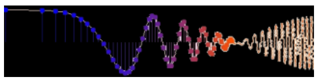

Signal Processing First

Lab 02: Introduction to Complex Exponentials – Multipath

Pre-Lab and Warm-Up: You should read at least the Pre-Lab and Warm-up sections of this lab assignment and go over all exercises in the Pre-Lab section before going to your assigned lab session.

1. Introduction

The goal of this laboratory is to gain familiarity with complex numbers and their use in representing sinusoidal signals such as $x(t) = A\cos(\omega t + \varphi)$ as complex exponentials $z(t) = Ae^{j\varphi}e^{j\omega t}$. The key is to use the appropriate complex amplitude together with the real part operator as follows:

$$x(t) = A\cos(\omega t + \varphi) = \Re e\{Ae^{j\varphi}e^{j\omega t}\}$$

2. Overview

Manipulating sinusoidal functions using complex exponentials turns trigonometric problems into simple arithmetic and algebra. In this lab, we first review the complex exponential signal and the phasor addition property needed for adding cosine waves. Then we will use MATLAB to make plots of phasor diagrams that show the vector addition needed when adding sinusoids.

2.1 Sinusoid Addition Using Complex Exponentials

Recall that sinusoids may be expressed as the real part of a complex exponential:

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

The *Phasor Addition Rule* shows how to add several sinusoids:

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_0 t + \phi_k) \tag{2}$$

assuming that each sinusoid in the sum has the *same* frequency, f_0. This sum is difficult to simplify using trigonometric identities, but it reduces to an algebraic sum of complex numbers when solved using complex exponentials. If we represent each sinusoid with its *complex amplitude*

$$X_k = A_k e^{j\phi_k} \tag{3}$$

Then the complex amplitude of the sum is

$$X_s = \sum_{k=1}^{N} X_k = A_s e^{j\phi_s} \tag{4}$$

Based on this complex number manipulation, the *Phasor Addition Rule* implies that the amplitude and phase of x(t) in equation (2) are A_s and φ_s, so

$$x(t) = A_s \cos(2\pi f_0 t + \phi_s) \tag{5}$$

We see that the sum signal x(t) in (2) and (5) is a single sinusoid that still has the same frequency, f_0, and it is periodic with period $T_0=1/f_0$

2.2 Harmonic Sinusoids

There is an important extension where x(t) is the sum of N cosine waves whose frequencies (f_k) are *different.* If we concentrate on the case where the (f_k) are all multiples of one basic frequency f_0, i.e.,

$$f_k = kf_0 \qquad \text{(HARMONIC FREQUENCIES)}$$

then the sum of N cosine waves given by (2) becomes

$$x_h(t) = \sum_{k=1}^{N} A_k \cos(2\pi k f_0 t + \phi_k) = \Re e\left\{\sum_{k=1}^{N} X_k e^{j2\pi k f_0 t}\right\} \tag{6}$$

This particular signal $x_h(t)$ has the property that it is also periodic with period $T_0 = 1/f_0$, because each of the cosines in the sum repeats with period T_0. The frequency f_0 is called the *fundamental frequency*, and T_0 is called the *fundamental period*. (Unlike the single frequency case, there is no phasor addition theorem here to combine the harmonic sinusoids.)

3. Pre-Lab

Do all the exercises in this section before attending the regularly scheduled lab section.

3.1 Functions

Functions are a special type of M-file that can accept inputs (matrices and vectors) and also return outputs. The keyword `function` must appear as the first word in the ASCII file that defines the function, and the first 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.5 in Appendix B for more discussion.

The following function has a few mistakes. Before looking at the correct one below, try to find these mistakes (there are at least three):

i) Area of a circle;

```
function [area]= area_circle(r)
area=pi*r^2;
fprintf('The radius of the circle area of %f to %f units.',r,a)
```

Try the following text on the command window;
>> area_circle(3)

ii)

```
 matlab mfile [xx,tt] = badcos(ff,dur)
 %BADCOS Function to generate a cosine
```

```
wave
% usage:
%      xx  =  badcos(ff,dur)
%      ff  =  desired  frequency  in  Hz
%     dur  =  duration  of  the  waveform  in  seconds

tt  =  0:1/(100*ff):dur;  %--  gives  100  samples  per
period badcos = cos(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 = cos(2*pi*ff*tt);
```

Notice the word "function" in the first line. Also, "freeq" has not been defined before being used. Finally, the function has "xx" as an output and hence "xx" should appear in the left-hand side of at least one assignment line within the function body. The function name is *not* used to hold values produced in the function.

iii) a general routine that will compute the value of some arbitrary polynomial at an arbitrary set of equally-spaced x values. This turns out to be something that is best done using loops (or at least, I couldn't find a clever way to avoid loops – this is good because it at least gives an example of how to do loops).

```
function [y,x] = poly_equi(power,coeffs,xmin,xmax,del_x)
%
% USAGE: [y,x] = poly_equi(power,coeffs,xmin,xmax,del_x);
%
% Inputs: power = an integer specifying the highest power of the
% polynomial
%            coeffs = a row vector of the polynomial's coefficients
%            xmin = the desired minimum x value for evaluation
%            xmax = the desired maximum x value for evaluation
%            del_x = the desired spacing between the x values
%
% Output: x = a row vector of the x values
%         y = a row vector of the y values

x = xmin:del_x:xmax;  % this uses MATLAB's colon operator to generate
                      % equally spaced points at a spacing of del_x

for n=0:power;


% Loop through the powers in the polynomial
% and compute a row vector for each of the terms in the polynomial
% and put that row vectro into each row of matrix X

 X(n+1,:)=coeffs(n+1)*x.^n;   % note use of "n+1" for indexing-matlab
                              %can't index using "0"
```

```
end

%  You now have a matrix X who's nth row is C_n*x.^n
%  The x values run across this matrix, the power value runs down this
matrix
%  Now to create the values of y we have to add the rows together:

if power>0
   y=sum(X);  % when matlab's sum function is applied to a matrix it
              %sums down the columns to give a row vector

else
   y=X;       % if power=0 there is only a single row in X, so no need
              %to sum

end

%Now, to run this function we could do this:

>> [y_0,x_0] = poly_equi(0,3,-5.1,5.0,0.2);
>> [y_1,x_1] = poly_equi(1,[3 2],-5.1,5.0,0.2);
>> [y_2,x_2] = poly_equi(2,[3 2 2],-5.1,5.0,0.2);
>> subplot(3,1,1)
>> plot(x_0,y_0)
>> xlabel('x values')
>> ylabel('y values')
>> subplot(3,1,2)
>> plot(x_1,y_1)
>> xlabel('x values')
>> ylabel('y values')
>> subplot(3,1,3)
>> plot(x_2,y_2)
>> xlabel('x values')
>> ylabel('y values')
```

Note that we ran the function three times – each time we put in different values for the input variables and called the output variables something different each time so that we could store the three different results. We then plotted the results on three different subplots and labeled the axes.

4 Warmup: Complex Exponentials

In the Pre-Lab part of this lab, you learned how to write M-files. In this section, you will write two functions that can generate sinusoids or sums of sinusoids.

4.1 M-file to Generate a Sinusoid

Write a function that will generate a single sinusoid, $x(t) = A\cos(\omega t + \varphi)$, by using four input arguments: amplitude (A), frequency (ω), phase (φ) and duration `(dur)`. The function should return two outputs: the values of the sinusoidal signal (x) and corresponding times (t) at which the sinusoid values are known. Make sure that the function generates 20 values of the sinusoid per period. Call this function `one cos()`. *Hint: use* `goodcos()` *from part (a) as a starting point.*

Demonstrate that your `one cos()` function works by plotting the output for the following parameters: $A = 95$, $\omega = 200\pi$ rad/sec, $\varphi = \pi/5$ radians, and `dur = 0.025` seconds. Be prepared to explain to the lab instructor features on the plot that indicate how the plot has the correct period and phase. What is the expected period in millisec?

4.2 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 function for this operation. To be general, we will allow the frequency of each component (f_k) to be different. The following expressions are equivalent if we define the complex amplitudes X_k as $X_k = A_k e^{j\varphi k}$.

$$x(t) = \Re e\left\{\sum_{k=1}^{N} X_k e^{j2\pi f k\ t}\right\} \tag{7}$$

$$x(t) = \sum_{k=1}^{N} A_k \cos(2\pi f_k t + \phi \tag{8}$$

4.2.1 Write the Function M-file

Write an M-file called `syn sin.m` that will synthesize a waveform in the form of (7). Although `for` loops are rather inefficient in M ATLAB, *you must write the function with one loop in this lab.* The first few statements of the M-file are the comment lines—they should look like:

```
function [xx,tt] = syn_sin(fk, Xk, fs, dur, tstart)
%SYN_SIN Function to synthesize a sum of cosine waves
% usage:
%   [xx,tt]  =  syn_sin(fk,  Xk,  fs,  dur,  tstart)
%     fk  =  vector  of  frequencies
%            (these  could  be  negative  or  positive)
%     Xk  =  vector  of  complex  amplitudes:  Amp*e^(j*phase)
%     fs  =  the number of samples per second for the time axis
%    dur  =  total  time  duration  of  the  signal
&    tstart =starting  time  (default  is  zero,if  you  make  this  input
     optional)
%     xx  =  vector  of  sinusoidal  values
%     tt  =  vector  of  times,  for  the  time  axis
  %
  % Note:  fk  and  Xk  must  be  the  same length.
  %          Xk(1) correspons  to  frequency      fk(1),
  %          Xk(2) corresponds to  frequency      fk(2),  etc.
```

The MATLAB syntax `length(fk)` returns the number of elements in the vector `fk`, so we do not need a separate input argument for the number of frequencies. On the other hand, the programmer (that's you) should provide error checking to make sure that the lengths of `fk` and `Xk` are the same. See `help error`. Finally, notice that the input `fs` defines the number of samples per second for the cosine generation; in other words, we are no longer constrained to using 20 samples per period.

Include a copy of the MATLAB *code with your lab report.*

4.2.2 Default Inputs

You can make the last input argument(s) take on default values if you use the `nargin` operator in MATLAB. For example, `tstart` can be made optional by including the following line of code:

```
        if  nargin<5,  tstart=0,  end     %--default  value  is  zero
```

4.2.3 Testing

In order to use this M-file to synthesize harmonic waveforms, you must choose the entries in the frequency vector to be integer multiples of some desired fundamental frequency. Try the following test and plot the result.

```
[xx0,tt0] = syn_sin([0,100,250],[10,14*exp(-*pi/3),8*j],10000,0.1,0);
%-Period = ?
```

Measure the period of `xx0` by hand. Then compare the period of `xx0` to the periods of the three sinusoids that make up `xx0`, and write an explanation on the verification sheet of why the period of `xx0` is longer.

4.3 Spectrogram

There is a fundamental trade-off between knowing which frequencies are present in a signal (or its spectrum) and knowing how those frequencies vary with time. A spectrogram estimates the frequency content over short sections of the signal. If we make the section length very short we can track rapid changes in the frequency. However, shorter sections lack the ability to do accurate frequency measurement because the amount of input data is limited. On the other hand, long sections can give excellent frequency measurements, but fail to track frequency changes well. For example, if a signal is the sum of two sinusoids whose frequencies are nearly the same, a long section length is needed to “resolve” the two sinusoidal components. This trade-off between the section length (in time) and frequency resolution is equivalent to Heisenburg’s Uncertainty Principle in physics.

The signal may be viewed as two signals with different constant frequencies, *or* as a single frequency signal whose amplitude varies with time. Both views will be useful in evaluating the effect of window length when finding the spectrogram of a signal.

$$x(t) = A\cos(2\pi(f_c - f_\Delta)t) + A\cos(2\pi(f_c + f_\Delta)t) = A[\cos(2\pi(f_\Delta t)]\cos(2\pi f_c t)$$

(a) Create and plot a signal with

(i) f = 32 Hz
(ii) T_{dur} = 0.26 sec
(iii) f_s = 11025 Hz
(iv) f_c = 2000 Hz

(b) Find the spectrogram using a window length of 2048 using the commands:

```
specgram(x,2048,fsamp);  colormap(1-gray(256)).
```

Comment on what you see. Are the correct frequencies present in the spectrogram? If necessary, use the zoom tool to examine the important region of the spectrogram.

(c) Find the spectrogram using a window length of 16 using the commands:

```
specgram(x,16,fsamp); colormap(1-gray(256)).
```

Comment on what you see, and compare to the previous spectrogram.