Chapter 2 Data in DataScope

Chapter Overview

DataScope works with a variety of two-dimensional dataset formats. This chapter defines the requirements for the regular 2-D, stretch 2-D, and polar forms which DataScope works with. It also explains how to prepare your data in a format compatible with DataScope.

Regular 2-D Datasets

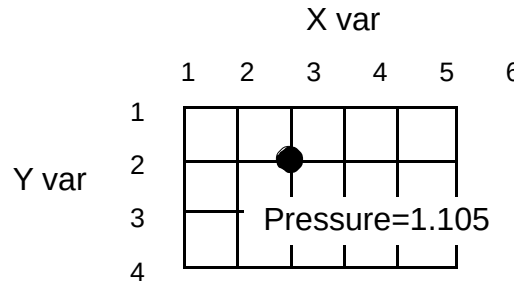
DataScope is designed to work with two-dimensional arrays of floating point numbers. Once a 2-D array is in memory, the data is displayed as printed numbers in a spreadsheet fashion that allows you to use DataScope's other commands on this data. DataScope's working representation of a dataset has the following properties:

- Thirty-two bit floating point numbers are used for all values at full precision.
- Each array can be defined in terms of three variables: two independent variables which correspond to the row and column labels, and a dependent variable, which has a value at every point in the array.
- The names of the dependent variable and the independent row and column variables are stored, and used when appropriate. The dependent variable name also functions as the name of the window in which the data appears, and is also used in notebook calculations.
- DataScope stores the size of the grid, the number of rows (nrows) and the number of columns (ncols).
- A vector exists for each of the independent variables: one has an entry for each column and defines the values along the width of the grid; the other has an entry for each row, and defines the values along the height of the grid.
- The values of the dependent variable are located inside the grid, one at each grid point. The number of values is therefore nrows*ncols.
- A format field, as a FORTRAN specification (e.g., F8.4, E11.5), is maintained for the dependent variable and for the independent variables. When printed as text, the floating point numbers appear in this format.

In the example given in Figure 2.1, the *ncols* is 6 and the *nrows* is 4. The grid, therefore, has 24 data points. The column labels vector contains {1,2,3,4,5,6} and is named "X var." The row labels vector contains {1,2,3,4} and is named "Y var." The dependent variable is named Pressure and it must have 24 values, one for each point in

the grid. The example highlights the value at (row = 2, column = 3) where the Pressure has the value 1.105.

Figure 2.1 Example Regular 2-D Dataset



Polar-Oriented Data

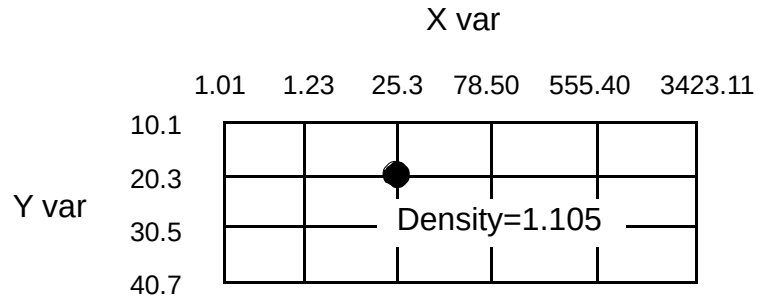
The only difference for polar-oriented data is that the value for each row should be treated as a radius value; and the value for each column becomes a theta angle which should always have values in the range 0 to 2*PI. Polar data can, in fact, be treated as both rectangular *and* polar, a feature which can be helpful.

Stretch Grids

In the next example, Figure 2.2, the nrows and ncols remain 4 and 6, but the values which are associated with the independent variables are no longer simple integers {1,2,3 . . .}. Instead, they define a scale which does not have to be linear. In this example, the row labels are almost linear {10.1, 20.3, 30.5, 40.7}, whereas the column headings are almost logarithmic {1.01, 1.23, 25.3, 78.50, 555.40, 3423.11}. In fact, the row and column values are determined by the scientific data. They may have any set of increasing values.

The indicated value in Figure 2.2 is still at grid point (2,3), but this is shown on the screen as the point where row = 20.3 and
column = 25.3. The value of Density at this point is 1.105. DataScope does not try to interpolate the values for you in the text window.

Figure 2.2 Example Dataset with Non-linear Row and Column Values



Current Restrictions and Caveats

There is one important exception to the above description of the values of independent variables. For this version, the independent variables must increase in value from left to right for columns and from top to bottom for rows. Some of the calculations rely upon this property of the independent variables to work correctly. The independent variables may be positive or negative, and may be any numbers which can be represented in IEEE 32-bit floating point format.

Also note that for irregularly spaced grids with non-linear independent variable values, the selection of a box on the interpolated image window does *not* provide the correct selection in the text number window. The synchronizing capability between the text window and the interpolated image window selections uses a linear fit which does not work unless the independent variables have a linear distribution.

Loading and Saving HDF Files

HDF files are data files which are created using NCSA's Hierarchical Data Format for file storage. HDF files can contain all of the data for DataScope's two independent and one dependent variables, along with the auxiliary data for the dataset. This might include the nrows, ncols, printing formats, variable names, user notes, and associated images. All of these are automatically retrieved when you use the Open command from the File menu.

The Open command from the File menu presents the standard file dialog and allows you to choose a filename to be read in. It checks to verify that you have chosen an NCSA HDF file and presents an error message if you have not. The file is then processed and the floating point dataset is read into memory along with any and all accompanying information. DataScope creates a text window for the display of the data. If there are images or notes in the file, DataScope creates and displays the appropriate windows. DataScope ignores information from the HDF file that is beyond its scope to interpret.

The Save command from the File menu saves the current dataset in the HDF file format. DataScope uses the data associated with the frontmost window at the time the Save or Save As command is selected. The 32-bit floating point data is written into the HDF file in binary format as a Scientific Data Set (SDS) which includes its row and column independent variable information and formats. The details for storage of an SDS are documented as part of the HDF library package.

The currently selected dataset in the frontmost window may have an associated notebook window and image windows which were generated from the data. In addition to the data for the array, the Save command finds the notebook and most recent image of each type which are linked to the current dataset and saves them into the

same HDF file. When this file is reloaded, the Open command recreates each of these windows with the appropriate contents for each.

Loading Text Files

If you do not have access to HDF libraries on the computer where you generate your data, you can store your data in a text file instead. By choosing the Load Text command from the File menu, you can cause a text file in a particular DataScope format to be read in and used to create a text window. DataScope can then save the data as an HDF file.

Files should be created according to the following criteria:

- Real numbers can be in any C-standard text format for floating point numbers.
- Integers can have any field width as long as they are separated by spaces or tabs.
- Maximum and minimum values define the range of number values in the region of interest. Outliers are automatically ignored during image generation, but remain in the dataset unchanged.
- Arrays should be ordered by rows, left to right, top to bottom, depending upon how you want images to be generated. This also happens to be the default order which FORTRAN uses to print out arrays.

The format will look like this:

```
nrows  ncols
max_value  min_value
row1 row2 row3 row4........
col1 col2 col3 col4........
data1 data2 data3 ....
.....
```

The following example shows an array in FORTRAN, declared with dimensions of 5 rows of 10 columns. Although the example shows a situation where the entire row fits on one line of the file, the data values may be split among any number of lines.

For an array which is declared in FORTRAN, for example:

```
REAL*4  MYDATA(5,10)
```

An example data file for this array might be:

```
5  10
9.9e0  0.0e0
1.1  2.3  3.4  5.6  7.8
1 2 3 4 5 6 7 8 9 10
0.1  0.2  0.3  0.4  0.5  0.6  0.7  0.8  0.9  1.0
1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  1.0
3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9  1.0
6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9  1.0
9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8  9.9  1.0
```