REU 2018: Introductory LabVIEW Workshop

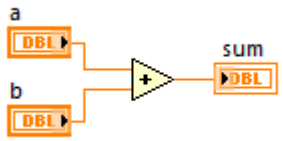
M.P. Hasselbeck, University of New Mexico

SUMMARY: This hands-on workshop will provide an introduction to the capabilities of the National Instruments LabVIEW software package. LabVIEW is easy to learn and can be very useful for automated instrument control and data recording. We will use LabVIEW to setup a program (called a Virtual Instrument or VI for short) to control the temperature of a simulated optical cryostat. Optical cryostats are used in some of the research labs in our department. These instructions are designed to be used in tandem with instructor guidance.

Part 1. Open LabVIEW and create a new Virtual Instrument (VI). There are two primary interfaces: the Front Panel and the Block Diagram. The user interface is on the Front Panel and the program code goes in the Block Diagram. You can place these two windows side-by-side or toggle between them.

To see how LabVIEW works, we will implement the simple arithmetic operation $a + b = sum$. Go to the Front Panel of your blank VI. Select View: Tools Palette. The top button in the palette enables/disables automatic tool selection. For a new user, it's probably a good idea to disable this function. Select the arrow tool. Right click anywhere on the blank Front Panel to open the Controls palette. Pin this window open at its upper-left corner. Select and place two numeric controls and label them **a** and **b**. Next right-click and place a numerical indicator called **sum** on the Front Panel.

Go to the Block Diagram and identify the three terminals (two inputs, one output). These icons are orange double-precision floating point, but notice the different color shading and location of the small black arrow that distinguishes inputs and outputs. Right-click in a blank area and open the Numeric palette. Pin it open at the upper-left corner of the window. You will see icons representing the various mathematical functions including x^2, $\sqrt{x}$, and $(-x)$. We want the addition operation. Select it and drop it into the Block Diagram. Use the Connect Wire tool to implement the equation above; this is a left-click and drag operation to connect the various ports on the terminal. The Block Diagram should look similar to this:



LabView is constantly monitoring for errors while you are building a VI. If one or more problems exist, the run button (arrow) on the upper-left menu bar will appear broken. If

you click on the broken arrow, a window will open listing all the existing errors. When the execution arrow is solid, the VI will run. Switch to the Front Panel, choose two input values (*a,b*) and push the run button to generate the sum. These are floating point values, so digits to the right of the decimal points are valid inputs.

Go to the Block Diagram and click Highlight Execution (light bulb icon). Run the VI again. This slows down the operation so that you can see the flow of information.

Part 2. When working with LabView, it's usually a good idea to have the Context Help enabled. This is found in the Help menu, clicking the question mark button on the menu bar, or by typing CTRL-H. When you scroll over an icon or object, a Help window will provide useful specific information. LabView also uses "tip strips", which can display information when the mouse is hovered over a control or indicator of a running VI.

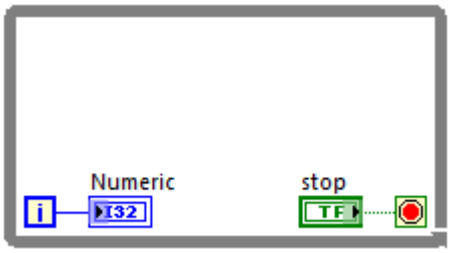
Open a new blank VI. Go to the Block Diagram, right-click, and select Structures: While Loop. A square dotted cursor will appear; use this to create a While Loop. This structure executes all the code contained within it over and over until it is commanded to stop.

Note that the Run arrow is broken; clicking on this broken arrow reveals that a "Conditional terminal is not wired". This terminal defines the stop condition for the While Loop and appears in the lower-right corner of the structure. The stop terminal is expecting a Boolean (T/F) input. Generate a Boolean control by right-clicking on the terminal and "Create Control". A Stop Button terminal appears along with a corresponding control on the Front Panel. This will result in a working VI, although with no useful code.

Notice that the Boolean control and its connection wire are green. In the first block diagram the icons were both orange corresponding to floating point values. These colors help the programmer identify the different data types that are in use.

The blue terminal **[i]** in the lower-left is outlined in blue, representing an integer index that counts each time the While Loop executes. Right-click on it and create an indicator (see diagram below); this will also appear on the Front Panel. Note that the indicator icon is also colored blue and denoted as **I32**, which corresponds to a 32-bit integer. The first iteration of a While Loop is always 0, not 1.

Go to the Front Panel and run the VI. Depending on the processor speed and computer resources, the loop will likely execute many millions of times in just a few seconds. Stop the VI with the button on the panel.



It can be a dangerous programming practice to let a While Loop run as fast as possible because it will attempt to draw the maximum CPU cycles, potentially slowing down other important tasks. While Loop execution can be slowed down with the use of timing functions. Right-click the Block Diagram and select Timing: Wait (ms) and place this icon inside the While Loop. Right-click on the left icon input terminal and Create Constant. This will produce an integer constant of 0. Set the value to 1000 ms, i.e. 1-second and run the VI again. Verify that the loop count increments at 1-Hz.

Go to the Block Diagram and place the stop button icon outside the While Loop. Re-connect it to the conditional stop terminal inside the loop. The point where the wire passes through the While Loop structure is called a tunnel. Run the VI and notice that the stop button is unresponsive. The program can only be stopped with the Abort button on the menu bar. This illustrates an important aspect of data flow programming – when a loop is running it will ignore any input that is *outside* the structure. Data can be wired into a loop from outside, but once a loop is started anything present on input tunnels is ignored. Just as important: Data only propagates through output tunnels when the loop has stopped executing.

Part 3. Put the stop button back inside the While Loop and re-connect it to the conditional stop. Use the Numeric palette to take the square of the loop iteration counter each time the loop executes. You will need a numeric indicator to display the result on the Front Panel, which should be the integer sequence 0, 1, 4, 9, 16...

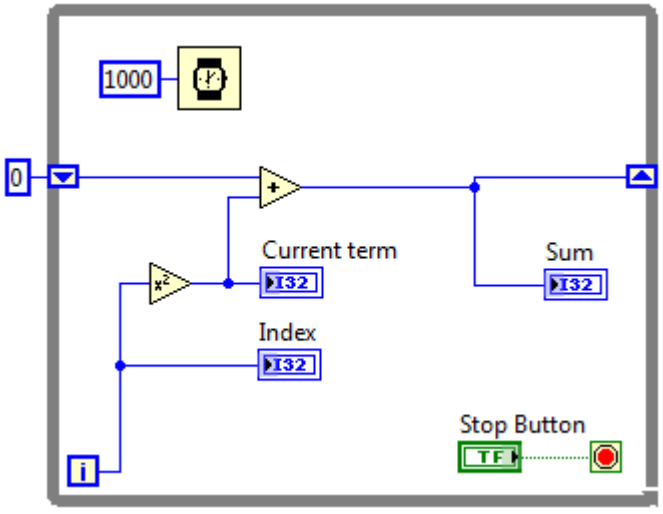
Now we will sum these terms to generate the series:

$$\sum_{i=0} i^2$$

This will require that information from each loop iteration be available for the next calculation in the sequence. Specifically, we must add each new term i^2 to update the sum.

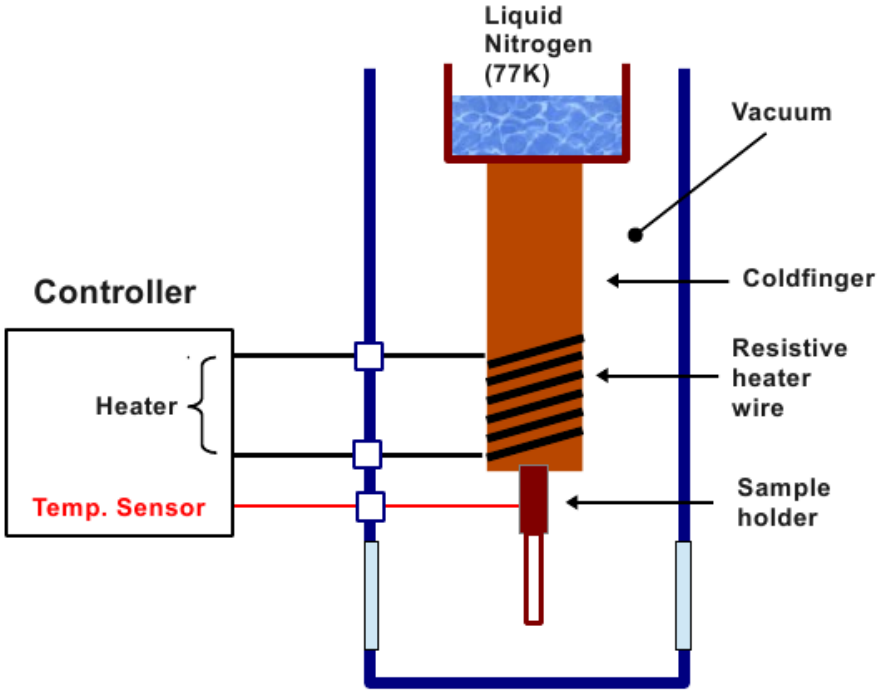
To pass information between loop iterations, LabVIEW makes use of a Shift Register. Right-click on any edge of the While Loop structure and Add Shift Register. You will see a tunnel on the left-side of the structure with a downward pointing arrow and a second tunnel on the right-side with an upward arrow. These tunnels are where data from the previous iteration is pulled in and out, respectively.

Setup a VI to generate the series using the Block Diagram shown below. The running sum exits each iteration at the right tunnel and re-enters at the left tunnel for the next iteration. On the very first iteration, data in the Shift Register is undefined. We initialize it to zero by connecting it to a constant outside the loop. This can be easily accomplished by right-clicking on the terminal and selecting Create Constant. The Shift Register changes color to show the data type it handles. Here it is blue for integer.



There are three indicators to show the loop iteration count i, the current term i^2, and the running sum. By placing Sum inside the loop, its value is updated on each loop iteration. If placed outside the loop, we will not see the result of the summation until the loop is stopped.

Part 4. LabVIEW is used to control the temperature of a simulated optical cryostat shown below.



A liquid nitrogen reservoir at 77 K is thermally linked to a sample holder via a conductive copper shaft called a coldfinger. The liquid nitrogen reservoir, coldfinger, and sample are insulated from room temperature laboratory air by a vacuum enclosure. A sensor reads the temperature of the sample. Temperature is adjusted by supplying current to a resistive heater wire that is wound around the coldfinger. Flat windows on the enclosure allow optical access to the sample for experiments, eg. a laser beam.

Temperature control can be understood as a balance between conductive heat removal by the liquid nitrogen reservoir (modeled by Fourier's Law) and Joule heating in the resistive wire (described by Ohm's Law):

$$\frac{dT}{dt} = -\beta \Delta T + \alpha I^2 R$$

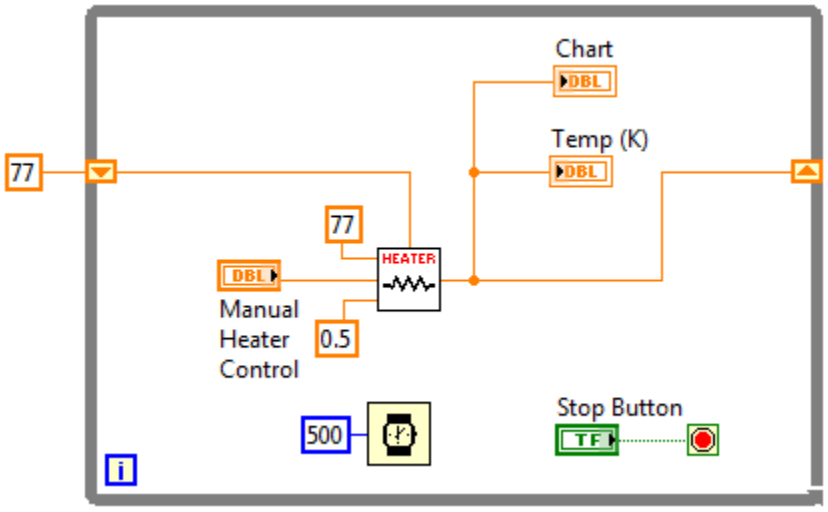
In this differential equation, I is DC current, R is the resistance of the heater wire, and α and β are constants. ΔT is the temperature difference between the sample and the coldfinger that remains fixed at 77K. This system can be setup and analyzed in LabVIEW by re-casting the differential equation as a difference equation. The differential time dt is implemented with discrete time-steps Δt:

$$T_{i+1} - T_i = \left[-\beta(T_i - 77\text{K}) + \alpha I^2 R\right] \Delta t$$

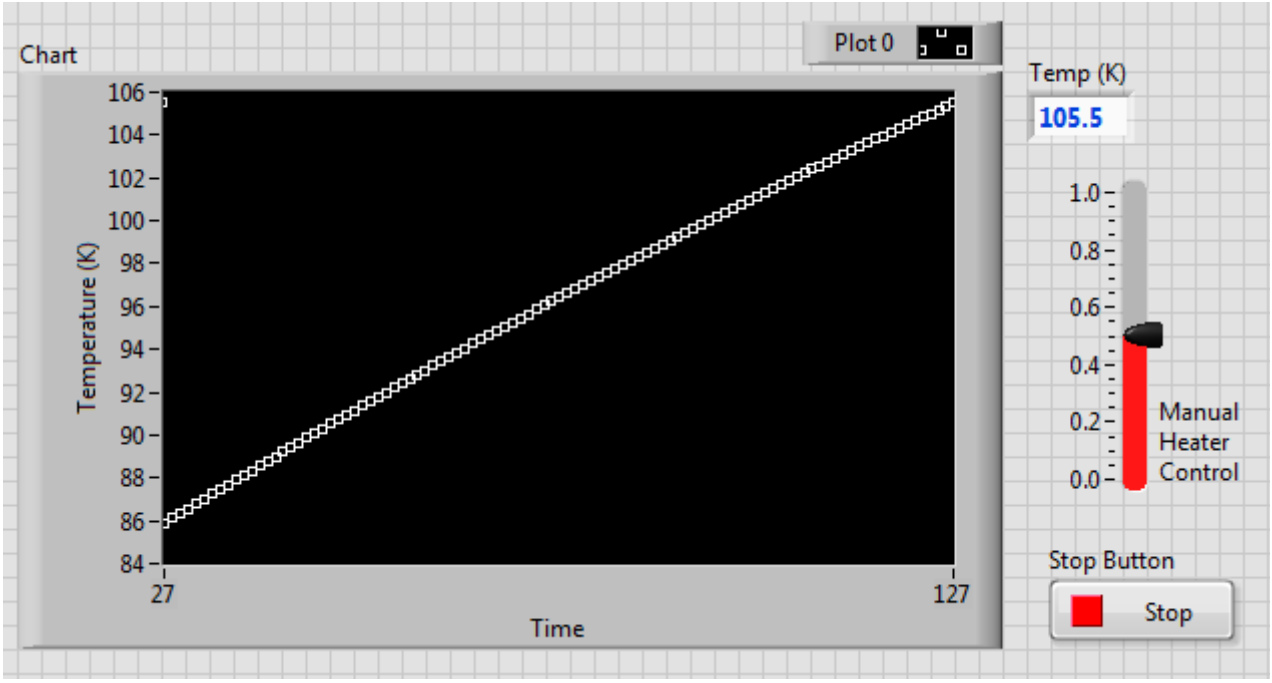
This allows LabVIEW to calculate a new temperature T_{i+1} on each iteration.

Create a While Loop and add a control button to the conditional stop. The time-behavior of the cryostat is modeled with a SubVI called heater2.vi found on the class homepage. Save it to your computer. Right-click inside the While Loop and highlight "Select a VI...". Browse to the location of heater.vi and select it. You can also drag it directly into the Block Diagram.

The While Loop should execute at 2 Hz. Right-click and select Timing: Wait (ms). Create a 500 ms constant and wire it to the left terminal of the Wait function. This sets the time step Δt. This time step must also be set on the **dt** input terminal of the SubVI. This input is specified in seconds, so wire a 0.5 constant there. Place a Shift Register on the While Loop by right-clicking on it and selecting "Add Shift Register". This is used to pass the temperature T_i between loop iterations. Connect the Shift Register output terminal (right-side arrow) to **Tout** on the SubVI. Connect **Tin** to the Shift Register input terminal. Create a constant 77 on its input (left-side arrow) to initialize the Shift Register. Wire a 77 constant to the **Tamb** terminal of the SubVI.



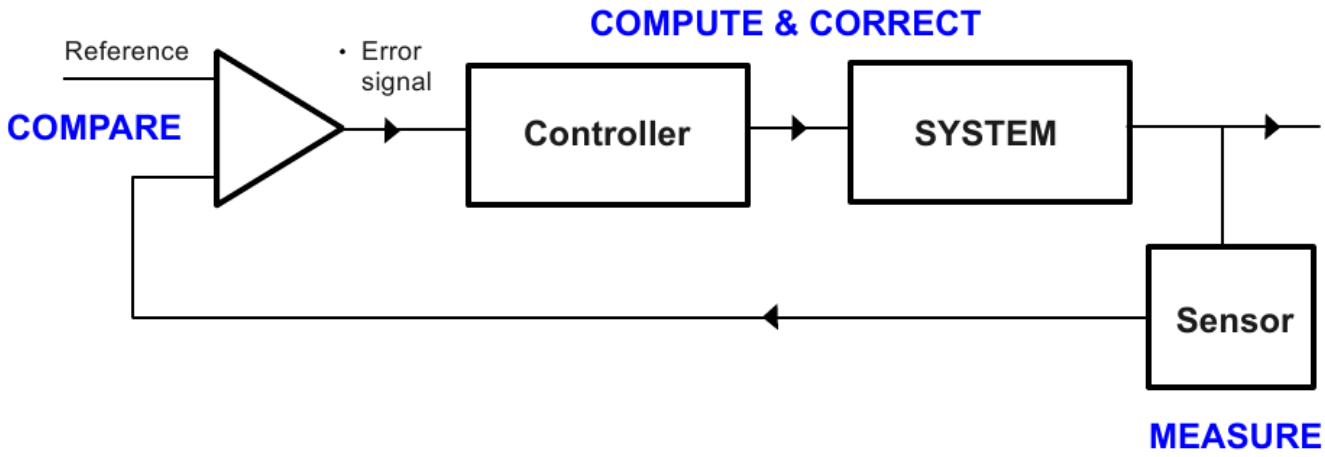
Right-click on the current input terminal of the SubVI and create a control. Double-click on the icon to find this control on the Front panel. Right-click on the control and Replace: Numeric: Vertical Pointer Slide. Right-click on the slider, select Properties, select the Scale tab, and change Maximum to 1 on the Scale Range. Right-click on the Front Panel and select Graph; Waveform Chart. Right-click on the Plot Legend and choose the common plot depicting individual points. Go back to the Block Diagram and wire the Waveform Chart to the SubVI output terminal labeled **Tout**. Right-click on the **Tout** terminal and create a numerical indicator. The sample temperature will be shown on this indicator and displayed graphically on the chart.



The above Front Panel has been modified from the default values. The y-axis of the chart has been changed to Temperature (K) and the controls have been expanded and different colors used. This customization is not required to get a working VI.

When you have successfully built this VI, the run arrow on the menu bar should be unbroken. Set the manual heater control to zero and run the VI. Because no current is passing through the heater wire, the temperature should be at a constant 77K. Increase the current to attain a target temperature, for example 100K. Too much current will cause an overshoot and too little will prevent the desired temperature from being reached. You will find that it is very difficult to hit a desired setpoint using the manual control. In a real experiment, holding the setpoint is even more difficult because of changing conditions, eg. evaporating liquid nitrogen.

Part 5. LabVIEW can automate temperature control. This is done with a closed-loop controller, with operation illustrated in the following system diagram:



The idea is to measure the system temperature and compare to a reference, which is the desired temperature setpoint. The difference between the actual temperature and setpoint is the error signal. The error signal determines how much current is sent to the heater. This operation replaces the manual heater control.

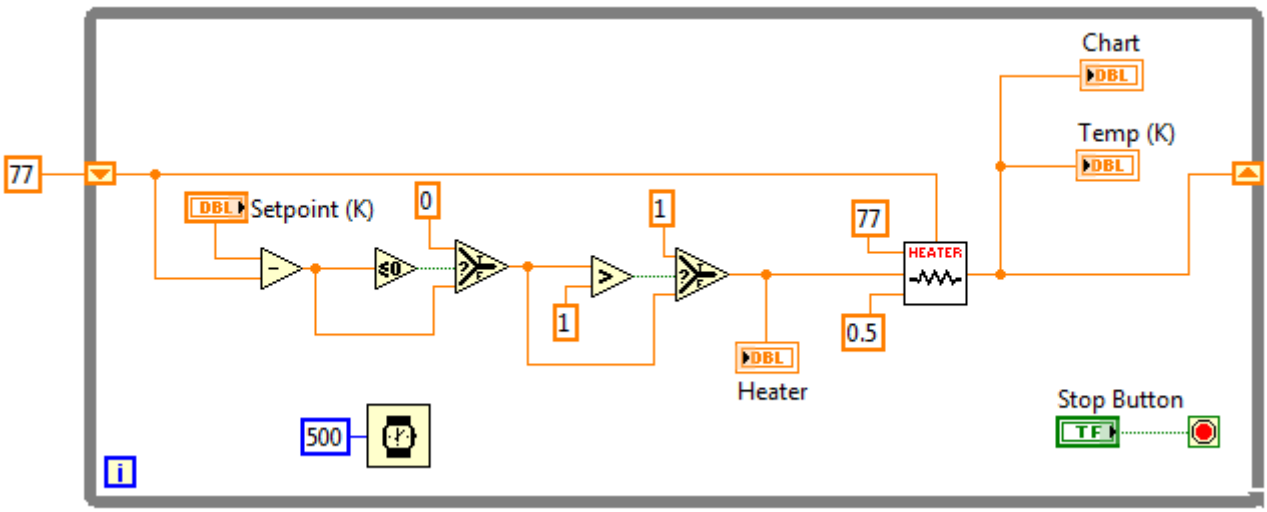
The error signal is $T_{\text{setpoint}} - T_{\text{i}}$, where T_{i} is the current temperature. We must use care, however, to prevent our simulation from being unphysical. There are two issues:

First, if the heater pushes temperature above the setpoint, the error signal is negative. If temperature goes above the setpoint, we want the sample to cool but it makes no sense to apply negative current to the heater. The heater works no matter which direction current is flowing. Our only option is to turn the current off and let the temperature drift back to 77K.

Second, there is a limit to how much current can be produced. If the sample temperature is far below the setpoint, the error signal will be a large positive number. We must clamp the maximum current to 1, which is the limit of the heater power supply.

These conditions require us to make *comparisons* and *decisions* in the program logic. Specifically, if the error signal is negative, the current is set to zero. If the error signal is greater than 1, the current must be clamped at 1. This is easily coded into the LabVIEW Block Diagram as shown below.

The existing While Loop can be expanded to make more space. Select the arrow tool, place it to the left of the SubVI, hold down the Ctrl key, right-click and drag the cursor. Delete the manual heater control. Add a Setpoint control for the Front Panel.

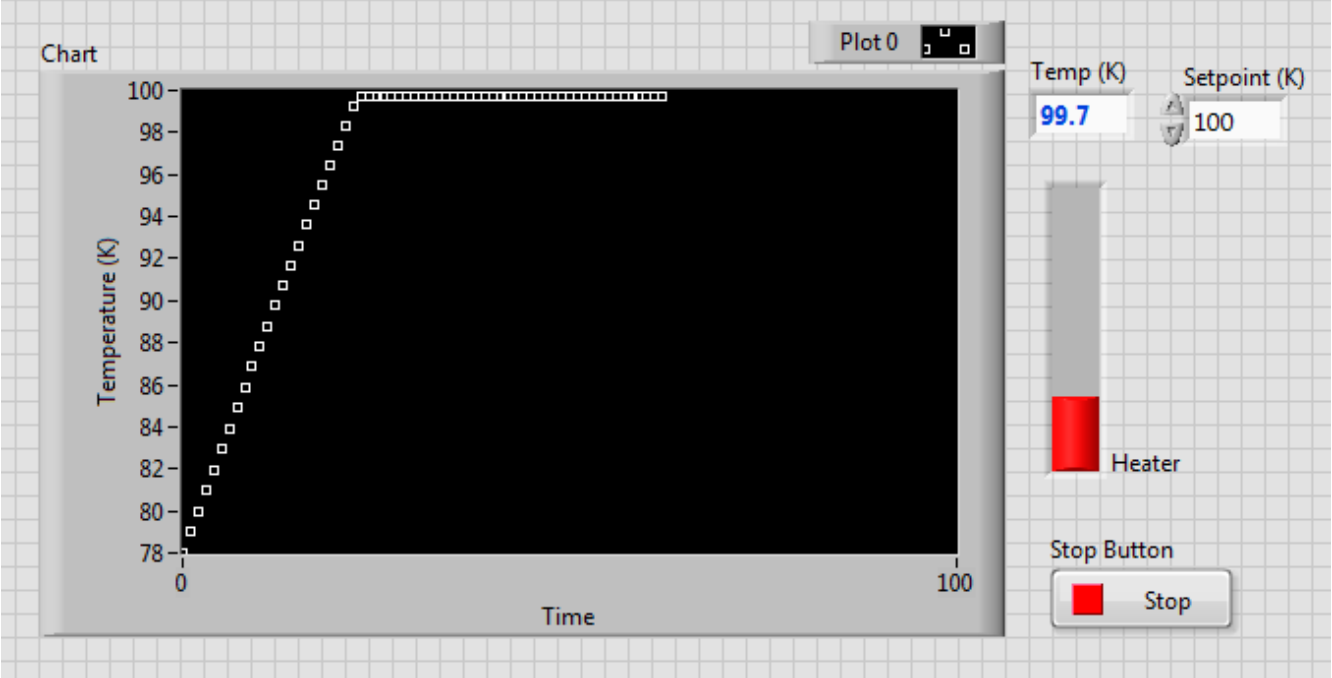


LabVIEW must check if the error signal is negative. Right-click in a blank area and pin open the Comparison palette. Find the ≤ 0 function and place it in the Block Diagram. The output of a comparison is Boolean TRUE or FALSE, shown by the green data line. The T/F Boolean output is used to decide whether to make the heater current zero. Find the Select function in the Comparison palette and add it to the Block Diagram as shown. LabVIEW's Context Help will assist in connecting it. Wire the Boolean output of the comparison to the center terminal of Select. If the comparison is TRUE, the top terminal is selected. Place a constant 0 there. If FALSE, the error signal is unaffected. Wire the error signal directly to the lower terminal of Select.

Next, LabVIEW will check if the error signal is > 1. Find the > function in the Comparison palette, and wire it as shown. If the error signal exceeds 1, the Select function clamps it at 1. Otherwise, the error signal propagates through to the current input of the SubVI.

Even though we are automating temperature control, it's a good idea to have a Front Panel monitor to watch what the heater is doing. Right-click on the current wire feeding the heater and create an indicator. Double-click on the icon to take you to the Front Panel. Right-click and replace the indicator with a vertical progress bar. Set the scale to the range 0–1, just as was done with the manual heater control. A customized Front Panel is shown below.

On the Front Panel, change the Setpoint to 77K. Run the VI and confirm that the operating temperature holds constant at 77K. Change the Setpoint to 100K and observe that the heater takes the temperature there in less than 30 seconds and holds it steady. There should be a constant heater current and a slight negative offset from the setpoint. Change the setpoint to 98K and confirm that the controller current immediately shuts off



to allow the temperature to fall. The current should turn on when the temperature drops below 98K. Considerably more current and time will be needed to reach and maintain higher temperatures in the neighborhood of 200K.

This is a simulation and a real laboratory cryostat would not respond as rapidly as demonstrated here. Most temperature controllers have more sophisticated correction parameters to allow for better dynamic response to the changing environment.