Using IDL to Manipulate and Visualize Scientific Data

IDL Introduction*

Table of Contents

Introduction

IDL (the Interactive Data Language) is a general purpose scientific computing package, sold by Research Systems Inc. (RSI), which provides a suite of mathematical functions, data analysis tools, as well as some scientific visualization and animation tools. It uses a command line interface and is fairly robust. It is fairly easy to develop applications using point-and-click widgets, as well. It does not have as large a market share as something like Matlab, but IDL is used extensively in the Fusion community.

This tutorial is organized as a set of simple examples with explanations. It is only a glimpse at what you can do with IDL. Once you start using it, you will want to look at the online and/or hardcopy documentation, and explore.

The most effective way for you to go through this tutorial is by running IDL in a separate window, and trying out the commands and programs as you read the tutorial. Experiment with variations on these commands. Note what happens when you make errors. Alternately, you may prefer using the IDL Basics manual over this tutorial.

Getting started

First read about How to Setup IDL at PPPL. This will tell you on which machines IDL is running, how to set up your environment, how to set your display, and how to find documentation.

A few basic hints that will help you get started with the command line interface.

Type a question mark "?" for online help if you are running X. If you are running Versaterm on VMS, type "$help idl" at the IDL prompt.

Try typing the following four commands at the IDL prompt:

```
IDL> a = 5
IDL> print, a
IDL> a = [2, 3]
IDL> print, a
```

Observe that commands are followed by a comma, before the parameter list.

To repeat a command, you can go up and down through previous commands using the

arrow keys. When you reach the command you want to repeat, hit return (this is like VAX/VMS and the Unix tcsh shell).

IDL programs can be stopped using control-C. (Hold down the control key and hit the letter c).

IDL can be aborted immediately using control-\. (All variables are lost and the state of open files will be uncertain).

Programs and batch mode

IDL can be run by typing commands interactively, by creating programs interactively, by reading programs in from the command line, or it can be run in batch mode.

When you type commands on the command line, each line is executed immediately when you hit the return key. (It is possible to carry over to the next line using a dollar sign "$" at the end of the line).

Batch mode

Running in batch mode is similar, except the commands come from a file. You precede the file name with the at-sign "@". Using your favorite editor, create a file called batch_two_prints (in the directory from which IDL is running) that contains the four lines

```
a = 5
print, a
a = [2, 3]
print, a
```

Type the following at the IDL prompt

```
IDL> @batch_two_prints
```

This will execute the four commands exactly as if you had typed them. This can also be done directly from your Unix shell, by typing

```
% idl batch_two_prints
```

Programs

When typing interactively and running in batch mode, each line is executed immediately. A problem arises with control statements that span lines. Consider

the following simple computation of the factorial function. As a batch file it would look like:

```
f = 1
for k=1,6 do begin $
    f = k * f & $
    print, f & $
end
```

The commands must be separated by ampersands "&", and the lines must be continued. The entire loop must essentially be on one line, since each line is executed as soon as it is encountered. Imagine nested loops with long calculations inside.

As a program it is somewhat simpler:

```
f = 1
for k=1,6 do begin
    f = k * f
    print, f
endfor
end
```

Files can be created while you experiment with syntax and your algorithm by:

```
IDL> journal,'mycode.pro'
IDL> (many idl commands)
IDL> journal
```

Then, edit out the unwanted lines in the file mycode.pro. Add a STOP and END statement. Then .run the file.

Conversely, a good way to debug procedures or functions found in files is to copy lines from them and paste them into an IDL session.

For more information on creating and running programs, including other commands, see Chapter 2 of the User's Guide.

Variables and arithmetic

You can explicitly "type" variables, or not. IDL is very good about adapting variables to the right type, even within procedures. See chapter 4 of the User's Guide for more information on type conversion, etc. Try some of the following to get used to the syntax.

The simplest thing to work with is scalars.

```
IDL> y = 2.5
IDL> z = x + y
IDL> w = x^y + sin(z)
IDL> print, x, y, z, w
       3      2.50000      5.50000      14.8829
```

Square braces are used to define vectors (1-dimensional arrays):

```
IDL> v1 = [1, 2, 0]
IDL> v2 = [1, 0, 0]
IDL> print, "v1 = ", v1
v1 =        1       2       0
IDL> print, "v2 = ", v2
v2 =        1       0       0
```

Vectors can be componentwise added, multiplied, etc.:

```
IDL> v3 = v1 + v2
IDL> print, "v3 = v1 + v2 = ", v3
v3 = v1 + v2 =        2       2       0
IDL> print, "v1 * v2 = ", v1 * v2
v1 * v2 =        1       0       0
IDL> print, "v1 * sin(v3) = ", v1 * sin(v3)
v1 * sin(v3) =      0.909297      1.81859      0.00000
```

There are other useful operators, such as min and max:

```
IDL> min1 = min(v1)
IDL> max1 = max(v1)
IDL> print, "min(v1), max(v1) = ", min1, max1
min(v1), max(v1) =        0       2
```

Scalars and arrays can be allocated with specific types. Scalar examples:

```
IDL> x = float(1.3)
IDL> sx = fix(x)
IDL> lx = long(x)
IDL> bx = byte(x)
IDL> dx = double(x)
IDL> cx = complex(x)
IDL> print, x, sx, lx, bx, dx, cx
       1.30000       1           1   1       1.3000000
(      1.30000,      0.00000)
```

Array examples:

```
IDL> a = fltarr(5)
IDL> for i=0, 4 do $
IDL>     a(i) = 2*i
IDL> b = complex(a)
IDL> print, "b = ", b
b = (      0.00000,      0.00000)(      2.00000,      0.00000)
```

```
(      4.00000,      0.00000)(      6.00000,      0.00000)
(      8.00000,      0.00000)
```

Matrices

A matrix (a 2-dimensional array) may be defined algorithmically:

```
IDL> A = dblarr(2, 4)
IDL> for i = 0, 1 do begin $
IDL>     for j = 0, 3 do begin $
IDL>         a(i, j) = 10 * i + j
IDL> print, A
        0.0000000       10.000000
        1.0000000       11.000000
        2.0000000       12.000000
        3.0000000       13.000000
```

Note that as it is printed, the first index corresponds to the column, and the second index to the row. Another way to think of it is that the way the data is stored, the first index varies fastest, and the second varies the slowest. This agrees with the way the data is printed.

The WHERE function can be extremely useful. Play around with it. (Continuing from the above example:)

```
IDL> print, WHERE(A GT 10 AND A LT 13)
   3           5
```

A matrix may be constructed explicitly from vectors:

```
IDL> v1 = [1, 2, 0]
IDL> v2 = [1, 0, 0]
IDL> v3 = [4, 5, 6]
IDL> A = [[v1], [v2], [v3]]
IDL> print, A
    1       2       0
    1       0       0
    4       5       6
```

Create the transpose:

```
IDL> Atrans = transpose(A)
IDL> print, Atrans
    1       1       4
    2       0       5
    0       0       6
```

Take the determinant:

```
IDL> d = determ(float(A))
% Compiled module: DETERM.
IDL> print, d
     -12.0000
```

Invert:

```
IDL> Ainv = invert(A)
IDL> print, Ainv
       0.00000      1.00000      0.00000
      0.500000    -0.500000      0.00000
     -0.416667    -0.250000     0.166667
```

Multiply vectors by matrices:

```
IDL> v = [1, 2, 3]
IDL> print, A
       1       2       0
       1       0       0
       4       5       6
IDL> print, v
       1       2       3
IDL> print, A ## v
           5
           1
          32
IDL> print, v ## A
          15          17          18
```

You can solve a linear system Ax = b for x by Cramer's rule (the cramer function expects float or double inputs, requiring an explicit type conversion):

```
IDL> b = float([1, 2, 16])
IDL> A = float(A)
IDL> x = cramer(A, b)
IDL> print, x
      2.00000    -0.500000      1.75000
```

Plots of Y vs. X

Two examples of plots of a function of one variable are listed below. Also check out the examples with MDS access.

The program **fit_y.pro** reads in a set of 10 y values. It creates 10 x values from 0 to 9 suing the "findgen" command. It then fits a cubic to the data, and plots the original data as points, and the fitted function as a curve.

```
; File: fit_y.pro -  Author: Erik Brisson
N = 10
```

```
y = fltarr(N)
openr, 1, 'ex_y.dat'
readf, 1, y
close, 1
print, 'print y'
print, y
print, 'another way to print y'
for i=0, N-1 do begin
    print, y(i)
endfor
plot, psym=1, y
x =findgen(N)
ypoly = poly_fit(x, y, 3)
print, 'coefficients of third degree fit'
print, ypoly
yapprox = ypoly(0) + ypoly(1)*x + ypoly(2)*x^2 +
ypoly(3)*x^3
print, 'values of fit at corresponding x values' print,
yapprox oplot, yapprox
end
```

The program **fit_xy.pro** reads in a set of 10 (x,y) pairs, into a 2 x 10 array. It separates them into two 1-dimensional arrays, and fits a cubic to it, and plots this as in the preceding example.

```
; File: fit_xy.pro ; Author: Erik Brisson
N = 10
xy = fltarr(2,N)
openr, 1, 'ex_xy.dat'
readf, 1, xy
close, 1
x = xy(0,*)
y = xy(1,*)
print, 'x'
print, x
print, 'y'
print, y
plot, psym=1, x, y
ypoly = poly_fit(x, y, 3)
yapprox = ypoly(0) + ypoly(1)*x + ypoly(2)*x^2 + ypoly(3)*x^3
print, 'yapprox'
print, yapprox
oplot, x, yapprox
end
```

Surface plots

Also check out the contour examples with MDS access.

An example of drawing a surface plot directly, rendered as a wire mesh,

```
a = findgen(35)
b = 45 -findgen(45)
c = a # b
window, 0, retain=2, title='surf_wire',xsize=500, ysize=500
SURFACE, c
```

IDL provides an interactive viewer for surface plots, called xsurface.

```
XSURFACE,c
```

An example of drawing a surface plot directly, rendered as a shaded surface. ; File: shade_surf.pro - Author: Erik Brisson

```
SHADE_SURF, c
```

These plots can be combined with various contour plots in various ways.

Producing surface plots from scattered data is demonstrated in hydro.pro. This is data in which the (x,y) locations for which we have the function evaluated do not lie on a regular gird. To make a surface plot, IDL needs to have the function evaluated on a regular rectangular grid. There are two steps involved. The first is to form a triangulation using the input (x,y) points to use for interpolation, and the second is to produce a mesh from that interpolation. Another example, at scv.bu.edu/SCV/Tutorials/IDL/examples/idl/pro/triangulate.pro, shows this process and renders the result as a wire mesh surface plot. http://scv.bu.edu/SCV/Tutorials/IDL/examples/idl/pro/tri_shade.pro, does the same thing, and in addition, draws a shaded surface plot, using two different kinds of shading.

Animation

One method of producing animation is to create a sequence of images and then display them in order. This is quite easy using IDL.

```
; File: making_waves.pro - Author: Erik Brisson
pro making_waves
a = fltarr(256,3,63)
for i=0,255 do for j=0,2 do for k=0,62 do $
    a(i,j,k) = sin(float(i-k)/10.)/exp((float(i)/200.))
frames=bytarr(400,300,63)
window,retain=2,/free,title='making waves',xsize=400,ysize=300
for k=0,62 do begin $
shade_surf, a(*,*,k) & $
frames(0,0,k) = tvrd() & $
end
movie, frames, order=0
end
```

Hardcopy

You can save your plots and other images by rendering them to a postscript file instead of to an X window. Basically, you set your plotting area to be a file, then change the graphics device to be a postscript device, and close that device when you are done:

```
IDL> set_plot,'PS'
IDL> device, filename='your_filename.ps'
IDL> ...
IDL> ... plot some stuff ...
IDL> ...
IDL> device, /close
IDL> set_plot,'X'
```

Alternately, the local routines setup_ps and setup_x can span the plot commands you want to send to a file when using x-windows. When using Versaterm on VMS at PPPL, the SGLIB graphing system can be used, or the standard printing of Tektronix files from Versaterm.

Gotchas

1. IDL leaves you in the procedure when an error is detected (unless the ONERROR routine is called within the procedure). Enter RETALL to return to the top-level.
2. Array indices are zero based, e.g.,

   ```
   a = BYTARR(100)
   FOR i=0,99 DO a(i) = i
   ```

3. If you don't use LONG(0), the maximum value is 64K, so:

   ```
   FOR i=LONG(0),100000 DO sum = sum+i
   ```

   will give an error.
4. It may not be obvious how to read files written by other programs. There are user routines to read columns of ASCII numbers.
5. > and < are not what they may seem.

   ```
   a = 5 < 3                ; sets a to 3 (the lesser value)
   a = WHERE (array < 2)    ; sets a = array (if first element is < 2
   ```

```
a = WHERE (array LT 2)  ; is probably what you want
```

Why do my graphics get erased?

When a window gets covered, then uncovered, someone has to keep a copy of the obscured part of the image. You may or may not want to have all the images saved when they are obscured, by reasons of speed and memory.

In any case, this topic is called "backing store". It can be done by IDL, done by the windowing system, or not done. By default, the X window system does not have backing store turned on.

In IDL, there is a keyword RETAIN, for specifying which kind of backing store to use.

- RETAIN = 0 implies no backing store,
- RETAIN = 1 IDL asks the window system to do it,
- RETAIN = 2 IDL does it

This may be done on a per window basis, e.g.,

```
window,0,retain=2,xsize=500,ysize=500
```

Backing store will now be maintained for this window by IDL.

How to find out more about IDL.

There is an excellent **demo** supplied with idl, which shows many of the advanced things IDL can do. Just type "demo" at the IDL prompt.

You may also want to look at the IDL supplied examples in **/usr/local/rsi/idl/examples** on Unix or IDL_DIR:[EXAMPLES] on VMS.

You should also visit these valuable IDL sites, especially the ones that let you search for IDL routines written by others (no such search exists for fusion, or PPPL-specific software, but it should).

Other examples are in the Class2 Directory. For a simple, but useful example of a widget for plotting MDS signals, see mdsw_noch.pro

---

Return to the NSTX Software Page

*Borrowed heavily from http://scv.bu.edu/SCV/Tutorials/IDL/idl_webtut.html

http://nstx.pppl.gov/nstx/Software/idl_intro.html
Edited for NSTX: 01-Feb-1999
by: Bill Davis

)