GEORGIA INSTITUTE OF TECHNOLOGY
SCHOOL of ELECTRICAL and COMPUTER ENGINEERING

ECE 2025 Spring 2011
Lab #3: AM and FM Sinusoidal Signals

Date: 7–10 Feb. 2011

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 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.*

It is only necessary to turn in Section 4 as the lab report for this lab. More information on the lab report format can be found on **`t-square`** under the **INFO** link. Please **label** the axes of your plots and include a title for every plot. In order to reduce "orphan" plots, include each plot as a figure *embedded* within your report. For more information on how to include figures and plots from MATLAB in your report file, consult the **INFO** link on **`t-square`**, or ask your TA for details.

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.

Due Date: The lab report will be **due during the period 14–17 Feb. at the start of your lab.**

---

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) and amplitude modulation (AM) are widely used in communication systems such as radio and television. In addition, they can be used to create interesting sounds that mimic musical instruments. 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:

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

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:

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

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

2.1.1 Beat Control GUI

beatcon.m

To assist you in your experiments with beat notes and AM signals, the MATLAB GUI tool called **`beatcon`** has been created. This *user interface controller* will exhibit the basic signal shapes for beat signals and play the signals. A small control panel will appear on the screen with *buttons* and *sliders* that vary the different parameters for the beat signals. It can also call a user-written function named `beat.m`. Experiment with the **`beatcon`** control panel and use it to produce a beat signal with two frequency components: one at 690 Hz and the other at 700 Hz. Use a longer duration than the default to hear the *beat frequency* sound.

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 (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:

$$x(t) = A\cos(\psi(t)) = \Re\{A e^{j\,\psi(t)}\} \tag{3}$$

where $\psi(t)$ is the angle function. The time derivative of the angle function $\psi(t)$ in (3) 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:*

$$f_i(t) = \frac{1}{2\pi}\frac{d}{dt}\psi(t) \qquad \text{(Hz)} \tag{4}$$

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 value to an ending one. The formula for such a signal can be defined by creating a complex exponential signal with a quadratic angle function. Mathematically, we define $\psi(t)$ in (3) as

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

The derivative of $\psi(t)$ yields an instantaneous (cyclic) frequency (4) that changes *linearly* versus time.

$$f_i(t) = 2\mu t + f_0 \quad \text{(hertz)} \tag{5}$$

The slope of $f_i(t)$ is equal to 2μ and its intercept is f_0. The frequency variation produced by the time-varying angle function is called *frequency modulation,* so these signals are called FM signals. Finally, since the linear variation of the frequency can produce an audible sound similar to a siren or a bird chirp, linear-FM signals are also called *chirps.*

If the signal starts at time $t = 1$ s with a frequency of f_1 Hz, and ends at time $t = 2$ s with a frequency of f_2 Hz, then the slope of the line in (5) will be

$$\text{SLOPE} = 2\mu = \frac{f_2 - f_1}{t_2 - t_1} \tag{6}$$

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

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 *samples* of the signal. For the chirp we use the following:

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

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 (`fsamp` in the code above).
For more information do `help sound` and `help soundsc` in MATLAB.

2.5 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 alink to a video about MATLAB's debugger:

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

3 Warm-up

The instructor verification sheet may be found at the end of this lab. The "Beat Control GUI" is part of the *SP-First* toolbox, and it should be on the MATLAB path already installed on the computers in the ECE Labs.

3.1 Use 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.

```
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;
```

3.2 Beat Control GUI

Use the **beatcon** control panel to produce a beat signal with two frequency components: one at 700 Hz and the other at 710 Hz. Use a longer duration than the default to hear the "beating" sound. Demonstrate the plot and sound to your TA.

Instructor Verification (separate page)

3.3 Advanced Topic: Spectrograms

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 as in (1) 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 on a two-dimensional plot versus frequency and time.

Sounds & Spectro-grams

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. Th `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 any `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://users.ece.gatech.edu/mcclella/SPFirst/Updates/SPFirstMATLAB.html`

   • 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 commonly 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. Spectrograms are numerical calculations and provide only an estimate of the time-varying frequency content of a signal. There are theoretical limits on how well they can actually represent the *true* frequency content of a signal. Another lab on the *SP-First* CD-ROM treats this problem by describing how to use the spectrogram to extract the frequencies of piano notes.

4. A common call to the 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" the separate spectrum lines with a longer (window) section length,[2] e.g., 1024 or 2048.

5. **(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.

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. For 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
```

You will also be able to show the spectrogram of the chirp produced in the next part, Section 3.4.

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 following template. This will require that you relate the chirp parameters μ, f_0, and φ to the starting and ending frequencies. Fill in code where you see `???`.

[1] If the second argument of `specgram` 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.

```
function   sigOut = makeLFMvals( sigLFM, dt )
%MAKELFMVALS      generate a linear-FM chirp signal
%
%  usage:   sigOut = makeLFMvals( sigLFM, dt )
%     sigLFM.f1 = starting frequency (in Hz)
%     sigLFM.f2 = ending frequency
%     sigLFM.t1 = starting time (in secs)
%     sigLFM.t2 = ending time
%     sigLFM.complexAmp = defined by amplitude and phase of the FM signal
%        dt = time increment for the times vector, e.g., 10000 samples/sec is dt=1e-4
%
%     sigOut.values = (vector of) samples of the chirp signal
%     sigOut.times  = vector of time instants from t=t1 to t=t2
%
if( nargin < 4 )   %-- Allow optional input argument for dt
   dt = 1/8000;  %-- 8000 samples/sec
end
%--------NOTE: use (f1,t1) and (f2,t2) to determine mu and f0 needed in psi(t)
%-------   from the slope and intercept of the desired linear frequency change.
tt = ???
mu = ???
f0 = ???
psi = 2*pi*( f0*tt + mu*tt.*tt);
xx = real( ??? * exp(j*psi) );
sigOut.times = ???
sigOut.values = ???
```

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

```
myLFMsig.f1 = 3200;  myLFMsig.t1 = 1;
myLFMsig.f2 =  200;  myLFMsig.t2 = 2.2;
myLFMsig.complexAmp = 10*exp(j*0.3*pi);
dt = 1/8000;
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 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. Recall that you can use MATLAB's cell-mode feature to make this plot within a web page. Zoom in on the beginning and end of the plot to verify that the frequency is higher at the beginning than at the end of the chirp.

Instructor Verification (separate page)

4 Lab: Chirps and Beats

For the lab exercise and lab report, you will synthesize some AM and FM signals. In order to verify that these signals have the correct frequency content, you will use the spectrogram. Your lab report should discuss the connection between the *time-domain* definition of the signal and its *frequency-domain* content.

4.1 Beat Notes

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.,

$$x(t) = B\cos(2\pi f_\Delta t + \varphi_\Delta)\cos(2\pi f_c t + \varphi_c) \tag{7}$$

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:

$$x(t) = A_1\cos(2\pi f_1 t + \varphi_1) + A_2\cos(2\pi f_2 t + \varphi_2) \tag{8}$$

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

4.1.1 MATLAB Structure for Beat Signals

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

```
sigBeat.Amp = 10;    %-- B in equation (7)
sigBeat.fc = 440;    %-- center frequency in (7)
sigBeat.phic = 0;    %-- phase of 2nd sinusoid in (7)
sigBeat.fDelt = 30;  %-- modulating frequency in (7)
sigBeat.phiDelt = -2*pi/3;  %-- phase of 1st sinusoid (7)
sigBeat.t1 = 1.1;    %-- starting time
sigBeat.t2 = 5.2;    %-- ending time
%
%----- extra fields for the parameters in equation (8)
%
sigBeat.f1     %-- frequencies in equation (8)
sigBeat.f2     %--
sigBeat.camp1  %-- complex amps frequencies in equation (8)
sigBeat.camp2  %--     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 `sigBeat` structure. Follow the template below:

```
function sigOut = updateBeatToSumForm( sigBeatIn )
%
%--- Assume the five basic fields are present, plus the starting and ending times
%--- Add the four fields for the parameters in equation (8)
%
%  sigBeat.f1,  sigBeat.f2,  sigBeat.camp1,  sigBeat.camp2
```

(b) Write a MATLAB function that will add signal-value fields to a `sigBeat` structure.

```
function  sigOut = makeBeatVals( sigBeatIn, dt )
%
%--- Assume the five basic fields are present, plus the starting and ending times
%---  and "dt" is the time-sampling interval for the time axis
%
%--- Add the following two fields for the vector of signal values:
%          sigBeat.times,  sigBeat.values
```

4.1.2 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 varies with time, *the spectrum 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 4.1.1 to create and plot a beat signal defined via: $b(t) = 10\cos(2\pi(40)t - 2\pi/3)\cos(2\pi(100)t)$, starting at $t = 0$ with a duration of 3.04 s. Use a sampling rate of $f_s = 800$ samples/sec to produce the signal in MATLAB. Use `myBeat` as the name of the MATLAB structure for the signal. Observe 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 $b(t)$ using a (window) section length of 64 using the commands[3]:

```
plotspec(myBeat.values+j*1e-12,fs,64); 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).

4.2 Spectrogram of an FM Signal: Sinusoidal Modulation

Define an FM signal whose instantaneous frequency is

$$\omega_i(t) = 2\pi f_c + 2\pi\alpha\cos(2\pi\beta t + \gamma) \quad \text{radians/sec} \tag{9}$$

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

(a) Determine the mathematical formula for an FM signal that has the instantaneous frequency in (9).

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

```
sigFMcos.Amp;   %-- Amplitude
sigFMcos.fc     %-- center frequency
sigFMcos.alpha  %-- FM parameters
sigFMcos.beta   %--
sigFMcos.gamma  %--
sigFMcos.t1     %-- starting time
sigFMcos.t2     %-- ending time
```

(c) Create a sinusoidal-FM signal with $f_c = 100$ Hz, $\alpha = 50$, $\beta = 1.5$, $\gamma = \pi/3$ and amplitude $A = 1$. Make the signal duration equal to 3.04 secs, starting at $t = 0$. Use a sampling rate of 800 samples/s to define `dt`.

(d) Create a spectrogram of this chirp signal, and use it to verify that you have the correct instantaneous frequency predicted by (9). .

[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.

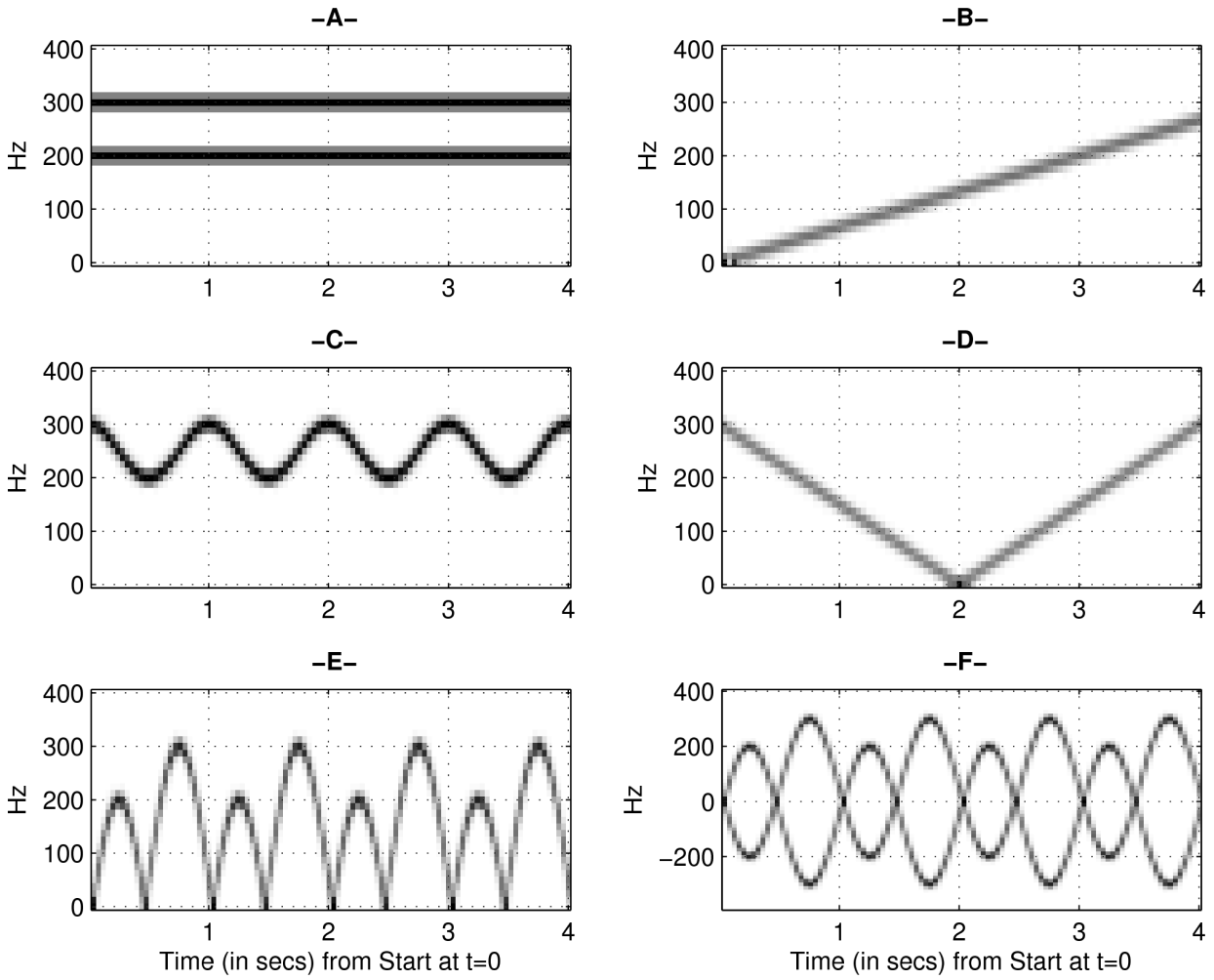


Figure 1: Six spectrograms for five unknown signals. All spectrograms were computed with a window length of 64, and the signals were sampled every 1/800 secs. Cases **E** and **F** are done for the same signal; the spectrogram for **F** is *two-sided* to show that there are always negative frequency components present.

4.3 Matching Unknown Spectrograms

Now you are given five spectrograms in Fig. 4.3, and you must synthesize signals that will match them.

1. For each case **(A–E)**, 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 good approximation.

2. Then use one of the MATLAB functions for beat signals, linear-FM or sinusoidal-FM to generate samples of the signal over the time interval $[0, 4.04]$ secs. The sampling interval should be 1/800 s.

3. Make a *two-sided* spectrogram of the signal with a window length of 64 to confirm your answers.

4. Explain and discuss your work for each of the five cases.

Lab #3
ECE-2025 Spring-2011
INSTRUCTOR VERIFICATION SHEET

Turn this page in to your TA before the end of your lab period.

Name: _____ Date of Lab: _____

Part 3.1 Show that you can use the debugger on the text file `coscos.m`:

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

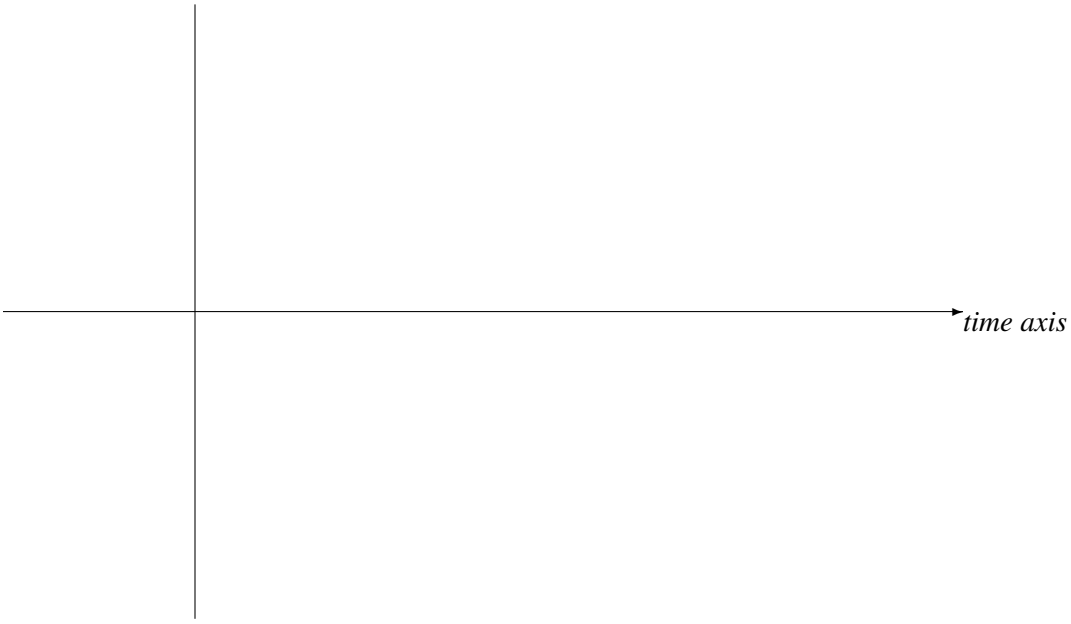
Verified:_____ Date/Time:_____

Part 3.2 Demonstrate usage of the Beat Control GUI.

Verified:_____ Date/Time:_____

Part 3.3 Demonstrate the ***two-sided*** spectrogram of a sinusoid and a chirp, i.e., include the negative frequencies by using the MATLAB function `plotspec`. Make a sketch of the resulting spectrogram below.

Verified:_____ Date/Time:_____



Part 3.4 Demonstrate the `makeLFMvals.m` function. In the space below, write how you would determine the length of output vector, `myLFMwithVals.values`.

Verified:_____ Date/Time:_____