
DSP First
Lab 02: Introduction to Complex Exponentials

Lab Report: It is only necessary to turn in a report on Section 5 with graphs and explanations. You are asked to **label** the axes of your plots and include a title for every plot. In order to keep track of plots, include your plot *inlined* within your report. If you are unsure about what is expected, ask.

1 Introduction

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

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

2 Pre-Lab

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 LabVIEW to make plots of phasor diagrams that show the vector addition needed when adding sinusoids.

2.1 LabVIEW Review

You learned several LabVIEW techniques during the last lab. Here is a list of some of them:

- Where to find DSP First demos
- How to turn on and use Context Help
- Using right-click to find palettes
- How to search for VIs (blocks)
- Using right-click to attach constants, controls, and indicators
- Where to find the complex operators
- How to create array controls
- How to declare the ‘type’ of an array
- How to change the data representation of a numeric input
- How to generate a cosine waveform
- How to change the sampling rate and number of samples generated
- How to use a **TripleDisplay**.
- How to copy a plot to be pasted into a word processor.

If you don’t remember how to do these, now is a good time to review lab 1 and refresh your memory.

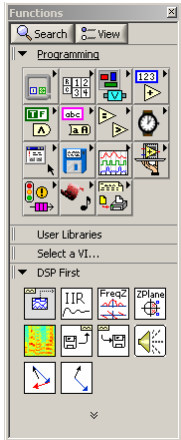
2.2 Complex Numbers in LabVIEW

LabVIEW can be used to compute complex-valued formulas and also to display the results as vector or “phasor” diagrams. For this purpose several new LabVIEW VI’s have been written and are available at www.rose-hulman.edu/DSPFirst. You installed these for the previous lab.

To be sure these have been installed,

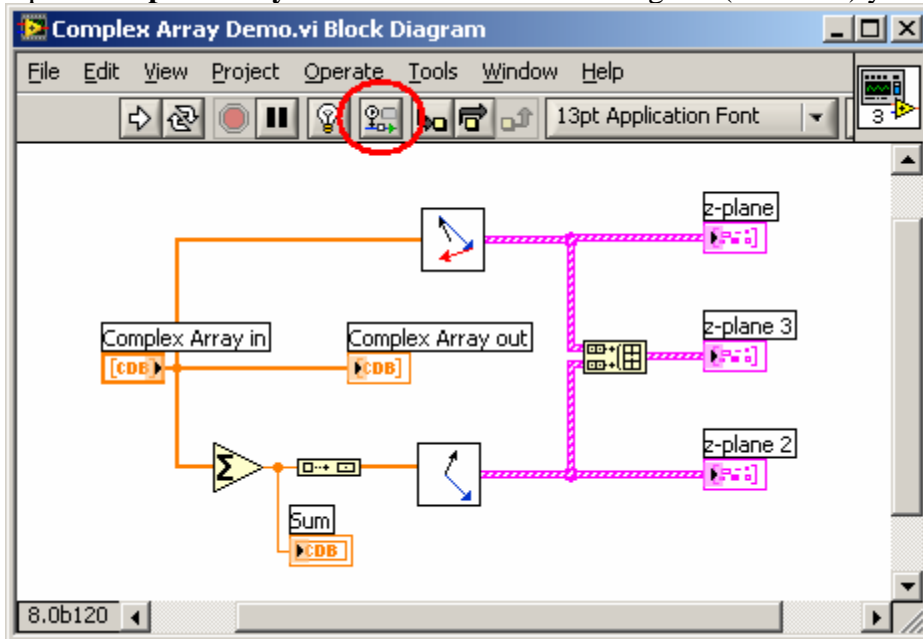
1. Start LabVIEW
2. Open a Blank VI

3. Go to the block diagram view and right-click. You should see



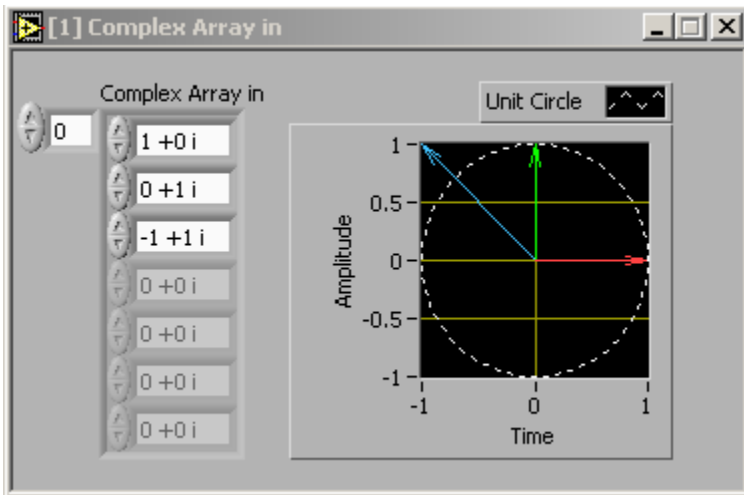
If you don't see the DSP First blocks at the bottom, go to the "Getting Started" section of the CD and following the installation instructions there.

1. Open **ComplexArrayDemo.vi**¹. On the block diagram (<Ctrl-E>) you will see




2. Click on the **Retain Wire Values** button circled above. This allows probes to work after the VI has stopped executing.
3. Right-click on the wire coming from **Complex Array in** and select **Custom Probe»zvectPa.vi**. After running you will see

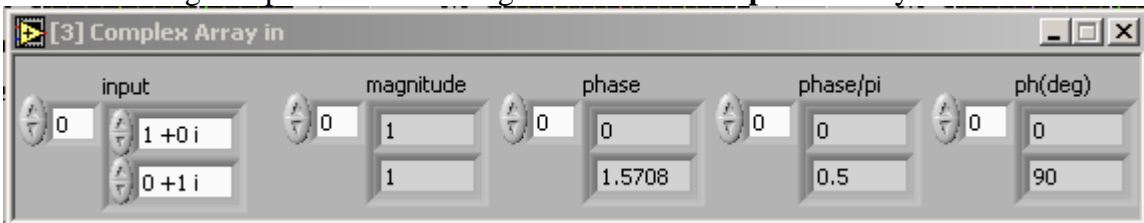
¹ Hint: **Help»Find Examples** and search for **dsp**.



This is showing each of the three complex values with the real part on the x-axis and the imaginary part on the y-axis. It also shows a circle of radius one (the unit circle). Notice

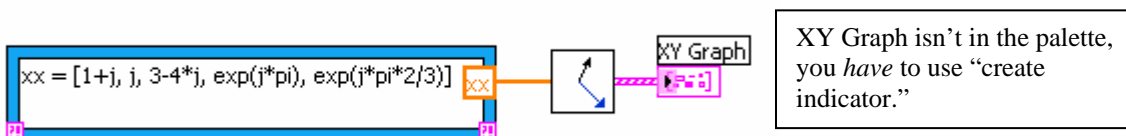
that the **zvect** VI () does the same thing but produces a graph as an output.

4. After removing that probe and selecting **Custom Probe»zprintPa.vi** you will see:



The left most number is the index of the first value of the array that is shown. If you change this value and run the VI again all the other values will change too allowing you to see other values in the array. Try it.

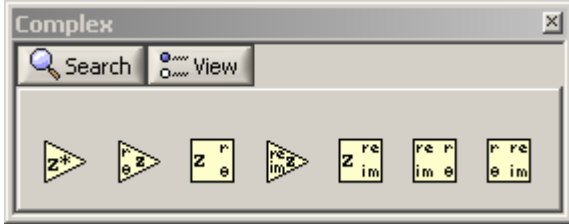
5. A **MathScript** node is an easy way to generate vectors with complex values. The example below generates a vector of five complex values and plots them on the complex plane. Try it. Get a MathScript node² and right click on the right edge and select **Add Output**. Call the output **xx**. Right-click on **xx** and select **Choose Data Type»1D-Array»CDB 1D**. This says **xx** is a vector (1D Array) of complex values. Next enter the expression as shown above³. If you enter a bad expression, the run arrow for the VI will disappear until you fix it. The middle block shown above is a **zvect** VI. Get it from the **DSP First** palette. Right-click on its output and select **Create»Indicator**. The XY Graph should appear. Click Run and see what happens.



Here are some of LabVIEW's built-in complex number operators:

² Hint: Search for it.

³ Does the syntax in the MathScript node look familiar? Yup, it's pretty much MATLAB.



Use the help system to learn what each one does. Each of these takes a vector as its input argument and operates on each element of the vector.

When unsure about a command, use **Context Help** (<Ctrl-h>)

2.3 Sinusoid Addition Using Complex Exponentials

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

$$x(t) = A \cos(2\pi f_0 t + \phi) = \text{Re}\{Ae^{j\phi} e^{j2\pi f_0 t}\} \quad (1)$$

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

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

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*

$$X_k = A_k e^{j\phi_k} \quad (3)$$

Then the complex amplitude of the sum is

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

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

$$x(t) = A_s \cos(2\pi f_0 t + \phi_s) \quad (5)$$

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 $T_0 = 1/f_0$.

2.4 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.,

$$F_k = k f_0 \quad (\text{Harmonic Frequencies})$$

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

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

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.)

2.5 Complex Numbers

This section will test your understanding of complex numbers. Use $z_1 = 10e^{j2\pi/3}$ and $z_2 = -5 + j5$ for all parts of this section.

- (a) Modify your MathScript node⁴ (multiple lines of code are allowed) to generate the complex numbers z_1 and z_2 . Create two separate outputs on your MathScript node. Plot them with `zvect`. See **ComplexArrayDemo.vi** for examples of how this is done. Hint: Use the **Build Array VI**⁵ to combine z_1 and z_2 into an array and then use `zvect`. Don't forget to switch **Build Array** to concatenate⁶.
When unsure about a command, use Context Help.
- (b) The VI `zcat.vi` can be used to plot vectors in a "head-to-tail" format. Use an array control to create a complex array with values $[j, -1, -2j, 1]$. Attach this to the `zcat` VI and graph the results to see how `zcat` works when its input is a vector of complex numbers.
- (c) Compute $z_1 + z_2$ and plot the sum using `zvect`. Then use `zcat` to plot z_1 and z_2 as 2 vectors head-to-tail, thus illustrating the vector sum.
- (d) Compute $z_1 z_2$ and plot the answer using `zvect` to show how the angles of z_1 and z_2 determine the angle of the product. Use the probe `zprint` to display the result numerically.
- (e) Compute z_1/z_2 and plot the answers using `zvect` to show how the angles of z_1 and z_2 determine the angle of the quotient. Use the probe `zprint` to display the result numerically.
- (f) Compute the conjugate z^* for both z_1 and z_2 and plot the results. Display the results numerically with `zprint`.
- (g) Compute the inverse $1/z$ for both z_1 and z_2 and plot the results. Display the results numerically with `zprint`.
- (h) Arrange your front panel into a 2 by 2 grid of graphs that displays the following four plots:
 - a. z_1 and z_2 on the same plot.
 - b. z_1^* and z_2^* on the same plot;
 - c. $1/z_1$ and $1/z_2$ on the same plot; and
 - d. $z_1 z_2$.

2.6 ZDrill

There is a complex numbers drill program called `zdrill` which uses LabVIEW to generate complex number problems and check your answers. Select **Help»Find Examples...** Click the **Search** tab and type **dsp**. Double-click on **DSPFirst**. You will see all the demos for this class. Double-click on **zdrill**. *Please spend some time with this drill since it is very useful in helping you to get a feel for complex arithmetic.*

You can also get to demos via the DSP First web site. Go to the site and click on the [Demos](#) link on the right. Select a demo and it should run in your browser.

⁴ Don't forget to change the output data type to (scalar) CDB.

⁵ Hint: right-click, click **Search**, type **Build**.

⁶ Right-click on the **Build Array** VI and select **Concatenate Inputs**.

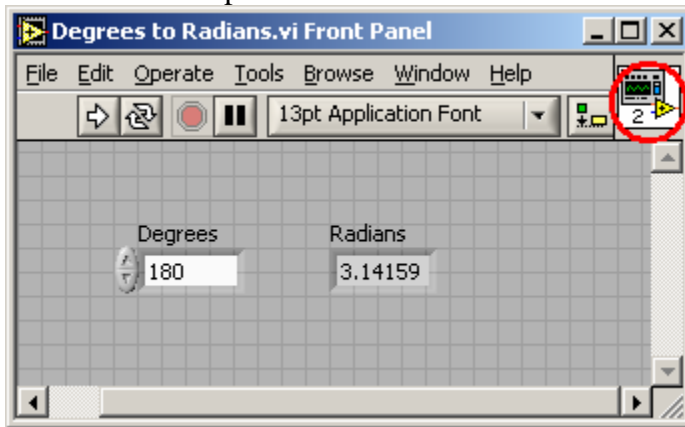
3 Warmup

3.1 Sub VIs

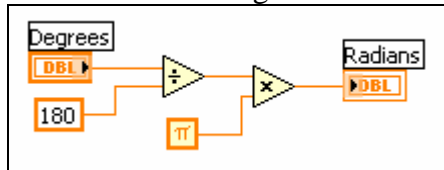
Sub VIs are a special type of VI that can accept inputs and also return outputs. Converting a VI into a sub VI is rather easy. Here's an example that shows how to build a sub VI to convert from degrees to radians.

Front Panel

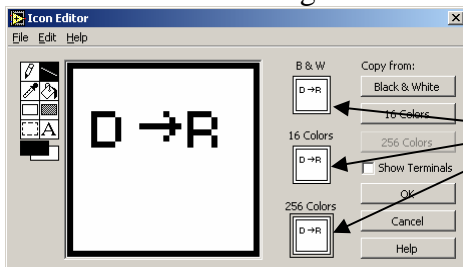
1. Select **File»New** and create a new VI.
2. Build a front panel the looks like



3. Make the block diagram look like



4. Test it to be sure it works correctly.
5. Right-click the icon in the upper right corner of the front panel (circled) and select **Edit Icon** from the shortcut menu. The **Icon Editor** dialog box appears.
6. Double-click the Select tool on the left side of the **Icon Editor** dialog box to select the default icon.
7. Press the <Delete> key to remove the default icon.
8. Double-click the Rectangle tool to redraw the border.
9. Create the following icon.



All three should be the same when you're done.

- a. Use the Text tool to click the editing area.
- b. Type D and R.
- c. Use the Pencil tool to create the arrow.

Note To draw horizontal or vertical straight lines, press the <Shift> key while you use the Pencil tool to drag the cursor.

- d. Use the Select tool and the arrow keys to move the text and arrow you created.
 - e. Select the **B&W** icon and select **256 Colors** in the **Copy from** field to create a black and white icon, which LabVIEW uses for printing unless you have a color printer.
 - f. When the icon is complete, click the **OK** button to close the **Icon Editor** dialog box. The icon appears in the upper right corner of the front panel and block diagram.
10. Right-click the icon on the front panel and select **Show Connector** from the shortcut menu to define the connector pane terminal pattern.
LabVIEW selects a connector pane pattern based on the number of controls and indicators on the front panel. For example, this front panel has one input terminal, **Degrees**, and one output terminal, **Radians**, so LabVIEW selects a connector pane pattern with all these terminals.
11. Assign the terminals to the digital control and digital indicator.
- a. Select **Help»Show Context Help** to display the **Context Help** window. View each connection in the **Context Help** window as you make it.
 - b. Click the left terminal in the connector pane. The tool automatically changes to the Wiring tool, and the terminal turns black.
 - c. Click the **Degrees** control. The left terminal turns orange, and a marquee highlights the control.
 - d. Click an open area of the front panel. The marquee disappears, and the terminal changes to the data type color of the control to indicate that you connected the terminal.
 - e. Click the right terminal in the connector pane and click the **Radians** indicator. The right terminal turns orange.
 - f. Click an open area on the front panel. Both terminals are orange.
 - g. Move the cursor over the connector pane. The **Context Help** window shows that both terminals are connected to floating-point values.
12. Select **File»Save** to save the VI because you will use this VI later in the course. Save it as D to R.

3.2 Vectorization and For Loops

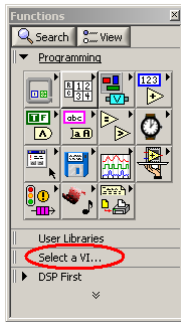
In this section we'll test the sub VI you just created and then show how to use a For Loop to use your VI to convert all the values in an array. First, let's test it.

3.2.1 Using your sub VI

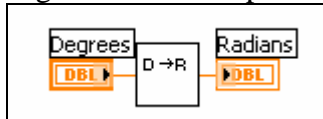
Here's how to use your VI:

1. Select **File»New** and create a new VI.

- Go to the block diagram and right-click. Select **Functions»Select A VI...** (circled)



- Browse for your VI and select it.
- Right-click the input and create a control.
- Right-click the output and create an indicator. It should look like:

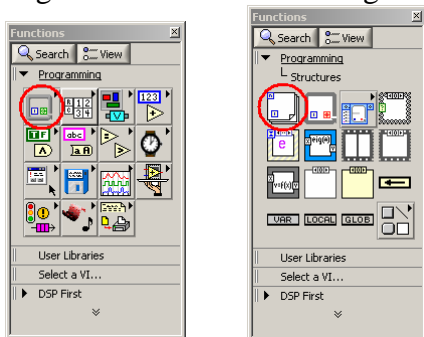


- Test it. Does it work? If not, fix it.

3.2.2 Looping over an array

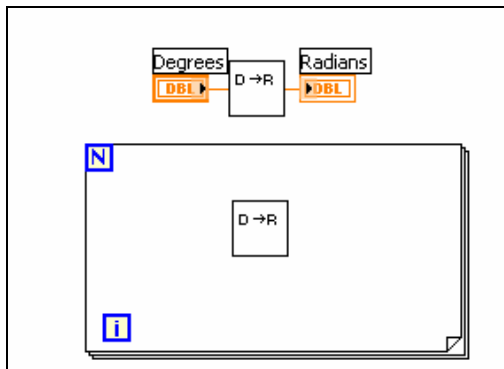
This VI works fine, but suppose you have a whole array of degree values you would like to convert to an array to radians? That's easy if you use a For Loop. Do this:

- Hold down the Ctrl key and click and drag the D to R VI to make a new one.
- Right-click on the block diagram and select **Functions»Structures**.



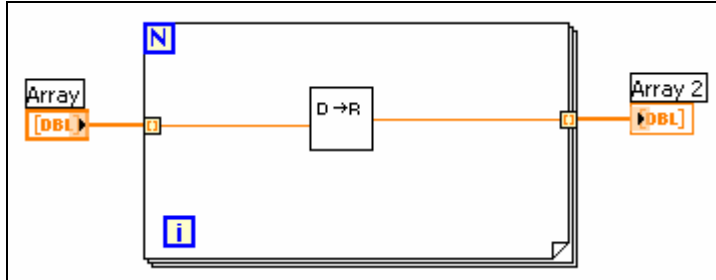
Select the For Loop as shown above.

- Click to the upper-left of the new D to R VI and drag to the lower-right. It should look like:



You've just created a For Loop. This will allow you to run the VI (or whatever you put in the loop) over and over again.

4. Go to the front panel and create an array of doubles⁷.
5. Go back to the block diagram and move the array outside of the loop.
6. Attach the array to the input of the $D \rightarrow R$ VI.



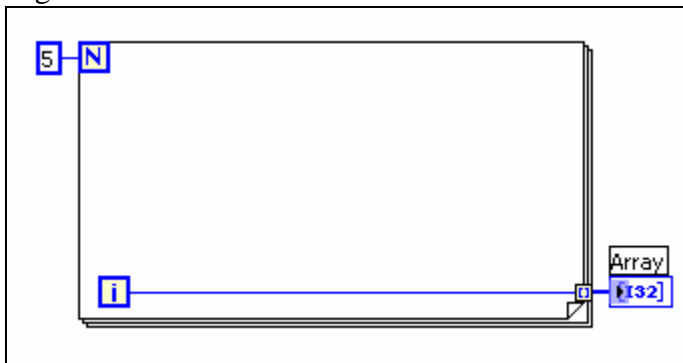
7. Attach the output of the VI to the right edge of the for loop.
8. Right-click on the little square that was just created and select **Create»Indicator**. Your diagram should look like above.
9. Return to the front panel and put a few values in the input array and run the VI. Are the output values correct?

What's going on here? You've just created a for loop that takes the values in the input array, one at a time and runs them through the $D \rightarrow R$ VI. It then takes the output, one at a time and puts them in a new array (Array 2 in this case). This is a very powerful operator that will let you do repetitive tasks with just a simple loop. Look up help of the For Loop to see what all it can do. What is that little blue **i** on the bottom left of the loop? Try connecting it to the right border and connecting it to an indicator like you did for Array 2. Run the VI. What is **i**?

3.2.3 Summing elements with a loop

Suppose you would like to sum up all the values from 1 to 5 using a loop. Here's how you could do it:

1. Select **File»New VI** and create a new VI.
2. Create a For Loop.
3. Right-click on the left terminal of the blue **N**. Select **Create Constant**. Enter 5.



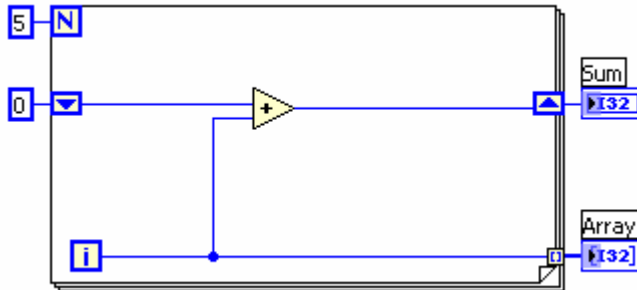
4. Attach **i** to an array and run the VI.
5. What is output to the array?

⁷ Hint: See Lab 1.

3.2.3.1 Working with shift registers

We want to take each value of i and add them up. This means that for each iteration of the loop we want to add i to a running sum.

1. Right-click on the edge of the For Loop and select **Add Shift Register**.
2. Some small connectors will appear. Wire them as shown.



3. Run the VI. The shift register takes the output of the **add** block during one iteration of the loop and feeds it into the input of the same **add** block during the next iteration of the loop. The 0 on the outside left of the loop is the initial value before the loop starts, and the output attached to **Sum** is the value after the last iteration of the loop. What is the value of **Sum**? Is it the sum of 1 through 5? No, we added up 0 through 4.
4. Change the diagram to sum 1 through 5.
5. Now change it to sum 1 through 100.
6. Now change it to sum 6 through 100. How many times do you have to do the loop to do this?
7. Now change it so the input is an array and the output is the sum of all the elements in the array.

4 Complex Exponentials

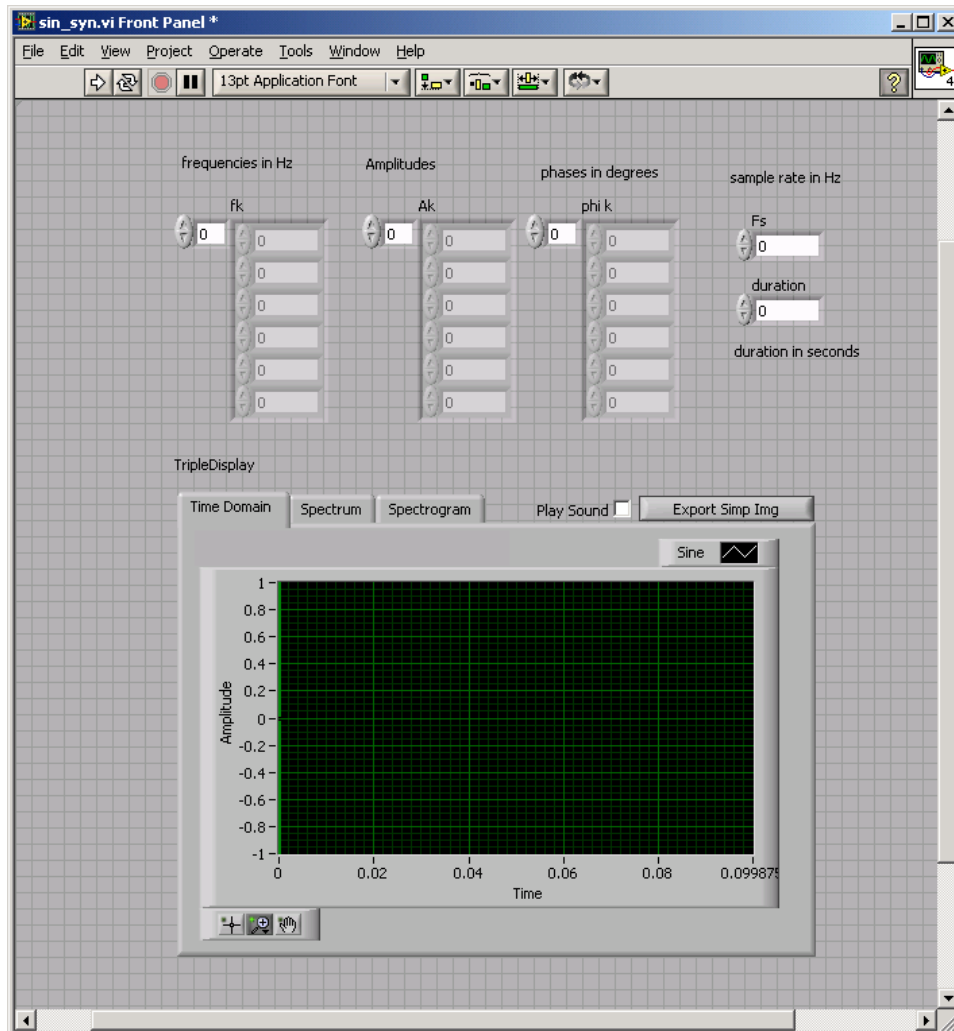
In the Pre-Lab part of this lab, you learned how to write VIs. In this section; you will write a sub VI that can generate the sum of sinusoids.

4.1 Sinusoidal Synthesis with a sub VI: 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 $X_k = A_k e^{j\phi_k}$.

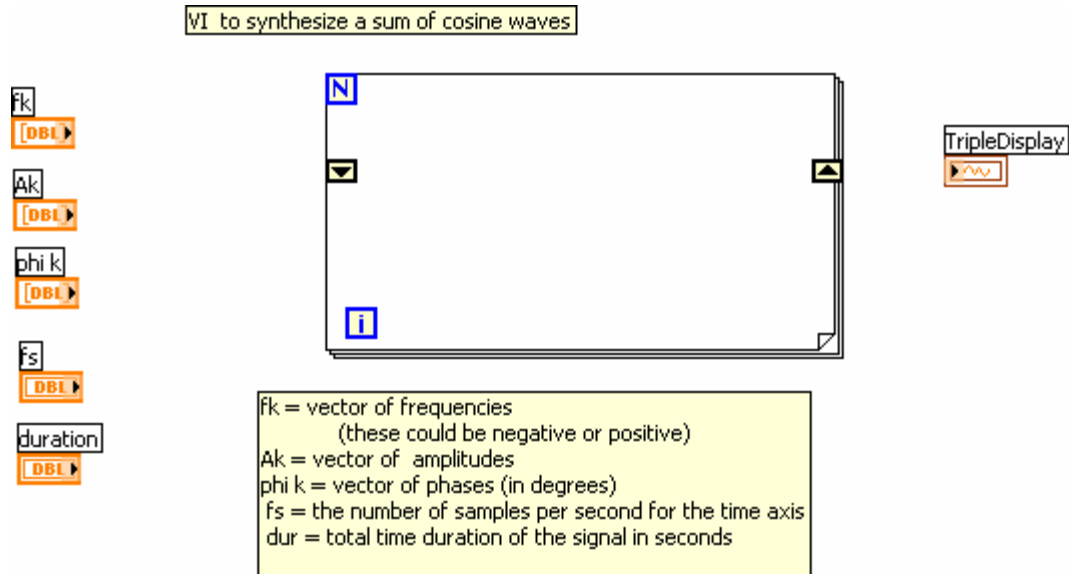
$$x(t) = \operatorname{Re} \left\{ \sum_{k=1}^N X_k e^{j2\pi f_k t} \right\} \quad (7)$$

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



4.1.1 Write the sub VI

Write sub VI called sine_syn.vi that will synthesize a waveform in the form of (8). Above right is what your front panel should look like and below is what block diagram should look like:



Notice that the input f_s 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 **commented** copy of the LabVIEW code with your lab report. Comments are added by simply double-clicking where you want the comment.

4.1.2 Testing

In order to use this to synthesize periodic (harmonic) waveforms, you would simply choose the entries in the frequency vector to be integer multiples of the desired fundamental frequency. Try the following tests and plot the results (if the function does not return an error). Make a subplot with all the working test cases. Measure the period of x_4 (by hand).

Signal	F_k	A_k	ϕ_k	f_s	duration
x1	50	2	$-\pi/2$	1000	0.1
x2	80, 80	1, 1	$\pi/2, \pi$	1000	0.1
x3	50, 50, 50	1, 2, 1	$-\pi/2, 0, -\pi/2$	1000	0.1
x4	100, 150, 200	1, 1, $1/2$	$\pi/2, 0, -\pi/2$	2000	0.1

5 Lab Exercise: Representation of Sinusoids

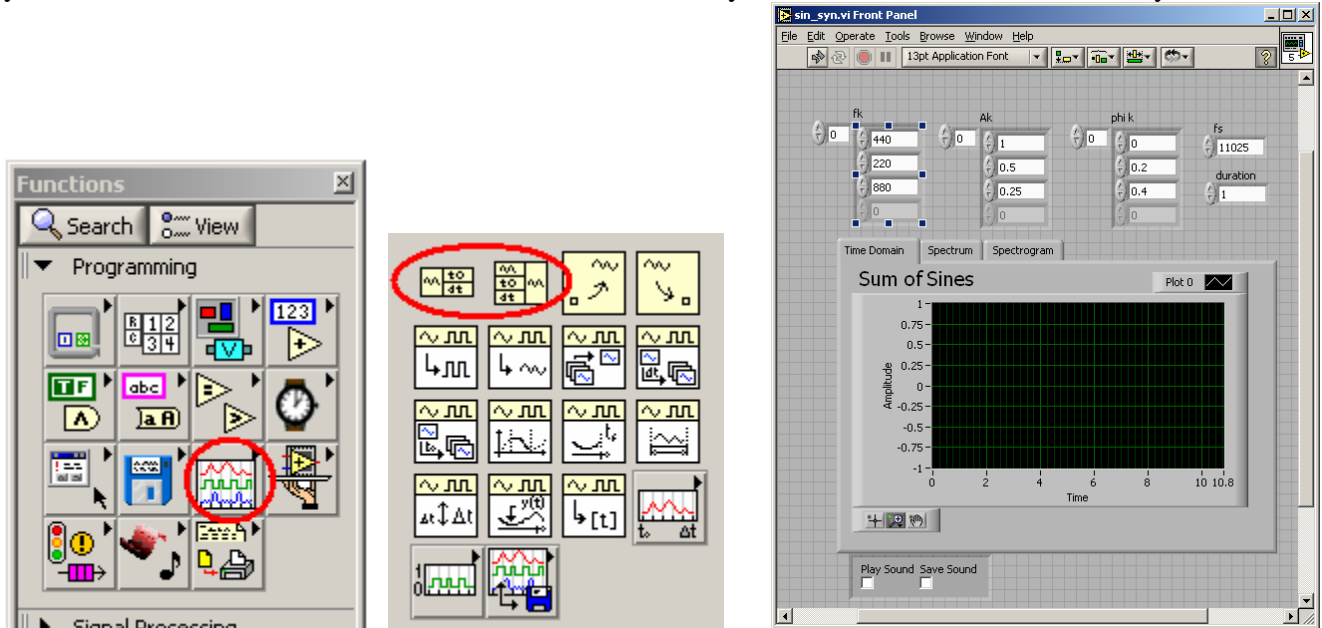
Be aware that you can also use the DSP First probe `zprint` to print the polar and rectangular forms of any vector of complex numbers.

- Generate the signal $x(t) = \text{Re}\{2e^{j\pi t} + 2e^{j\pi(t-1.25)} + (1-j)e^{j\pi t}\}$ and make a plot versus t . Use the `sine_syn` VI and take a range for t that will cover three periods. *Include the LabVIEW code with your report.* Be sure to comment it.
- From the plot of $x(t)$ versus t , measure the frequency, phase and amplitude of the sinusoidal signal by hand. Show annotations on the plots to indicate how these measurements were made and what the values are. Compare to the calculation in part (c).
- Use the phasor addition theorem and LabVIEW to determine the magnitude and phase of $x(t)$.

6 FAQ

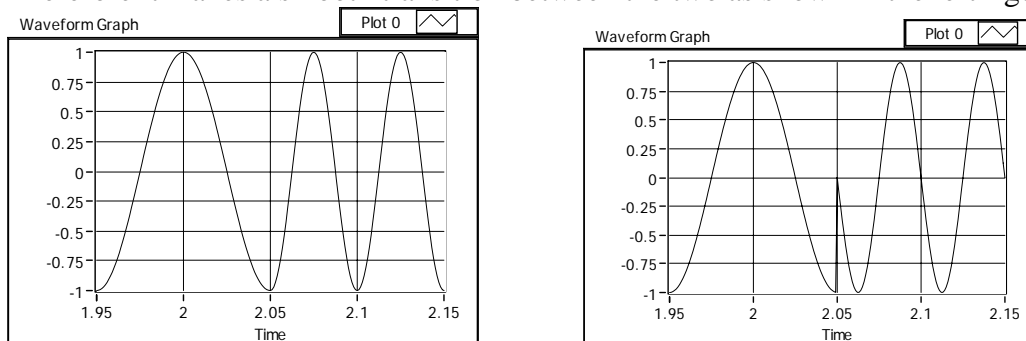
6.1 What is a Waveform?

The **DSP First Function Generator VI** produces an output whose data type is a waveform. The waveform data type carries the data (as an array, Y), time between samples (dt) and the start time of a waveform ($t0$). If you invert dt , you get the sampling rate (f_s) of the signal. We will generally ignore the start time. Click the **Waveform** symbol (circled below left) to see the VIs for working with waveforms. The two most useful VIs are circled in the middle. These VIs let you access the Y and dt values in a waveform, and build your own waveform from an array.



6.2 The phase output of the Generator doesn't appear correct. How do I fix it?

By default the generator assumes you are going to use it to append one waveform after another. Therefore it makes a smooth transition between the two as shown in the left figure.



To do this it must ignore the phase input after the first time through the loop. Attach a **true** constant to the **reset signal** input to have the phase used each time through the loop. You will get a result like the right figure.

6.3 When adding waveforms in a for loop how do I initialize the shift register ?

The easiest way is to attach function generator (outside the loop) to the shift register initialization. Set the amplitude on the generator to **0** and attach the same **sampling info** as you did to the generator in the loop.

6.4 The Sampling Info input to the Waveform Generator has two values, Fs and #s. How do I control these separately?

Use a **Bundle** (Search for **Bundle**). It will combine two inputs into one. Don't worry if you get an error when you attach the Bundle to the generator. The error will go away once you've attached the correct inputs to the Bundle.

8 LabVIEW things to remember for future labs:

- How to use probes.
- How to convert a VI to a sub VI
 - Adding terminals to a sub VI
 - Adding a sub VI to a block diagram
- How to use a for loop
 - Iterating over an array
 - Using N to specify the number of iterations
 - Using i
 - Using shift registers to carry values from one iteration to another.
- ... What else?