be sufficient, so that you can illustrate issues such as 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 (depending onf f_s), but start with an initial value of 512 for the window length in `spectrogram()` or `plotspec()`, and then adjust it up and down (most likely up).

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

5.1.1 Write and Test a Synthesis Function in MATLAB

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

(a) Write a `mySineSynth` function that will take the outputs from `getSines2011` 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 written in a previous lab to sum the sinusoids within one frame (make sure it was vectorized for speed).
Note: Since the signal was analyzed with an overlap in `getSines2011`, it is reasonable to use the same overlap-segmenting strategy inside of `mySineSynth`. For the best sounding results, overlapping will be needed. The MATLAB code will be similar to that in `overlapFramesBy50`.

(b) Apply `getSines2011` to a recording[2] of your own voice which is sampled at $f_s = 11025$ Hz. Run the analysis function `getSines2011` with a frame duration of 40 ms, a 20 ms overlap, and extract at least

[2] 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."

ten sinusoidal frequency components (per frame).[3] Good candidates for the test utterance should be short and should contain many vowels, e.g., "We were away a year ago" is often used.

(c) Synthesize the speech using all ten frequency components obtained in the analysis of part (b), using the same frame duration (40 ms) and overlap (20 ms).

(d) Compare spectrograms of the original and the resynthesized signal, and comment on the differences that you hear in the sounds, and/or see in the spectrograms.

(e) Now, extract only five frequency components and synthesize the speech (these should be the five largest ones). Compare spectrograms of the original and the resynthesized signal, and comment on the differences that you hear in the sounds versus the ten-frequency case.

5.2 Modifying Speech

In this section, you will study signal modifications that depend on the fact that you have the signal decomposed into "atoms" that are concentrated in time and frequency. Each atom has three parameters: complex amplitude, frequency, and time duration (which equals the frame duration used in the `getSines2011` analysis for normal synthesis). The atoms can be modified to synthesize different versions of the signal: e.g., frequencies can be raised or lowered, and time durations can be stretched or squeezed.

5.2.1 Time-Scale Modification of Speech

It is possible to resynthesize the speech signal so that it plays faster (or slower) without changing the pitch of the speaker. In other words, the result will sound like the same speaker talking very fast, or very slow. In this synthesis the time durations are multiplied by a factor α, e.g., if the original duration is 40 ms and $\alpha = 0.75$, the synthesis would be done with a shorter frame duration of 30 ms and a 50% overlap which is 15 ms. The other parameters should be kept the same: complex amplitudes and frequencies.

(a) Modify your `mySineSynth` function to have third input argument for the parameter α.

(b) Using the recording of your own voice tested in Section 5.1.1, do the synthesis for $\alpha = 0.8$ and $\alpha = 1.2$, with ten sinusoids. Make spectrograms, and compare them to the original. Listen to the synthesized signals, and describe how they sound versus the original.

(c) Try larger and smaller values of α, and listen for intelligibility. Discuss how much you can change α away from one, and still maintain some reasonable sound quality.

(d) Using the signal in `fedexMoschitta.wav`, perform the synthesis for $\alpha > 1$. Pick a number of sinusoids and a value of α that makes it much easier to understand John Moschitta who has been credited with being the world's fastest talker. Make a spectrogram of part of the synthesized signal, and compare it to the same part of the original.

5.2.2 Frequency Modification of Speech

It is possible to resynthesize the speech signal so that it plays at the same speaking rate, but all the frequencies are changed by a constant factor: $f_{\text{new}} = \beta f_{\text{old}}$. The other parameters, complex amplitudes and time durations, should be kept the same.

The file `Lab04s11.mat` contains a structure `Xs04s11` created by `getSines2011` for an utterance by Morgan Freeman. The analysis parameters f_s (sampling rate) and T (window duration) can be found in the structure. This encoded utterance must be used in this section. For the tests in part (b)–(d) below, keep α fixed at 1.0.

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

(a) Modify your `synthSines` function to have a fourth input argument for β, the frequency scaling. *Note:* When $\beta > 1$ you must avoid aliasing, so that no frequencies will become greater than $\frac{1}{2} f_s$. You can raise the sampling rate during synthesis, or you need to perform error checking of the scaled frequencies versus $\frac{1}{2} f_s$.

(b) Using `Xs04s11`, do the synthesis for $\beta = 1.0$, with twelve sinusoids. Make a spectrogram, and compare it to the original. Listen to the synthesized signal, and describe how it sounds versus the original.

(c) Using `Xs04s11`, do the synthesis for $\beta = 1.2$, with 12 sinusoids. Make a spectrogram, and compare it to the original. Listen to the synthesized signal, and describe how it sounds versus the original.

(d) Using `Xs04s11`, do the synthesis for $\beta = 0.8$, with 12 sinusoids. Make a spectrogram, and compare it to the original. Listen to the synthesized signal, and describe how it sounds versus the original.

(e) Using `Xs04s11`, do the synthesis for a time-scale modification of $\alpha = 1.33$ and $\beta = 1$, with 12 sinusoids. Make a spectrogram, and compare it to the original. Listen to the synthesized signal, and describe how it sounds versus the original.

(f) Discuss how all of these cases are related. Try to relate your listening comparisons to the spectrograms. Would changing the number of sinusoids improve the quality?

(g) *Experimentation:* How far from 1.0 can you make β? For example, $\beta = 2, 3 \ldots$? or, $\beta = 0.6, 0.3 \ldots$?

5.3 Testing in Lab

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 that produced by `getSines2011`. There might be various sampling rates f_s and frame durations T, 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).

5.4 Lab Report Suggestions

Your lab report should include spectrograms of the original and the synthesized signals whenever possible. 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 clearly to point out these features; hand-written annotations are acceptable if written neatly.

Lab #4
EE-2025 Spring-2011
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: _____

$\Longrightarrow$ Completed Peer Evaluation?
Uploaded ZIP file with .m and .doc files?

Part 3.1 Make a triangular window M-file `createTriWin.m`:

Verified:_____ Date/Time:_____

Part 3.2 Test overlapped signal segments:

Verified:_____ Date/Time:_____

Part 3.3 Illustrate overlapped triangular windows:

Verified:_____ Date/Time:_____

Part 3.4 (OPTIONAL) Add overlapped and windowed signal segments:

Verified:_____ Date/Time:_____

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 different numbers of sinusoids.

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

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