GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Spring 2025
Lab #2: Signal Synthesis: AM, FM and Chirp

Date: 27 Jan. – 07 Feb. 2025

The labs will be held in room 2440 of the Klaus building. Your GT login will work, if you specify the "Windows domain" to be AD. **Special Lab Instructions in Spring 2025:** The labs will be held in person (or remotely through Zoom when needed). Students registered in a particular lab session are required to be present at the designated times. Attendances will be taken before each class. A total of six labs will be conducted. Each lab will typically last two weeks. The first week is devoted to students' **Q&As** and the second is for students' **demos** to the instructors for codes and lab results. Students will be grouped into teams of 2 or 3 by instructors before starting Lab 1. For each lab session, there will be two instructors administrating all activities. Each team will be given a breakout period of 15 minutes in each session for Q&As and verifications. Students are encouraged to discuss lab contents in teamwork. At the end of each lab, each student are required to turn in an individual lab report in a single pdf fill, containing answers to all lab questions, including codes and plots. Georgia Tech's **Honor Code** will be strictly enforced. See CANVAS Assignments for submission instructions.

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.

1 Introduction

The objective of this lab is to introduce more complicated signals that are related to the basic sinusoid. These signals which implement frequency modulation (FM), to be discussed in the next lab, and amplitude modulation (AM), to be studied in this and next labs, are widely used in communication systems such as cellular telephones, radios and television broadcasts. In addition, they can be used to create interesting sounds that mimic musical instruments and useful tones. There are a number of demonstrations on the CD-ROM that provide examples of these signals for many different conditions.

FM Synthesis

2 Pre-Lab

We have spent a lot of time learning about the properties of sinusoidal waveforms of the form:

$$x(t) = A\cos(2\pi f_0 t + \varphi) = \Re\left\{\left(Ae^{j\varphi}\right)e^{j2\pi f_0 t}\right\} \tag{1}$$

Now, we will extend our treatment of sinusoids to more complicated signals composed of sums of sinusoidal signals. The objective of this lab is to learn how short-duration sinusoids can be concatenated to make longer signals that "play" musical notes and dial telephone numbers. The resulting signal can be analyzed to show its time-frequency behavior by using the *spectrogram.*

2.1 Using MATLAB

2.1.1 Cell Mode in MATLAB

MATLAB has a formatting syntax that allows you to produce documentation at the same time that you make an M-file. A quick summary is that double percent signs followed by a space (%%␣) are interpreted as

sections in cell mode so that your code is broken into natural blocks that can be run individually. In addition, the M-file can be "published" to an HTML file and then viewed as a nicely formatted web page. There are several sources with information about cell mode.

1. Videos:
   `https://www.youtube.com/watch?v=CWgl5Ylltxk`
   `https://www.youtube.com/watch?v=_TWBG95mCQU`
   and `https://www.mathworks.com/help/matlab/matlab_prog/run-sections-of-programs.html`

2.1.2 Functions

Functions are a special type of M-file that can accept inputs (matrices, vectors, structures, etc.) and also return outputs. The keyword `function` must appear as the first non-comment word in the M-file that defines the function, and that line of the M-file defines how the function will pass input and output arguments. The file extension must be lower case "m" as in `my_func.m`. See Section B-6 in Appendix B of the text for more discussion.

The following function has several mistakes (there are at least four). Before looking at the correct one below, try to find these mistake(s):

```
matlab mfile  [xx,tt] = badcos(ff,dur)
%BADCOS  Function to generate a cosine wave
%  usage:
%       xx = badcos(ff,dur)
%       ff = desired frequency
%      dur = duration of the waveform in seconds
%
tt = 0:1/(100*ff):dur;   %-- gives 100 samples per period
badcos = real(exp(2*pi*freeq*tt));
```

The corrected function should look something like:

```
function [xx,tt] = goodcos(ff,dur)
tt = 0:1/(100*ff):dur;   %-- gives 100 samples per period
xx = real(exp(2i*pi*ff*tt));
```

Notice that the word `function` must be at the beginning of the first line. Also, the exponential needs to have an imaginary exponent, and the variable `freq` must be defined before being used. Finally, the function has `xx` as an output, so the variable `xx` must appear on the left-hand side of at least one assignment line within the function body. In other words, the function name is *not* used to hold values produced in the function.

2.1.3 Structures in MATLAB

MATLAB can do structures. Structures are convenient for grouping information together. For example, we can group all the information about a sinusoid into a single structure with fields for amplitude, frequency and phase. We could also add fields for other attributes such as a signal name, the signal values, and so on. To see how a structure might be used, run the following program which plots a sinusoid:

```
x.Amp = 7;
x.phase = -pi/2;
x.freq = 100;
x.fs = 11025;   %-- sampling rate controls the spacing of values on the time grid
x.times = 0:(1/x.fs):0.05;
x.values = x.Amp*cos(2*pi*(x.freq)*(x.times) + x.phase);
x.name = 'My Signal';
x            %---- echo the contents of the structure "x"
plot( x.times, x.values )
title( x.name )
```

Notice that the fields in the structure can contain different types of variables: scalars, vectors or strings.

You can also have arrays of structures. For example, if `xx` is array of sinusoid-structures with the same fields as above, you would reference one of the sinusoids via:

```
 xx(3).name, xx(3).Amp, xx(3).freq, xx(3).phase
%
 plot( xx(3).times, xx(3).values )
 title( [xx(3).name,' Amp=',num2str(xx(3).Amp),' Phase=',num2str(xx(3).phase)] )
```

Notice that the array name is `xx`, so the array index, 3, is associated with `xx`, e.g., `xx(3)`.

2.2 Summation of Sinusoidal Signals

If we add several sinusoids, each with a different frequency (f_k), we cannot use the phasor addition theorem, but we can still express the result as a summation of terms with complex amplitudes via:

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \varphi_k) = \Re\left\{ \sum_{k=1}^{N} \left(A_k e^{j\varphi_k} \right) e^{j2\pi f_k t} \right\} \tag{2}$$

where $A_k e^{j\varphi_k}$ is the complex amplitude of the k^{th} complex exponential term. The choice of f_k will determine the nature of the signal—for amplitude modulation or beat signals we would pick two or three frequencies f_k that are very close together, see Chapter 3.

Therefore, it will be necessary to establish the connection between musical notes, their frequencies, and sinusoids. A secondary objective of the lab is to learn the relationship between the synthesized signal, its spectrogram and the musical notes. There are several specific steps that will be considered in this lab:

1. Synthesizing a single short-duration sinusoid with a MATLAB M-file, and adding it to an existing long signal vector.

2. Mapping piano keys to explicit frequencies using the "equally-tempered" definition of twelve notes within each octave.

3. Concatenating many short-duration sinusoids with different frequencies and durations.

4. *Spectrogram:* Analyzing the long (concatenated) signal to display its time-frequency spectral content.

2.3 Beat Signals: Summing Two Sinusoids with A Small Frequency Difference

In the section on beat notes in Chapter 3 of the text, we discussed signals formed as the product of two sinusoidal signals of slightly different frequencies; i.e.,

$$x(t) = B \cos(2\pi f_\Delta t + \varphi_\Delta) \cos(2\pi f_c t + \varphi_c) \tag{3}$$

where f_c is the (high) center frequency, and f_Δ is the (low) frequency that modulates the envelope of the signal. An equivalent representation for the beat signal is obtained by rewriting the product as a sum:

$$x(t) = A_1 \cos(2\pi f_1 t + \varphi_1) + A_2 \cos(2\pi f_2 t + \varphi_2) \tag{4}$$

It is relatively easy to derive the relationship between the frequencies $\{ f_1,\ f_2 \}$ and $\{ f_c,\ f_\Delta \}$.

2.4 AM, FM and Chirp Signals

2.4.1 Amplitude Modulation (AM)

By multiplying a time-varying signal $y(t)$ with a sinusoid we get an amplitude modulated (AM) signal:

$$x(t) = y(t) \cos(2\pi f_c t) \tag{5}$$

where f_c is called a carrier frequency. The beat is a special type of AM signal in which we pick a sinusoid with a low signal frequency for $y(t)$ and one high carrier frequency resulting in a signal $x(t)$ with an addition of two sinusoids with individual frequencies very close together, see Chapter 3.

2.4.2 Frequency Modulation (FM)

We will also examine signals whose frequency varies as a function of time. Recall that in a constant-frequency sinusoid (1) the argument of the cosine is $(2\pi f_0 t + \varphi)$ which is also the exponent of the complex exponential. We will refer to the argument of the cosine as the ***angle function***. In (1), the *angle function* changes *linearly* versus time, and its time derivative, $2\pi f_0$, equals the constant frequency of the cosine.

FM Synthesis

A generalization is available if we adopt the following notation for the class of signals with time-varying angle functions (and constant amplitude):

$$x(t) = A \cos(\psi(t)) = \Re\{A e^{j\psi(t)}\} \tag{6}$$

where $\psi(t)$ is the angle function. The time derivative of the angle function $\psi(t)$ in (6) gives a frequency that we call the *instantaneous radian frequency:*

$$\omega_i(t) = \frac{d}{dt}\psi(t) \qquad \text{(rad/sec)}$$

If we prefer units in hertz, then we divide by 2π to define the *instantaneous cyclic frequency:*

$$f_i(t) = \frac{1}{2\pi}\,\frac{d}{dt}\psi(t) \qquad \text{(Hz)} \tag{7}$$

2.4.3 Chirp, or Signals with Linearly Swept Frequency

Spectrograms & Sounds: Wideband FM

A linear-FM *chirp* signal is a sinusoid whose frequency changes linearly from a starting frequency value to an ending frequency. The formula for such a signal can be defined by creating a complex exponential signal with a quadratic angle function. Thus, we define $\psi(t)$ in (6) as the following quadratic function:

$$\psi(t) = 2\pi\mu t^2 + 2\pi f_0 t + \varphi$$

Using (7), we obtain an instantaneous *cyclic* frequency that changes *linearly* versus time.

$$f_i(t) = 2\mu t + f_0 \quad \text{(Hz)} \tag{8}$$

The slope of $f_i(t)$ is equal to 2μ and its intercept is f_0. For example, if the signal starts at time $t = t_1$ s with an initial frequency of f_1 Hz, and ends at time $t = t_2$ s with a final frequency of f_2 Hz, then the slope of the line in (8) will be

$$\text{SLOPE} = 2\mu = \frac{f_2 - f_1}{t_2 - t_1} \tag{9}$$

Note that if the signal starts at time $t = 0$ s, then the intercept $f_0 = f_1$ is equal to the starting frequency.

The frequency variation produced by the time-varying angle function is called *frequency modulation,* or simply FM. Finally, since the linear variation of the frequency can produce an audible sound similar to a bird chirp, linear-FM signals are also called *chirps.*

2.4.4 MATLAB Synthesis of Chirp Signals

The following MATLAB code will synthesize a linear-FM chirp:

```
fsamp = 8000;   %-Number of time samples per second
dt = 1/fsamp;
dur = 1.1;
tt = 0 : dt : dur;
f1 = 400;
psi = 2*pi*(100 + f1*tt + 500*tt.*tt);
xx = real( 7.7*exp(j*psi) );
soundsc( xx, fsamp );
```

(a) Determine the total duration of the synthesized signal in seconds, and also the length of the `tt` vector.

(b) In MATLAB signals can only be synthesized by evaluating the signal's defining formula at discrete instants of time. These are called *sample values* of the signal, or simply *samples.* For the chirp,

$$x(t_n) = A\cos(2\pi\mu t_n^2 + 2\pi f_0\, t_n + \varphi)$$

where t_n is the n^{th} time sample. In the MATLAB code above, identify the values of A, μ, f_0, and φ.

(c) Determine the range of frequencies (in hertz) that will be synthesized by the MATLAB script above, i.e., determine the minimum and maximum frequencies (in Hz) that will be heard. This will require that you relate the parameters μ, f_0, and φ to the minimum and maximum frequencies. Make a sketch by hand of the instantaneous (cyclic) frequency $f_i(t)$ versus time.

(d) Use `soundsc()` to listen to the signal in order to determine whether the signal's frequency content is increasing or decreasing. Notice that `soundsc()` needs to know two things: the vector containing the signal samples, and the rate at which the signal samples are to be played out. This rate should be the same as the rate at which the signal values were created, i.e., `fsamp` in the code above.
• More information is available from `help sound` and `help soundsc` in MATLAB.

2.5 Concatenating Signals via Addition

When we want to play several pieces of signals in succession, e.g., generating a C-Major scale with music notes or a telephone number with DTMF digits as studied in the previous lab, we must form a MATLAB vector to hold the signals. There are two strategies:

1. *Concatenation by appending:* if we want to play three signal vectors (`sa`, `sb` and `sc`) successively, then we can form a new longer signal vector as `ss = [ sa, sb, sc];` (assuming row vectors).

2. *Concatenation via addition into a long vector:* This technique relies on MATLAB's colon notation. If we pre-allocate a very long vector that will hold the entire concatenated signal, then we can add short signals to get concatenation. For example, if the three signal vectors (`sa`, `sb` and `sc`) have lengths (`La`, `Lb` and `Lc`), respectively, and `ss` is a very long vector initialized to zeros, then the following MATLAB code will produce the concatenation of the three signals.

```
ss(1:La) = ss(1:La) + sa;
ss(La+1:La+Lb) = ss(La+1:La+Lb) + sb;
Lab = La+Lb;
ss(Lab+1:Lab+Lc) = ss(Lab+1:Lab+Lc) + sc;
```

The drawback of the *Concatenation by appending* strategy, we studied in the previous lab, is that it cannot accommodate the common situation where we want to add together multiple signals with different durations that might overlap in time. On the other hand, the *Concatenation via addition into a long vector* method will deal naturally with overlapping signals.

2.6 Synthesizing and Concatenating Sinusoids

Whenever we take samples of a continuous-time formula, e.g., $x(t)$ at $t = t_n$, we are, in effect, implementing the ideal C-to-D converter. 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 a formula for the input signal, but we have already been doing this in previous labs.

(a) To begin, create a vector `xx` of samples of the sum of two sinusoidal signals: the first with $A_1 = 100$, $\omega_1 = 2\pi(800)$, and $\varphi_1 = 0.6\pi$; the second with a different frequency: $A_2 = 120$, $\omega_2 = 2\pi(2000)$, and $\varphi_2 = -0.1\pi$. Use a sampling rate of 8000 samples/sec, and compute a total number of samples equivalent to a time duration of 1.2 secs. You may find it helpful to recall that the MATLAB statement `tt=(0:0.01:3);` which creates a vector of numbers from 0 through 3 with increments of 0.01. Therefore, it is necessary to determine the time increment needed to obtain 8000 samples in one second.

```
function xs = shortSinus(amp, freq, pha, fs, dur)
% amp = amplitude
% freq = frequency in cycle per second
% pha = phase, time offset for the first peak
% fs = number of sample values per second
% dur = duration in sec
%
tt = 0 : 1/fs : dur; % time indices for all the values
xs = amp * cos( freq*2*pi*tt + pha );
end
```

This function `shortSinus` can be called once the argument values are specified. For example, the following MATLAB code will add two sinusoids—once you fill in the missing code at two places where you see `???`.

```
amps = [100, 120]
freqs = [800,2000]
phases = [0.6*pi, -0.1*pi]
fs = 8000;
tStart = [0.1, 0.1];
durs = [0.4,0.4];
maxTime = max(tStart+durs) + 0.1;  %-- Add time to show signal ending
durLengthEstimate = ceil(maxTime*fs);
tt = (0:durLengthEstimate)*(1/fs);  %-- be conservative (add one)
xx = 0*tt;   %--make a vector of zeros to hold the total signal
for kk = 1:length(amps)
  nStart = round(????)+1;  %-- add one to avoid zero index
  xNew = shortSinus(amps(kk), freqs(kk), phases(kk), fs, durs(kk));
  Lnew = length(xNew);
  nStop = ????;  %<========== Add code
  xx(nStart:nStop) = xx(nStart:nStop) + xNew;
end
plotspec(xx,fs,256); grid on
```

The starting index is an integer computed from the starting time (in secs) via the sampling rate (f_s). The relationship is $t = n/f_s$. In addition, if we define a subvector via the colon notation, e.g., `xx(n1:n2)`, then the length of that subvector is `n2-n1+1`. The fact that you must add one to get the length is confusing, but think of `xx(7:7)` which gives the single element `xx(7)`; its length is one.

Use `soundsc()` to play the resulting vector through the D-to-A converter of the your computer, assuming that the hardware can support the $f_s = 8000$ Hz rate. Listen to the output. Also, the *spectrogram* is an effective tool to verify the spectral content of a changing signal with multiple frequency components (see next Section and Fig. 1).

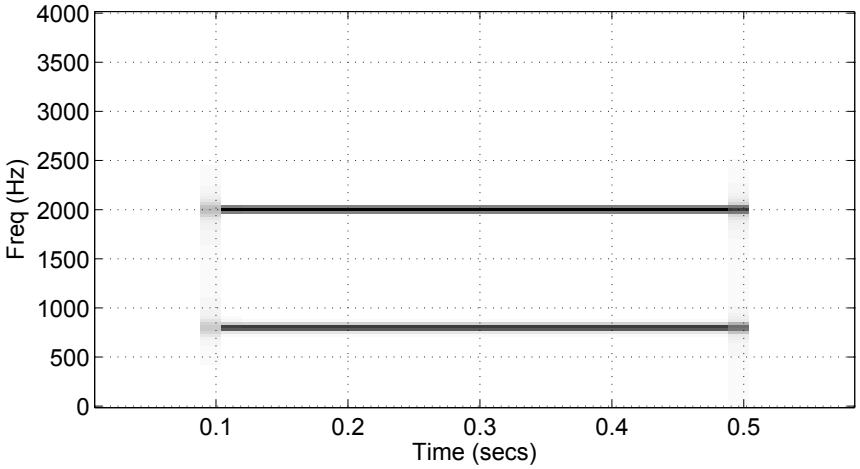

Figure 1: Spectrogram of two sinusoids from Section 2.6(a).

(b) ***Concatenate*** the two short sinusoidal signals defined in the previous part, and put a short duration of 0.1 seconds of silence in between, i.e., the second sinusoid will start after the first one ends Modify the MATLAB code above to accomplish this task.

(c) To verify that the concatenation operation was done correctly in part b, make the following plot:

```
tt = (1/fs)*(0:length(xx)-1);   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 120 at the correct times. Notice that the time vector `tt` was created to have exactly the same length as the signal vector `xx`.

2.7 Preliminary Topic: Spectrograms (More in Lab #3)

It is often useful to think of a signal in terms of its spectrum. A signal's spectrum is a representation of the frequencies present in the signal. For a constant frequency sinusoid, the spectrum consists of two spikes, one at $\omega = 2\pi f_0$, the other at $\omega = -2\pi f_0$. For a more complicated signal the spectrum may be very interesting, e.g., the case of FM, where the spectrum components are time-varying. One way to represent the time-varying spectrum of a signal is the *spectrogram* (see Chapter 3 in the text). A spectrogram is produced by estimating the frequency content in short sections of the signal. The magnitude of the spectrum over individual sections is plotted as intensity or color over a two-dimensional domain of frequency and time.

When unsure about a command, use `help`.

There are a few important things to know about spectrograms:

1. In MATLAB the function `spectrogram` will compute the spectrogram. Type `help spectrogram` to learn more about this function and its arguments. The `spectrogram` function used to be called `specgram`, and had slightly different defaults—the argument list had a different order, and the output format always defaulted to frequency on the vertical axis and time on the horizontal axis.

2. If you are working at home, you might not have a `spectrogram` 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* which can be downloaded from
   `http://dspfirst.gatech.edu/matlab/toolbox/`

   • Note: The argument list for `plotspec()` has a different order from `spectrogram` and `specgram`. In `plotspec()` the third argument is optional—it is the *section length* (default value is 256) which is often called the *window length*. In addition, `plotspec()` does not use color for the spectrogram; instead, darker shades of gray indicate larger values with black being the largest.

3. A common call to the MATLAB function is `spectrogram(xx,1024,[],[],fs,'yaxis')`. The second argument[1] is the *section length* (or window length) which could be varied to get different looking spectrograms. The spectrogram is able to "see" very closely spaced separate spectrum lines with a longer (window) section length,[2] e.g., 1024 or 2048.

In order to see a typical spectrogram, run the following code:

```
fs=8000; xx = cos(2000*pi*(0:1/fs:0.5)); spectrogram(xx,1024,[],[],fs,'yaxis'); colorbar
```

or, if you are using `plotspec(xx,fs)`:

```
fs=8000; xx = cos(2000*pi*(0:1/fs:0.5)); plotspec(xx,fs,1024); colorbar
```

Notice that the spectrogram image contains one horizontal line at the correct frequency of the sinusoid.

[1] If the second argument of `spectrogram` is made equal to the "empty matrix" then the default value used, which is the maximum of 256 and the signal length divided by 8.

[2] Usually the window (section) 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 FFT length is a power of 2.

3 In-Lab Exercise: Using M-file Functions and Structures

3.1 Debugging Skills

Testing and debugging code is a big part of any programming job. 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 semicolons). 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. Here is a link to a video about MATLAB's debugger:

`http://www.youtube.com/watch?v=Z4vFymKhNno`

Completion of Lab Results (on separate Report page)

3.2 M-file to Generate One Sinusoid

Write a function that will generate a **single** sinusoid, $x(t) = A\cos(2\pi ft + \varphi)$. The function should have the following input arguments: a sinusoid-structure with two fields for the frequency (f) in Hz, the complex amplitude $(X = Ae^{j\varphi})$, and then three other arguments: a duration argument (`dur`), followed by an argument for the starting time (`tstart`), and then a final argument which is the spacing between times, Δt. The function should return a structure having both of the fields of the input structure plus two new fields: the

vector of values of the sinusoidal signal (x) along with the corresponding vector of times (t) at which the sinusoid values are known. The spacing between times in the time-vector, Δt, is a constant, but make sure that it is small enough so that there are at least 32 time points per period of the sinusoid. Call this function `makeCosVals()`. *Hint: use* `goodcos()` *from the Pre-Lab part as a starting point.* Here is a suggested template that needs to be completed for the M-file:

```
function sigOut = makeCosVals(sigIn, dur, tstart, dt )
%
freq = sigIn.freq;
X = sigIn.complexAmp;
%
%...(Fill in several lines of code)...
%
tt = tstart: dt : ???;    %-- Create the vector of times
xx = A*cos(...???;        %-- Vectorize the cosine evaluation
sigOut.times = ???;       %-- Put vector of times into the output structure
sigOut.values = ???;      %-- Put values into the output structure
```

Plot the result from the following call to test your function.

```
mySig.freq = 3;    %-- (in hertz)
mySig.complexAmp = 4*exp(j*pi/6);
dur = 3;
start = -1;
dt = 1/(32*mySig.freq);
mySigWithVals = makeCosVals( mySig, dur, start, dt );
%- Plot the values in sigWithVals
```

Completion of Lab Results (on separate Report page)

3.3 Sinusoidal Synthesis with an M-file: Different Frequencies

Since we will generate many functions that are a *sum of sinusoids,* it will be convenient to have a MATLAB function for this operation. To be general, we should allow the frequency of each component (f_k) to be different. The following expressions are equivalent if we define the complex amplitude X_k as $X_k = A_k e^{j\varphi_k}$.

$$x(t) = \Re\left\{\sum_{k=1}^{N} X_k e^{j2\pi f_k t}\right\} = \Re\left\{\sum_{k=1}^{N} (A_k e^{j\varphi_k}) e^{j2\pi f_k t}\right\} \tag{10}$$

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \varphi_k) \tag{11}$$

3.3.1 Writing a Sum of Sinusoids M-file

Write an M-file called `addCosVals.m` that will synthesize a waveform in the form of (10) using X_k defined right above Eq. (10). The result is not a sinusoid unless all the frequencies are the same, so the output signal has to be represented by its values over some (finite) time interval.

Even though `for` loops are rather inefficient in MATLAB, *you must write the function with one outer loop in this lab.* The inner loop should be vectorized. The first few statements of the M-file are the comment lines—they should look like:

```
function    sigOut = addCosVals( cosIn, dur, tstart, dt )
%ADDCOSVALS  Synthesize a signal from sum of sinusoids
%               (do not assume all the frequencies are the same)
%
%  usage:   sigOut = addCosVals( cosIn, dur, tstart, dt )
%
%   cosIn = vector of structures; each one has the following fields:
%      cosIn.freq = frequency (in Hz), usually none should be negative
%      cosIn.complexAmp = COMPLEX amplitude of the cosine
%
%   dur = total time duration of all the cosines
%   start = starting time of all the cosines
%   dt = time increment for the time vector
% The output structure has only signal values because it is not necessarily a sinusoid
%      sigOut.values = vector of signal values at t = sigOut.times
%      sigOut.times  = vector of times, for the time axis
%
%    The sigOut.times vector should be generated with a small time increment that
%         creates 32 samples for the shortest period, i.e., use the period
%         corresponding to the highest frequency cosine in the input array of structures.
```

In order to verify that this M-file can synthesize ***harmonic*** sinusoids, try the following test:

```
ss(1).freq = 21;  ss(1).complexAmp = exp(j*pi/4);
ss(2).freq = 15;  ss(2).complexAmp = 2i;
ss(3).freq =  9;  ss(3).complexAmp = -4;
%
dur = 1;
tstart = -0.5;
dt = 1/(21*32);   %-- use the highest frequency to define delta_t
%
ssOut = addCosVals( ss, dur, tstart, dt );
%
plot( ssOut.????, ssOut.???? )
%
```

Use MATLAB to make a plot of `ssOut`. Notice that the waveform is periodic. Measure its period and state how the period is related to the fundamental frequency which is 3 Hz in this case.

Completion of Lab Results (on separate Report page)

3.3.2 Adding Short Sinusoid to a Long Signal Vector

For music synthesis, we need a way to add a short-duration sinusoid to an existing long vector. For example, suppose we have a MATLAB vector `xLong` which contains 100 elements, and we generate a sinusoid `xSinus` that contains 23 elements. If we want to add `xSinus` to `xLong` we cannot do `xSinus+xLong` because the dimensions do not match and MATLAB will report an error. Instead, we must find a 23-element subvector of `xLong`, and add `xSinus` to that subvector, e.g., `xSinus+xLong(31:53)`.

In music we must add the sinusoid at a location corresponding to a specified starting time, so we must know how to calculate the starting index of the subvector from a time (in secs). This time-to-index correspondence could be calculated, or it could be contained in a time vector that would accompany the signal vector. In order to calculate the index, we use the sampling relationship ($t_n = n/f_s$) and solve for the index n. Since the right hand side might not be an integer, in MATLAB we must round to the nearest integer (use

$f_s = 4000$ Hz in the following).

$$n_{\text{START}} = (f_s)(t_{\text{START}})$$

Generate two sinusoids both with zero phase and amplitude one. The first sinusoid should start at $t = 0.6$ s, have a frequency of 1200 Hz and a duration of 0.5 s; the second, a frequency of 750 Hz and a duration of 1.6 seconds starting at $t = 0.2$ s. Use a modified form of the MATLAB code from the pre-Lab. For verification, make a spectrogram (section length = 256) of the sum signal so that you can point out features that correspond to the parameters of the sinusoids. Also, be prepared to explain the modifications you made to the MATLAB code.

Completion of Lab Results (on separate Report page)

3.4 MATLAB Structure for Beat Signals

A beat signal is defined by five parameters $\{ B, f_c, f_\Delta, \varphi_c, \varphi_\Delta \}$ as shown in Section 2.3 so we can represent it with a MATLAB structure that has seven fields (by including start and end times), as shown in the following template:

```
sigBeat.Amp = 10;    %-- B in Equation (3)
sigBeat.fc = 480;    %-- center frequency in Eq. (3)
sigBeat.phic = 0;    %-- phase of 2nd sinusoid in Eq. (3)
sigBeat.fDelt = 20;  %-- modulating frequency in Eq. (3)
sigBeat.phiDelt = -2*pi/3;  %-- phase of 1st sinusoid Eq.~(\ref{Labeq:beatSigSum})
sigBeat.t1 = 1.1;    %-- starting time
sigBeat.t2 = 5.2;    %-- ending time
%
%----- extra fields for the parameters in Equation (4)
%
sigBeat.f1     %-- frequencies in Equation (4)
sigBeat.f2     %--
sigBeat.X1  %-- complex amps for sinusoids in Equation (4)
sigBeat.X2  %--     derived from A's and phi's
%
sigBeat.values %-- vector of signal values
sigBeat.times  %-- vector of corresponding times
```

(a) Write a MATLAB function that will add fields to a `sigBeatIn` structure. Follow the template below:

```
function sigBeatSum = sum2BeatStruct( sigBeatIn )
%
%--- Assume the five basic fields are present, plus the starting and ending times
%--- Add the four fields for the parameters in Equation (4)
%
%  sigBeatSum.f1, sigBeatSum.f2, sigBeatSum.X1, sigBeatSum.X2
```

(b) Produce a beat signal with two frequency components: one at 720 Hz and the other at 750 Hz. Use a longer duration than the default to hear the *beat frequency* sound. Use the feature discussed in Section 2.7 to generate a spectrogram plot of the beat signal here. Demonstrate the plot and sound to your lab instructor or TA.

3.4.1 Beat Note Spectrograms

Beat notes have a simple time-frequency characteristic in a spectrogram. Even though a beat note signal may be viewed as a single frequency signal whose amplitude envelope varies with time, *the spectrum or*

*spectrogram requires an **additive combination*** which turns out to be the sum of two sinusoids with different constant frequencies.

(a) Use the MATLAB function(s) written in Section 3.4 to create a beat signal defined via: $b(t) = 50\cos(2\pi(30)t + \pi/4)\cos(2\pi(800)t)$, starting at $t = 0$ with a duration of 4.04 s. Use a sampling rate of $f_s = 8000$ samples/sec to produce the signal in MATLAB. Use `testingBeat` as the name of the MATLAB structure for the signal. Plot a very short time section to show the amplitude modulation.

(b) Derive (mathematically) the spectrum of the signal defined in part (a). Make a sketch (by hand) of the spectrum with the correct frequencies and complex amplitudes.

(c) Plot the *two-sided spectrogram* of using a (window) section length of 512 using the commands[3]:

```
plotspec(testingBeat.values+j*1e-12,fs,512); grid on, shg
```

Comment on what you see. Can you see two spectral lines, i.e., horizontal lines at the correct frequencies in the spectrum found in the previous part? If necessary, use the zoom tool (in the MATLAB figure window).

Completion of Lab Results (on separate Report page)

3.5 Function for a Chirp

Use the code provided in the Pre-Lab section as a starting point in order to write a MATLAB function that will synthesize a "chirp" signal according to the template shown below. This will require that you determine the chirp parameters μ, f_0, and φ from the parameters passed in the LFM signal structure. Complete the function by filling in code where you see `???`.

```
function   sigOut = makeLFMvals( sigLFM, dt )
% MAKELFMVALS      generate a linear-FM chirp signal
%
% usage:   sigOut = makeLFMvals( sigLFM, dt )
%  sigLFM.f1 = starting frequency (in Hz) at t = sigLFM.t1
%  sigLFM.t1 = starting time (in secs)
%  sigLFM.t2 = ending time
%  sigLFM.slope = slope of the linear-FM (in Hz per sec)
%  sigLFM.complexAmp = defines the amplitude and phase of the FM signal
%  dt = time increment for the time vector, typically 1/fs (sampling frequency)
%
%  sigOut.values = (vector of) samples of the chirp signal
%  sigOut.times  = vector of time instants from t=t1 to t=t2
%
if( nargin < 2 )   %-- Allow optional input argument for dt
   dt = 1/8000;  %-- 8000 samples/sec
end
%--------NOTE: use the slope to determine mu needed in psi(t)
%--------  use f1, t1 and the slope to determine f0 needed in psi(t)
tt = ???
mu = ???
f0 = ???
psi = 2*pi*( f0*tt + mu*tt.*tt);
xx = real( ??? * exp(j*psi) );
sigOut.times = ???
sigOut.values = ???
```

[3] Use `plotspec` instead of `specgram` in order to get a linear amplitude scale rather than logarithmic. Also, use the tiny imaginary part to get the negative frequency region.

Testing: Plot the result from the following call to test your function.

```
myLFMsig.f1 = 200;
myLFMsig.t1 = 0;  myLFMsig.t2 = 1.5;
myLFMsig.slope = 1800;
myLFMsig.complexAmp = 10*exp(j*0.3*pi);
dt = 1/8000;   % 8000 samples per sec is the sample rate
outLFMsig = makeLFMvals(myLFMsig,dt);\\[1.5ex]
%- Plot the values in outLFMsig
%- Make a spectrogram for outLFMsig to see the linear frequency change
```

The test case above generates a chirp sound whose frequency starts low and chirps up. From the duration (in secs.) and the sampling rate of $f_s = 8000$ samples/s, the size of the output signal vector can be determined. Use MATLAB's `size` command to check that `outLFMsig.values` has the expected size.

Listen to the chirp using the `soundsc` function, and make a two-sided spectrogram of the signal, i.e., including the negative frequencies. You can use MATLAB's cell-mode feature to show the spectrogram within a web page. Zoom in on the beginning and end of the plot to verify that the frequencies have the values expected, and that the starting frequency is lower at the beginning than at the end of the chirp.

Completion of Lab Results (on separate Report page)

Lab #2
ECE-2026 Spring-2025
LAB COMPLETION REPORT

Name: _____ Date of Lab: _____

Part 3.1 Show a few of the debugging features.

Part 3.2 Write the MATLAB code for generating a single sinusoid with varying amplitude, frequency and phase. Show the signal, spectrogram and sound (Note: if your assigned frequency is too low, then it may be not audible, why?).

Part 3.3.1 Show that your `addCosVals.m` function is correct by running the test in Section 3.3.1 and plotting the result. Measure the period of signal in the structure `ssOut`, and explain its relationship to the fundamental frequency.

Part 3.3.2: Write the MATLAB code for calculating `nStart` and `nStop` needed in the code.

`nStart =`

`nStop =`

Part 3.4 Write the MATLAB code for generating the beat signal. Show the signal, spectrogram and sound

Part 3.5 Run the testing code to demonstrate your chirp signal and its spectrogram.

$\mu =$

$f_0 =$

$\varphi =$