BE/EE/MedE 189a: Design and construction of biodevices

Justin Bois

Caltech

Winter, 2017

4 Arrays, clusters, and plotting

4.1 Clusters

We saw last time that more than numbers can flow through wires in a VI. We made **error clusters** and we propagated through our VI. I did not carefully define what a **cluster** is. A cluster is a grouping of objects, must like a structure in C or a dictionary in Python.[1] When we make an error cluster, we specify an error code and an error message, and optionally whether the error is a warning and whether or not to show the full chain when the error is encountered, which are updated in an existing error cluster. Conveniently, we can run the entire cluster through wires in the diagram.

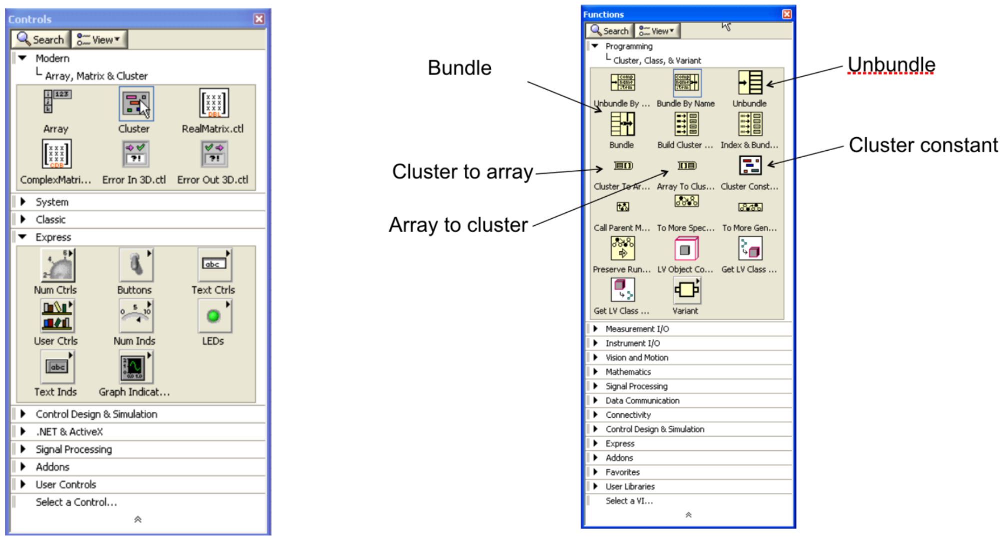

Figure 15: Left, controls for clusters for the front panel. Right, annotated cluster functions from the Functions Palette.

We can make clusters beyond error clusters. To insert a cluster in the front panel of a VI, choose Cluster in the Array, Matrix & Cluster menu of the Controls Palette (see Fig. 15). Once you drag this onto the front panel of your VI, you can drag whatever in controls you want to bundle into this cluster.

In the back panel, we can unpack clusters using the Unbundle object from the Functions palette (see Fig. 15). We can also bundle objects together into a new cluster using the Bundle object in the same palette.

To see an example use of clusters, see the `mass_of_air.vi` sample VI that computes the mass of air given the pressure, volume, and temperature using the ideal gas law.

[1]A LabVIEW cluster is actually more like a namedtuple from the collections module of the Python Standard Library. A LabVIEW cluster is not a hash table like a Python dictionary.

This tutorial is useful to learn about both clusters and arrays.

4.2 Arrays

Arrays are similar to clusters in that they are ordered collections of objects. They differ in that all of the objects must be of the same type. Further, arrays may be multidimensional, not just an ordered one-dimensional array. Because all of the data are of the same type, more operations may be done on arrays.

Arrays are **indexed**, as in other languages. Importantly LabVIEW array indexing starts at zero. You can index an array using that Index Array object in the Functions → Programming → Array palette (see Fig. 16).

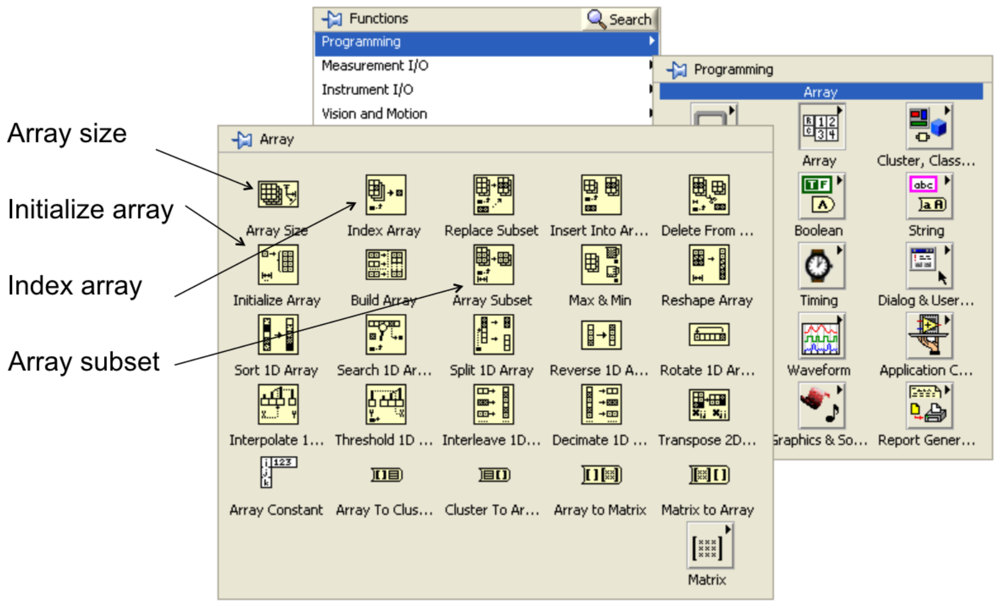

Figure 16: Annotated palette of array functions.

To create an array in the front panel, drag an Array object onto your front panel, as in Fig. 17. You will then need to populate it with a numeric constant or a strong constant to specify the data type of the array. When it appears on your front panel, the array will have an index selector to the left and then the values of the entries of the array to the right. If you want to make a two or more dimensional array, you can right click on the index selector and select Add Dimension. You can do a similar procedure by adding an array constant to a block diagram.

The array function palette (Fig. /16) has lots of options for building and modifying arrays. For an example of using the Insert Into Array function to split an array into two arrays, one containing the nonnegative values in the original and the other containing the negative ones, see `pos_neg_array.vi`.

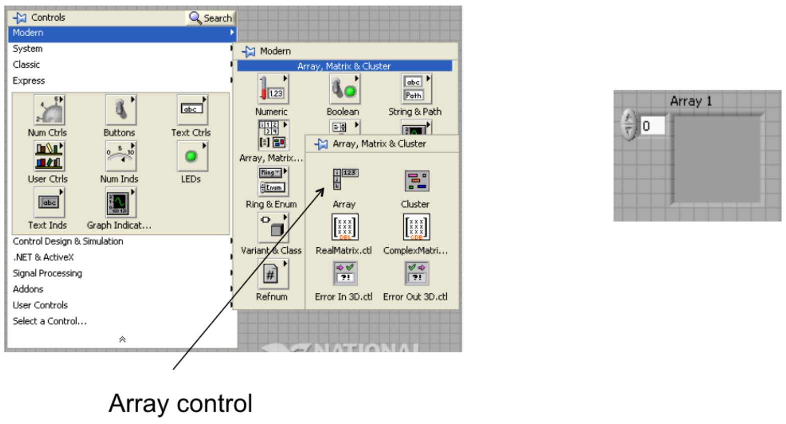

Figure 17: How to add an array control.

4.2.1 Operations on arrays

Arithmetic operations on arrays are elementwise. For example, if you add two arrays by wiring them into a + operator, the result is an array where like-indexed input values are added together. If you add an array and a scalar, the scalar value is added to each element of the array, with the result being an array. Similar results happen for other operations.

What happens when you add two arrays of different lengths? The result is an array equal to the length of the shorter array. I personally think this is a terrible idea. It just invites bugs. Therefore, *Beware*. You might see unexpected behavior in your VIs because of this. You might want to write your own VI that does error checking for array operations with arrays of different sizes, such as in the example `add_arrays_with_error.vi`.

4.2.2 Matrices

What if instead of doing elementwise multiplication of two 2D arrays (where one is $m \times p$ and the other is $p \times n$), I want to do a matrix multiplication? We can still use a two arrays and use the A $\times$ B function in the linear algebra palette (see Fig. 18).

Alternatively, I could convert the arrays into a new data type called a **matrix**. The $\times$ operator then acts like matrix multiplication on these objects. In my view, this is another bad idea. Use of the A $\times$ B function is unambiguous; it is completely clear that you are attempting matrix multiplication. Matrices and arrays are stored in the same way for LabVIEW, and there is no performance boost for using matrices versus arrays. I am of the opinion that you should never use matrices. The added flexibility is not helpful; it just opens you up to hard-to-find bugs. You can do all the matrix calculations you need using arrays, and matrices were not even added to LabVIEW until version 9.0 (I think).

4.3 Polymorphism in LabVIEW

Polymorphism is generally the idea that a given interface, say to a function, accepts inputs of different types and then does operations on them according to the type. One way this is achieved in Python, for example, is by operator overloading.

```
3 + 4 = 7

'base' + 'ball' = 'baseball'
```

The Python + operator can work on data of different types. LabVIEW also features polymorphism. The LabVIEW + operator can take different data types, as we have seen; it can take scalars or arrays, or a mix (or other types as well, including clusters; check out the sample VI `array_cluster_polymorphism.vi` to see how array operations can work on clusters). Usually, the choice of operation based on input type is intuitive, but not always, as we saw in the example of two arrays of different lengths. (I would expect an error to be raised.)

I bring up this polymorphism because it is at once convenient and dangerous. Your code might work in unexpected ways depending on the types of inputs you give. *Caveat emptor*.

4.4 Making *x*-*y* plots

LabVIEW's plotting applications are primarily built for real-time monitoring of signals. After all, you are creating virtual instruments. As a result, its plotting capabilities are limited, but nonetheless useful.

Plots are shown in the front panel, usually as indicators. To make a plot of *x*-*y* data, I usually use the Ex XY Graph VI from the Graph palette of the Controls palette. Dragging this onto your front panel will create the graph, along with an object on the back panel. By right clicking on the graph, you can set properties, such as how the data are represented (points versus lines, etc.).

To see an example of the construction of an *x*-*y* plot of one of my favorite data sets (relative levels of Bicoid protein across a *Drosophila* embryo from the classic Driever and Nüsslein-Volhard paper), see `plot_bcd_profile.vi`. Note that you wire in an array of *x*-values and *y*-values to specify the data points.

4.5 Waveform charts and graphs

More typically, you will want to monitor real-time signals that have evenly sampled points. To do this, you can use Waveform charts and graphs. To make a waveform chart or graph, drag the Waveform Chart or Waveform Graph onto your front panel. Once in the front panel, you can change the xaxis labels, as well as the plot title, which is to the top left. At the top right is the plot legend, which you can right click on to alter the appearance of the plot.

In the block diagram, a wave *chart* takes as input a scalar value. The plot is constructed as the value coming into the wave chart changes. This is in contrast to a wave *graph*, which takes an array of data points as input. Alternatively, you can construct a cluster that has the starting time, the time between samples, and the array containing the signal. This will scale the x (time) axis appropriately.

To see examples of a waveform chart and a waveform graph, look at `wf_chart_and_graph.vi`. Note that the waveform chart is inside the for loop, while the waveform graph is outside. An array is automatically created within the for loop, which is outputted to the waveform graph.

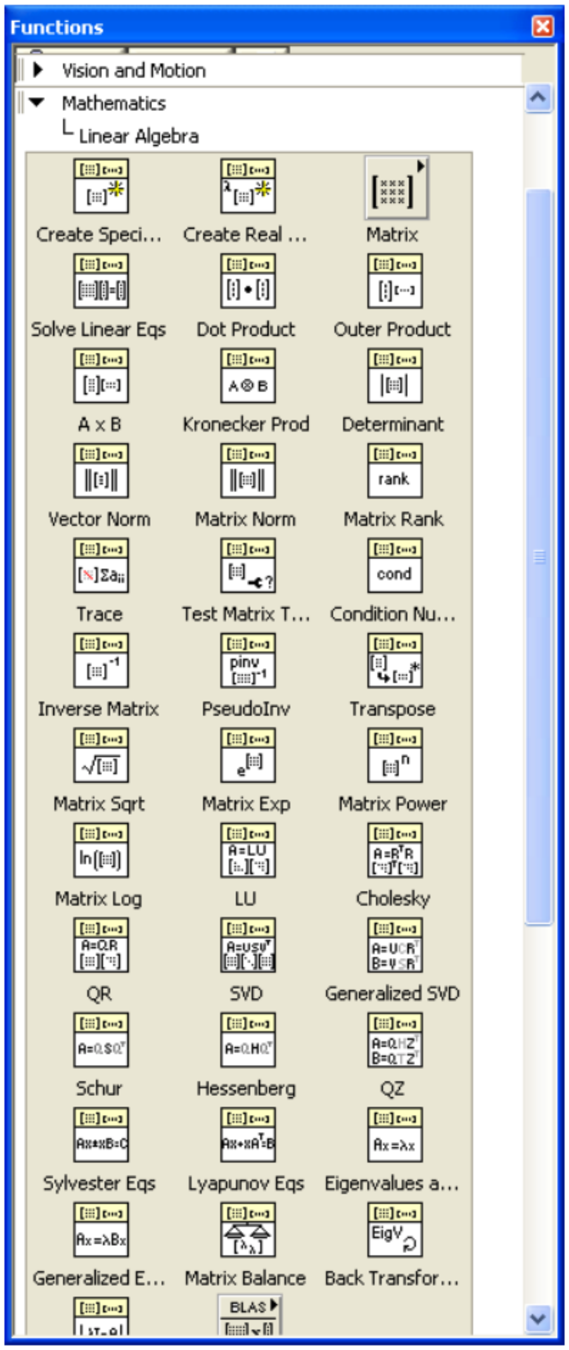

Figure 18: The linear algebra palette.