GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2026 Spring 2015
Lab #4: Synthesis of AM and FM Signals

Date: 2–5 Feb. 2015

Pre-Lab: 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.***

Verification: The Exercise 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**. One of the laboratory instructors must verify the appropriate steps by signing on the **Instructor Verification** line. When you have completed a step that requires verification, simply raise your hand and demonstrate the step to the TA or instructor. Turn in the completed verification sheet to your TA when you leave the lab.

Lab Homework Questions: The Lab-Homework Sheet has a few lab related questions that can be answered at your own pace. The completed Lab-HW sheet is due at the beginning of the next lab.

Forgeries and plagiarism are a violation of the honor code and will be referred to the Dean of Students for disciplinary action. You are allowed to discuss lab exercises with other students, 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 Overview

Please read through the information below prior to attending your lab.

Objective: The objective of this lab is to learn to how AM and FM signals can be synthesized. The resulting signal can be analyzed to show its time-frequency behavior by using the *spectrogram.*

This lab studies sound synthesis via short-duration sinusoidal waveforms of the form

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

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 Pre-Lab

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

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

In this lab, we will extend our treatment of sinusoidal waveforms to more complicated signals composed of sums of sinusoidal signals, or sinusoids with changing frequency, i.e., frequency-modulated sinusoids.

2.1 Amplitude Modulation

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:

$$x(t) = \sum_{k=1}^{N} A_k\cos(2\pi f_k t + \varphi_k) = \Re\left\{\sum_{k=1}^{N}\left(A_k e^{j\varphi_k}\right)e^{j2\pi f_k t}\right\} \tag{3}$$

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 pick two or three frequencies that are very close together, see Chapter 3.

2.2 Frequency Modulated Signals

In this lab, we will examine signals whose frequency varies as a function of time. Recall that in a constant-frequency sinusoid (2) 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 (2), 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):

$$x(t) = A\cos(\psi(t)) = \Re\{Ae^{j\psi(t)}\} \tag{4}$$

where $\psi(t)$ is the angle function. The time derivative of the angle function $\psi(t)$ in (4) 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:*

$$f_i(t) = \frac{1}{2\pi}\frac{d}{dt}\psi(t) \qquad \text{(Hz)} \tag{5}$$

2.3 Chirp, or 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 (4) as the following quadratic function:

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

Using (5), we obtain an instantaneous *cyclic* frequency that changes *linearly* versus time.

$$f_i(t) = 2\mu t + f_0 \quad \text{(Hz)} \tag{6}$$

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 (6) will be

$$\text{SLOPE} = 2\mu = \frac{f_2 - f_1}{t_2 - t_1} \tag{7}$$

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 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).

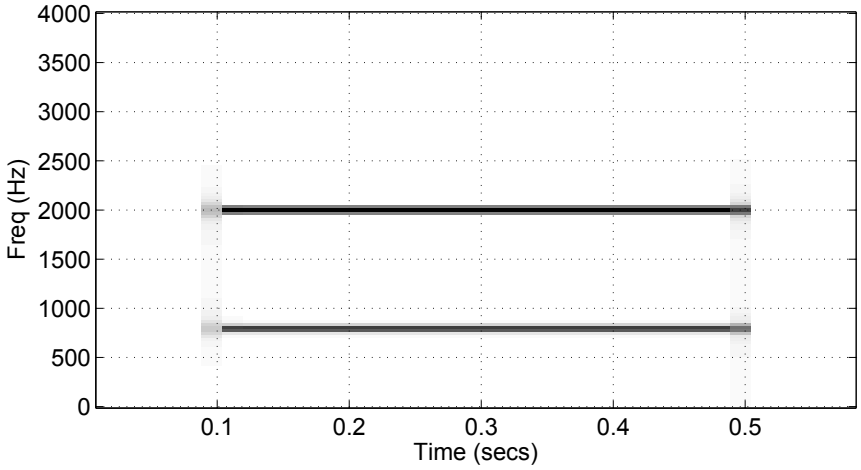

Figure 1: Spectrogram of two sinusoids from Section 2.6(a).

(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 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`

2.8 Advanced Topic: Spectrograms

It is often useful to think of a signal in terms of its spectrum as discussed in Chapter 3. 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 some additional important things to know about spectrograms following last week's preliminary exercises:

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. 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.

3. **(Negative) Frequency Range:** Normally the spectrogram image contains only positive frequencies. However, you can produce a spectrogram image containing negative frequencies ***if you use the function*** *`plotspec`* ***and if you make the input signal complex.*** Even if your signal is real, you can add a very tiny imaginary part, e.g., `xx = xx + j*1e-14`, to make it seem to be complex-valued.
   Warning: This trick works nicely with the *SP-First* function called `plotspec`. However, when used with `spectrogram` the result does not have the negative frequency region in the proper location.

4. **Section Length of Spectral Analysis Windows:** A spectrogram is formed by taking successive short sections of a signal and performing an FFT analysis of each of those sections to get the spectrum. Since this is done repeatedly, the result is the spectrum versus time, where time is the location of the short sections. For a specific example, assume that the section length is 100, and the signal is a MATLAB vector `xx`. Then the first short section will be `xx(1:100)`. The sections are usually overlapped and the default in `plotspec` is 50% overlap, so the second short section is `xx(51:150)`, the third `xx(101:200)`, and so on.

   The spectrogram image is, in effect, the spectrum versus time, so we need a reference time for each short section. In `plotspec` this reference time is the "midpoint" of the section. For the length-100 section, the reference index is 50, which is then converted to a time (in secs) by using the sampling rate (f_s). When the spectrogram is displayed as an image, these reference times are used along the horizontal axis.

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

```
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.
To obtain a spectrogram with negative frequencies, try the following

```
xx = cos(2000*pi*(0:1/fs:0.5)); plotspec(xx+j*1e-9,fs,1024); colorbar
```

[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

For the instructor verification, you will have to demonstrate that you understand concepts in a given subsection by answering questions from your lab instructor (or TA).

3.1 Using the MATLAB Debugger

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.
5. Another useful activity is to set a breakpoint in the first line of the `coscos` function, and then single-step one line at a time through the function.

```
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;
```

Use the feature discussed in Section 2.7 to show your debugging skills to your lab instructor or TA.

Instructor Verification (separate page)

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.

Instructor Verification (separate page)

3.3 Spectrogram: Section Length and Negative Frequency

In this part, you must display the spectrogram of the signal synthesized in the previous section with different parameters. Remember that the spectrogram displays an image that shows the *frequency* content of the synthesized *time* signal. Its horizontal axis is time and its vertical axis is frequency. Use the function `specgram(xx,512,fs)`. The second argument[3] is the *window length* which could be varied to get different looking spectrograms. The spectrogram is able to "see" the separate spectrum lines with a longer window length, e.g., 1024 or 2048.[4]

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

```
fs=8000; tt=0:1/fs:0.5; xx = cos(4000*pi*tt); spectrogram(xx,1024,[],[],fs,'yaxis'); colorbar
```

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

```
tt=0:1/fs:0.5; yy=xx+cos(1600*pi*tt); plotspec(yy,fs,1024); colorbar
```

Notice that the spectrogram image now contains two horizontal line at the correct frequencies of the sinusoids similar to what was displayed in Fig. 1.
To obtain a spectrogram with negative frequencies, try the following

```
plotspec(yy+j*1e-9,fs,1024); colorbar
```

To obtain a spectrogram with a different section length, try the following

```
plotspec(yy+j*1e-9,fs,128); colorbar
```

For this verification, show the spectrogram features with negative frequencies and different section lengths to your lab instructors. You can also use the spectrogram of the DTMF digit **7** obtained in the previous lab.

Instructor Verification (separate page)

3.4 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 `???`.

[3] If the second argument is made equal to the "empty matrix" then its default value of 256 is used.

[4] Usually the window length is chosen to be a power of two, because a special algorithm called the FFT is used in the computation. The fastest FFT programs are those where the signal length is a power of 2.

```
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 = ???
```

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);
%- 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 high and chirps down. 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 higher at the beginning than at the end of the chirp.

Instructor Verification (separate page)

4 Lab Homework: Chirp Signals, Arpeggio, and Their Spectrograms

For the lab homework, you will create an arpeggio and match FM signals with given spectrograms and discuss the connection between the *time-domain* definition of the signal and its *frequency-domain* content.

4.1 Synthesize an Arpeggio and a Chord

As we studied in the previous lab, a piano keyboard shown in Figure 2 is divided into octaves—the notes in one octave being twice the frequency of the notes in the next lower octave.

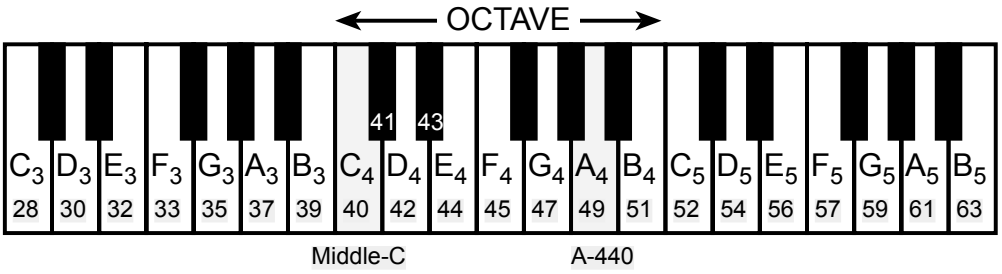

Figure 2: Layout of a piano keyboard. Key numbers are shaded. The notation C_4 means the C-key in the fourth octave, which is middle-C. The white keys in each octave are named A through G.

Use the technique of adding short sinusoids studied in Section 3.2, and create a signal that plays seven notes in succession: F_4, A_4, C_5, F_5, C_5, A_4, and F_4, starting each note at every 0.25 s, i.e., a new note should start with the starting times set at $\{0,\ 0.25,\ 0.5,\ 0.75,\ 1.0,\ 1.25,\ 1.5\}$ seconds. The total duration of the musical passage should be 2.5 s, i.e., ending the passage with silence. The amplitudes of all signals should be equal; the phases do not matter. Synthesize the signal with a sampling rate at $f_s = 4000$ Hz. The first three notes should have a duration of 0.2 s, but the remaining four notes should be held until the end before the ending silence segment of 0.1 s kicks in, so their individual note durations will vary. 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: starting and ending times, as well as frequencies.

4.2 Spectrogram of an FM Signals: Exponential and Sinusoidal Instantaneous Frequencies

Define an FM signal whose instantaneous frequency is

$$\omega_i(t) = \frac{d}{dt}\psi(t) = 2\pi\left(f_c + \gamma e^{\beta t}\right) \quad \text{radians/sec} \tag{8}$$

where f_c is the center frequency, and the parameters γ and β control the frequency modulation.

(a) Determine the mathematical formula for an FM signal that has the instantaneous frequency in (8).

(b) Write a MATLAB function to create exponential FM signals of the form found in the previous part. The MATLAB function should be called `makeFMexpVals( sigFMexp, dt )`. Use a structure for these signals which has the following parameters:

```
sigFMexp.Amp;   %-- Amplitude
sigFMexp.fc     %-- center frequency
sigFMexp.beta   %-- FM parameter
sigFMexp.gamma  %-- FM parameter
sigFMexp.t1     %-- starting time
sigFMexp.t2     %-- ending time
```

(c) Create an exponential-FM signal with $f_c = 500\,\text{Hz}$, $\gamma = 3$, $\beta = 2.05$, amplitude $A = 7.5$, with a signal duration equal to 3.04 s starting at $t = 0$. Use a sampling rate of 4000 samples/s to define `dt`.

An FM signal with a sinusoidal instantaneous frequency can also be found in a similar way as follows:

$$x(t) = A\cos(2\pi f_c t + \alpha\cos(2\pi\beta t + \gamma)) \tag{9}$$

where f_c centers the frequency plot, and α, β and γ control the frequency modulation.

4.3 Matching Unknown Spectrograms

Now given the six spectrograms in Fig. 3, Do the following and discuss your work for all six cases.

1. For each case **(A–F)**, define a time signal $x_i(t)$ whose spectrogram will match the given spectrogram. This signal definition should be a simple mathematical formula.
   Note: you might have to iterate with the following two steps to get a very good approximation.

2. Then write a MATLAB function or use one of the previous MATLAB functions for beat signals, linear-FM, exponential-FM, or sinusoidal-FM to generate samples of each signal over the time interval $[0, 5.04]$ secs. The sampling interval should be $\Delta t = 1/4000\,\text{s}$.

3. Make a *two-sided* spectrogram of each signal with a window length of 256 to confirm your answers.

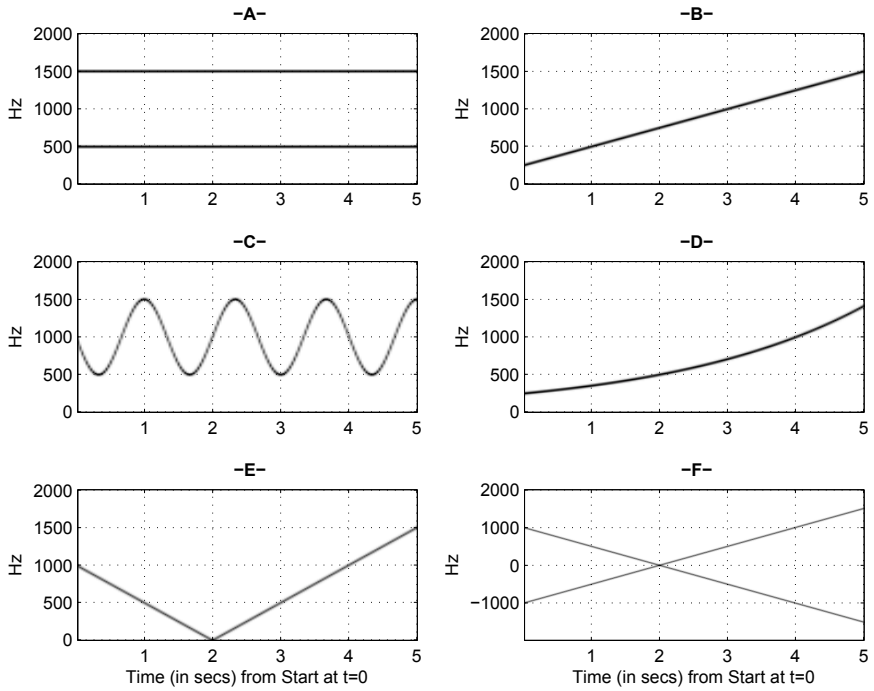

Figure 3: Six spectrograms for six unknown signals. All spectrograms were computed with a window length of 256, and the signals were sampled at a rate of 4000 samples/s. Case **A** has a pair of constant frequencies. Case **B** has a linear instantaneous frequency. Case **C** has a sinusoidal instantaneous frequency. Case **D** has an exponential instantaneous frequency that doubles every 2 seconds, which can be expressed as $\alpha 2^{0.5t}$. Case **E** has a linear instantaneous frequency, with a negative slope, that drops quickly and causes aliasing (more in Lab 5). Case **F** is simply a two-sided version of the corresponding spectrogram in Case **E**.

Lab #4
ECE-2026 Spring-2015
INSTRUCTOR VERIFICATION SHEET

Turn this page in to your lab grading TA before the end of your scheduled Lab time.

Name: _____ gtLoginUserName: _____ Date: _____

Part 3.1: Show a few of the debugging features.

Verified:_____ Date/Time:_____

Part 3.2: Write the MATLAB code for calculating `nStart` and `nStop` needed in the code from Sect. 2.6(a).

```
nStart =
nStop =
```

Verified:_____ Date/Time:_____

Part 3.3 Demonstrate the section length and negative frequency features in the spectrogram.

Verified:_____ Date/Time:_____

Part 3.4 Run the testing code to demonstrate your chirp signal and its spectrogram.

μ =

f_0 =

φ =

Verified:_____ Date/Time:_____

Lab #4
ECE-2026 Spring-2015
LAB HOMEWORK QUESTION

Turn this page in to your lab grading TA at the very beginning of your next scheduled Lab time.

Name: _____ gtLoginUserName: _____ Date: _____

Part 4.1: Using the techniques in Section 3.2 to synthesize an arpeggio of seven notes in succession: F_4, A_4, C_5, F_5, C_5, A_4, and F_4. Turn in your code and plot the spectrogram below.

Part 4.2(a): $x(t) = cos(\psi(t))$ =

Part 4.2(b): turn in your MATLAB code.

Part 4.2(c): plot the spectrogram below.

Part 4.3: Use the first five spectrograms in Fig. 3 to find the five matching signals that generated the plots.

1. Signal A: $x_a(t)$ =
2. Signal B: $x_b(t)$ =
3. Signal C: $x_c(t)$ =
4. Signal D: $x_d(t)$ =
5. Signal E: $x_e(t)$ =