Introduction to LabVIEW (Cont.)

GRAPHICAL PROGRAMMING
FOR ENGINEERS AND SCIENTISTS

mi.com
Lab Goals

• Learn LabVIEW concepts
  – Working with complex data types such as arrays clusters
  – Waveform generation
  – Displaying results on charts and graphs
• Become comfortable with the LabVIEW MathScript environment
  – MathScript syntax
  – Writing and running custom .m files
• Running scripts using LabVIEW MathScript Node

This lab prepares you to do the following:
• Use arrays and clusters to in a LabVIEW applications.
• Generate waveforms and display them on the front panel.
• Write custom MathScripts and use them in LabVIEW.
Section I – Complex Data Types

A. Introduction to Arrays
   • Building arrays
   • Auto-Indexing
   • Array manipulation

B. Introduction to Clusters
   • Building clusters
   • Accessing cluster data

C. Generating Signals
   • Express VI
   • Waveform generation

D. Graphs and Charts
A. Introduction to Arrays

• An array consists of elements and dimensions
  • Elements: data that make up the array
  • Dimension: the length, height, or depth of an array

• The element selected in the index display always refers to the element shown in the upper left corner of the element display.

An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as \(2^{31} - 1\) elements per dimension, memory permitting. Note that an array has 2 components: An index display and an element display.

Consider the following array control on the Front panel.

Notice that the first element shown in the array (3.00) is at index 1 and the second element (1.00) is at index 2. The element at index 0 is not shown in this image, because element 1 is selected in the index display.

You can build arrays of numeric, Boolean, path, string, waveform, and cluster data types. Consider using arrays when you work with a collection of similar data and when you perform repetitive computations. Arrays are ideal for storing data you collect from waveforms or data generated in loops, where each iteration of a loop produces one element of the array.
Creating an Array (Step 1 of 2)

1. Place an array shell from the Controls»Modern»Array, Matrix, and Cluster subpalette, select the Array icon. Drop it on the front panel.

To create an array control or indicator as shown, select an array on the Controls»Modern»Array, Matrix, and Cluster palette, place it on the front panel, and drag a control or indicator into the array shell. If you attempt to drag an invalid control or indicator such as an XY graph into the array shell, you are unable to drop the control or indicator in the array shell.

You must insert an object in the array shell before you use the array on the block diagram. Otherwise, the array terminal appears black with an empty bracket.
Create an Array (Step 2 of 2)

2. Insert data type into the shell (i.e., numeric control).

To add dimensions to an array one at a time, right-click the index display and select **Add Dimension** from the shortcut menu. You also can use the Positioning tool to resize the index display until you have as many dimensions as you want.

**1D Array Viewing a Single Element**

**1D Array Viewing Multiple Elements**

**2D Array Viewing a Single Element**

**2D Array Viewing Multiple Elements**
Arrays – Auto-indexing

• If you wire an array to/from a For Loop or While Loop, you can link each iteration of the loop to an element in that array by enabling auto-indexing on tunnel.

• The tunnel changes from a solid square to the image shown above to indicate auto-indexing.

If you enable auto-indexing on an array wired to a For Loop input terminal, LabVIEW sets the count terminal to the array size so you do not need to wire the count terminal. Because you can use For Loops to process arrays one element at a time, LabVIEW enables auto-indexing by default for every array you wire to a For Loop. You can disable auto-indexing if you do not need to process arrays one element at a time.
Building Arrays with Loops (Auto-Indexing)

- Loops can accumulate arrays at their boundaries with auto-indexing
- For Loops auto-index by default
- While Loops output only the final value by default
- Right-click tunnel and enable/disable auto-indexing

**Array**: Arrays group data elements of the same type. An array consists of elements and dimensions. Elements are the data that make up the array. A dimension is the length, height, or depth of an array. An array can have one or more dimensions and as many as \((2^{31}) - 1\) elements per dimension, memory permitting.

For Loops and While Loops can index and accumulate arrays at their boundaries. This is known as auto-indexing.

- The indexing point on the boundary is called a tunnel
- The For Loop is auto-indexing-enabled by default
- The While Loop is auto-indexing-disabled by default

Examples:
- Enable auto-indexing to collect values within the loop and build the array. All values are placed in the array upon exiting the loop.
- Disable auto-indexing if you are interested only in the final value.
Array Manipulation

• Functions from the Array subpalette

Most of the common array manipulation functions such as determining an array size, indexing an element in an array or building arrays can be found in the Array subpalette under Programming»Array.

The following are some of the standard array manipulation functions.

Array Size: This function returns the size of an array.

Index Array: This function returns the element specified at ‘index’ from an array.

Build Array: This function appends elements together to create an array.
Clusters group like or unlike components together. They are equivalent to a record in Pascal or a struct in ANSI C. Cluster components may be of different data types.

Examples

- Error information – Grouping a Boolean error flag, a numeric error code, and an error source string to specify the exact error.
- User information – Grouping a string indicating a user’s name and an ID number specifying the user’s security code.

All elements of a cluster must be either controls or indicators. You cannot have a string control and a Boolean indicator. Think of clusters as grouping individual wires (data objects) together into a cable (cluster).

The following is an example of a cluster used to store student information.
Creating a Cluster

1. Select a **Cluster** shell.
2. Place objects inside the shell.

**Controls»Modern»Array, Matrix & Cluster**

Create a cluster front panel object by choosing **Cluster** from the **Controls»Modern»Array, Matrix & Cluster** palette.

- This option gives you a shell (similar to the array shell when creating arrays).
- You can size the cluster shell when you drop it.
- Right-click inside the shell and add objects of any type.

**Note:** You can even have a cluster inside of a cluster.

The cluster becomes a control or an indicator cluster based on the first object you place inside the cluster.

You can also create a cluster constant on the block diagram by choosing **Cluster Constant** from the **Cluster** palette.

- This gives you an empty cluster shell.
- You can size the cluster when you drop it.
- Put other constants inside the shell.

**Note:** You cannot place terminals for front panel objects in a cluster constant on the block diagram, nor can you place “special” constants like the Tab or Empty String constant within a block diagram cluster shell.
Modifying a Cluster

- Use the Bundle/Unbundle By Name or the Bundle/Unbundle function to modify an existing cluster.

The terms bundle and cluster are closely related in LabVIEW. Standard functions for clusters can be found in the Functions palette under Programming>Cluster, Class, & Variant subpalette.

Example: You use a bundle function to create a cluster. You can then use an unbundle function to extract the parts of a cluster.

**Bundle** – Forms a cluster containing the given objects in the specified order.

**Bundle by Name** – Updates specific cluster object values (the object must have an owned label).

**Unbundle** – Splits a cluster into each of its individual elements by data type.

**Unbundle by Name** – Returns the cluster elements whose names you specify.

**Note:** You must have an existing cluster wired into the middle terminal of the function to use Bundle By Name.
Clusters – Array vs Cluster

- Clusters differ from arrays in that they are a fixed size.
- Clusters can contain mixed data types; arrays contain only one data type.
- Like an array, a cluster is either a control or an indicator and cannot contain a mixture of controls and indicators.
C. Generating Signal

- Use Simulate Signal Express VI to generate signals.
  - Require minimal wiring because you configure them with dialog boxes

Express VIs are simple and easy to use VIs and are generally configurable through a menu interface. Express VIs are denoted by their blue icon color. The Simulate Signal Express VI can be found under the Express»Input subpalette of the Functions palette.

Most Express VIs accept and/or return the dynamic data type. The dynamic data type appears as a dark blue wire/terminal. Generally, dynamic data types are wired to indicators that can best present the data, such as graphs, charts, or numeric indicators. When connecting the dynamic data to these indicators, an automatic conversion will generally take place to match the indicator to which it is wired, however, in the event that this data needs to be manipulated and operated on, we first need to convert this data to an acceptable data type by using the ‘Convert from Dynamic Data’ Express VI found in the Express»Signal Manipulation subpalette.
Generating Signal (Cont.)

- Use Waveform Generation VIs under 
Programming»Waveform»Analog Waveform»Generation

To make proper use of the Waveform Generation VIs, please refer to the Context Help menu <Ctrl+h>. Below is the context help for Basic Function Generator .vi.

Note that none of the inputs need to be connected to this VI in order to run since none of the input labels are marked in bold in the Context Help. When no inputs are connected to its input terminals, the VI will assume default values for those terminals as specified in the Detailed Help. If you wish to override the default values, data must be wired at its inputs. To ensure that the appropriate data types are connected, consider right-clicking on the input terminal and creating a control or a constant rather than connecting a custom control.
Waveform Data Type

The waveform data type contains 3 pieces of data:

- \( t_0 \) = Start time
- \( dt \) = Time between samples
- \( Y \) = Array of Y magnitudes

The waveform data type carries the data, start time, and \( \Delta t \) of a waveform. You can create waveforms using the Build Waveform function. Many of the VIs and functions you use to acquire or analyze waveforms accept and return the waveform data type by default. When you wire a waveform data type to a waveform graph or chart, the graph or chart automatically plots a waveform based on the data, start time, and \( \Delta x \) of the waveform. When you wire an array of waveform data types to a waveform graph or chart, the graph or chart automatically plots all the waveforms.
Creating Waveforms

- You can create a waveform cluster in two ways:

Build Waveform (absolute time) | Cluster (relative time)

**Build Waveform**
Builds a waveform or modifies an existing waveform with the start time represented as an absolute timestamp. Timestamps are accurate to real-world time and date and are very useful for real-world data recording. Use the “Get Wfm Comps” and “Build Waveform” function from the Programming»Waveform subpalette.

**Bundle**
Builds a waveform or modifies an existing waveform with a relative timestamp. The input to t₀ is a DBL. By building waveforms using the bundle, you can plot data on the negative x-axis (time).
There are 3 main types of data displays which can be used for plotting data. These can be found in the Modern Graph subpalette of the Controls palette.

1. **Waveform Charts:**
   
The waveform chart is a special type of numeric indicator which is used to update a plot one or multiple points at a time. This type of display appends the data wired at its input to any past data set. In order to clear a chart, right-click on the waveform chart in the front panel and select Data Operations » Clear Chart.
Textual Math in LabVIEW

• Integrate existing scripts with LabVIEW for faster development
• Use interactive, easy-to-use, hands-on learning environment
• Develop algorithms, explore mathematical concepts, and analyze results using a single environment
• Choose the most effective syntax, whether graphical or textual within one VI

In LabVIEW 8.6, you have the freedom to choose the most effective syntax for technical computing, whether you are developing algorithms, exploring DSP concepts, or analyzing results. You can instrument your scripts and develop algorithms on the block diagram by leveraging LabVIEW MathScript, a math-oriented textual programming language that is generally compatible with popular .m file script language.
Section II – LabVIEW MathScript

A. LabVIEW MathScript Window
B. MathScript Syntax
  • Loop Structures
  • Conditional Statement
  • Array Manipulation
C. MathScript Node
A. Interactive LabVIEW MathScript Window

- Prototype equations and formulas through the Command Window
- Select a variable to display its data in the preview pane and listen to the result
- Write, save, load, and run .m files using the Script tab
- Share data between the MathScript Node and the MathScript window using global variables

The LabVIEW MathScript Window offers an interactive interface in which you can enter .m file script commands and see immediate results, variables and commands history. The window includes a command-line interface where you can enter commands one-by-one for quick calculations, script debugging or learning. Alternatively, you can enter and execute groups of commands through a script editor window.

As you work, a variable display updates to show the graphical/textual results and a history window tracks your commands. The history view facilitates algorithm development by allowing you to use the clipboard to reuse your previously executed commands.

The LabVIEW MathScript window provides an interactive environment where you can prototype equations and make calculations. The MathScript window and MathScript Node share a common syntax and global variables, making the move from prototype to implementation seamless. The data preview pane provides a convenient way to view variable data as numbers, graphically, or audibly (with sound card support).

**Help for LabVIEW MathScript**

You can access help for the environment using the LabVIEW MathScript interactive environment window. Type `Help` in the command window for an introduction to LabVIEW MathScript help. Typing `Help` followed by a `function` will display help specific to that function.
Components of MathScript Window:

1. **Command Window**
   - Specifies the MathScript command you want LabVIEW to execute. Press the <Shift-Enter> keys to enter multi-line commands. You also can use the up and down arrow keys to display *Command History* items in the *Command Window* which you can edit or execute again.

2. **Output Window**
   - Displays the commands you enter in the *Command Window* and the output that MathScript generates from those commands. Note that the results of a command will not appear on the output window if you end the command line with a semicolon.

3. **Variables Pane**
   - Displays a list of all variables you define and previews variables that you select. This page includes components such as name, dimension, and data type.

4. **Script Pane**
   - Displays the script you create on the Script Editor page.

5. **History Pane**
   - Displays a history of the commands you executed. Double-click an item in the *Command History* list or select the item and press the <Enter> key to execute the command again. Use the <Shift> and <Ctrl> keys to select multiple commands.
B. MathScript Syntax

Common commands

- **Range (Used to autofill array)**

  **Expression**
  
  \[
  \text{start:}[\text{step:}]\text{end}
  \]

  **Example**
  
  \[b = 2:2:20\]

  \(b\) returns the even numbers between 2 and 20.

  If you do not specify a step size, LabVIEW uses a step size of 1.

  You can use the \text{end} function within a range construct to indicate the last element of a row, column, or matrix.

  \[A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}\]

  \(A(3, 2:\text{end})\) returns the elements in the third row of matrix \(A\) starting at the second element and ending at the last element.
B. MathScript Syntax

- **Element-by-Element Operator**

  **Expression**
  
  \[
  A \cdot <\text{operator}> B
  \]

  **Example**
  
  \[
  A = \begin{bmatrix} 1 & 2 & 3 \end{bmatrix}
  \]
  
  \[
  B = A\cdot^2
  \]

  B returns the squared elements of vector A. B = \[1\, 4\, 9\]

  However, \(A^2\) will return an error since without the \(.\) operator, MathScript assumes you are trying to perform a matrix operation.
B. MathScript Syntax

- **Transpose Operator**
  
  **Expression**
  
  \( A' \)
  
  **Example**
  
  \[
  A = [1 2 3] \\
  B = A'
  \]
  
  \( B \) returns the transposed vector \([1; 4; 9] \).

- **Code Comment**
  
  **Expression**
  
  \% Comment
B. MathScript Syntax (Cont.)

- **For Loop**
  
  **Expression**
  
  ```mathscript
  for expression
      statement-list
  end
  ```

  **Example**
  
  ```mathscript
  for i = 1:10
      a = sin(2*pi*i/10)
  end
  ```

- **While Loop**
  
  **Expression**
  
  ```mathscript
  while expression
      statement-list
  end
  ```

  **Example**
  
  ```mathscript
  while i < 10
      a = cos(2*pi*i/10)
      i = i + 1;
  end
  ```

- **If-Else Statement**
  
  **Expression**
  
  ```mathscript
  if expression
      statement-list
  elseif expression
      statement-list
  ...[else
      statement-list]
  end
  ```

  **Example**
  
  ```mathscript
  if b == 1
      c = 3
  else
      c = 4
  end
  ```
C. LabVIEW MathScript Node

• Implement equations and algorithms textually
• Input and output variables created at the border
• Generally compatible with popular .m file script language
• Terminate statements with a semicolon to disable immediate output

Prototype your equations in the interactive LabVIEW MathScript Window.

The LabVIEW MathScript Node enhances LabVIEW by adding a native text-based language for mathematical algorithm implementation in the graphical programming environment. You can open and use the .m file scripts you’ve written and saved from the LabVIEW MathScript window in the LabVIEW MathScript Node. The LabVIEW MathScript Node is located in the Programming⇒Structures subpalette.
Consider the options available within the LabVIEW development environment, which allows a developer to use a hybrid approach when creating code: the combination of the LabVIEW with text-based commands or scripts. These are usually implemented within the LabVIEW development environment through a “node” in which inputs/outputs (variables or constants) are defined to “connect” the node with the graphical programming language. Depending on the algorithm that needs to be implemented, you may want to consider the level of complexity involved in using both LabVIEW and MathScript.

Consider the Fibonacci series

\[
Fib(n) = \begin{cases} 
0 & \text{if } n = 0 \\
1 & \text{if } n = 1 \\
Fib(n-1) + Fib(n-2) & \text{if } n > 1 
\end{cases}
\]
Additional Resources

• MathScript Functions List
  LabVIEW Help»VI and Function Reference»Mathematics
    VIs»LabVIEW MathScript Functions

• Video Tutorial