PHYS 5061 Lab 3: Math tricks in LabVIEW, building re-usable VI's, saving data.

LabVIEW's graphical programming leads to programs driven by data flow, which is convenient in situations where one collects and processes data from an experiment. But it may seem that one often spends a fair amount of time building a diagram that does some relatively simple mathematical calculation. For example, think about what the block diagram to compute an array of values of $A\sin(\omega t + \varphi)$ for t = 0, 0.01, 0.02, …, 100.00, with values set by controls for the amplitude, frequency (which a user would prefer to give in Hz) and phase. That requires putting down VI icons for the trig function, a constant (2π), a few multiplications, and an addition, along with generating the array of input time values (there's a VI for that since it's such a common task), along with wiring it all up, placing and naming things so the next person who looks at it can figure it out. It's not that hard (although the figuring-out by the next user often is), but sometimes good old-fashioned text-based programming languages can achieve the same goals (functionality and ease-of-understanding) with significantly less effort. LabVIEW offers that capability through two of its programming structures.

The first is the "formula node". This icon expands to a region where one can put in a simple text-based programming calculation. It allows you to wire up input values (from other icons, constants, or front panel controls) to "tunnels" on the math function node frame and then do text-based calculations, i.e. calculate functions of the input values, and finally take the results from output tunnels to further processing or display.

The second and fancier option is the "MathScript node," which does much the same thing, but knows a larger set of high level commands. Essick makes use of this. It understands the same syntax and has many of the basic same functions built into it as Matlab. If you've used Matlab previously, it will be familiar. Unfortunately our version of LabVIEW has disabled Mathscript. But, we can call Matlab directly from LabVIEW using the Matlab Script Node, found under Functions - Mathematics – Script/Formula – Script Nodes – Matlab Script.

Fire up a copy of Matlab so that it is running in the background and can handle requests to do calculations from LabVIEW. The Matlab Script should perform very much like the Mathscript Node used in Essick. Some points to note in using Matlab vs Mathscript: Essick's *linramp* becomes *linspace* in Matlab; *gensignal(…)* becomes *gensig(…)* You can get help on these items by clicking on Matlab to bring it up and typing help *functionname* in the Matlab command window.

(1) Work through chapter 4 of Essick to become familiar with the basic functionality of these features. This chapter also introduces you to how to make your VI a re-usable entity. Having written a carefully designed VI to perform a useful task, you can give it its own icon, with input and output connections, save it, and use it as just another element you simply drop into a VI to perform a more complicated task. Frequently repeated tasks can be coded in LabVIEW once, with controls serving to input key variables, and then used repeatedly in future work. Be sure to save your AM Wave VI from the Do-it-

yourself exercise at the end of the chapter, and print out the front panel and block diagram for your VI for your lab notebook. You may skip the "Use It!" section.

(2) Complete problem 6 from chapter 4. Add an additional (Boolean) control named "Save" to the front panel so that when it is true, the θ,R data is written to a text file suitable for input to other programs. Write the data (both θ and R values, one column for each) to a text file using the "write-delimted-spreadsheet file" VI described at the start of chapter 7. You can assemble the θ and R data into a single 2-D array and use the 2D data option on the file-writing VI. There is also a transpose option on that VI that is useful in getting data out in columns vs. rows. Be sure to look at your output data file to be sure it's human-readable. Save your complete VI and print out the front panel and block diagram. Print out a copy of your output data file for your notebook, too.

(3) Complete problem 8 at the end of chapter 4 and save the VI. **IMPORTANT NOTE**: do ***NOT*** name your VI 'polyfit.vi.' Such a beast already exists in LabVIEW, and you will break both your version and the LabVIEW version if you do. And it is time consuming to fix in LabVIEW. Print out the VI front panel and block diagram with a sample of its results.

Always comment/annotate the block diagrams of your VI's to provide guidance as to how the VI works.