PHYS-4007/5007: Computational Physics
Course Lecture Notes
Appendix D

Dr. Donald G. Luttermoser
East Tennessee State University

Version 7.0

Abstract

These class notes are designed for use of the instructor and students of the course PHYS-4007/5007: Computational Physics I taught by Dr. Donald Luttermoser at East Tennessee State University.

Appendix D. Scientific Computing Using IDL

A. Introduction.

1. One of the most important aspects of scientific research is the ability to present one's work to the scientific community and to the public $\Longrightarrow$ one of the best way to do this is through good **graphics**.

2. Many programming languages are available that have graphics capabilities. In science, the most commonly used are Mathematica, Matlab, Maple, Origin, Mongo, Supermongo, and IDL.

3. For this Appendix of the notes, we will use **IDL (Interactive Data Language)** as our prototype graphics language. See §II.D in these notes for an overview of IDL and a discussion on the history of IDL.

B. IDL Highlights.

1. IDL is a mature packaged evolving through years of use by scientists.

a) If users are at all comfortable with programming, they will be able to perform calculations and produce graphics without many of the *irrelevant* aspects of programming.

b) Graphic output quality is very good and the annotation capabilities are better than some other software packages.

c) IDL is *case* *insensitive* (like Fortran), 'HELP', 'Help', and 'help' all mean the same thing.

2. Assuming IDL has been installed on a computer, one starts IDL on a Linux machine by opening a terminal window and typing: idl at the Unix prompt.

3. To run IDL on the Microsoft machines, double click on the IDL icon. Once the GUI is created, one can access the command prompt at the bottom of the GUI.

4. Using IDL. At the IDL prompt you can type commands like:

```
IDL> orig = sin((findgen(200)/35)^2.5)
IDL> plot, orig
IDL> exit
```

or run an interactive demo program

```
IDL> demo
```

5. Additional Help/Documentation.

a) Help from within IDL under Linux: Type a question mark (?) at the IDL prompt. This will open a web browser window with a Table of Contents allowing point-and-click access to various topics.

b) Help from within the IDL Graphical User Interface (GUI) under Windows. Typing ? at the IDL prompt will produce a window containing a search box. Just enter the command name for which you need help. Or just select the Help icon at the top of the GUI.

c) Note that the command HELP exists, but only gives information on the current IDL session and not on the commands.

C. A Tutorial of IDL.

1. Communicating with IDL is handled differently in the Microsoft world than it is in the Unix/Linux world. **Note that IDL must be installed on a machine in order for these steps to work.**

a) On a Microsoft Windows PC, double click the IDL icon that is on the background screen. This will bring up the IDL GUI interface.

b) On a Linux machine, there are two ways to start IDL:

i) Type 'idlde' at the Linux command prompt. This will bring up an IDL GUI similar to what is on the PC.

ii) Or, type 'idl' at the Linux command prompt. This will place you in the standard IDL Command Prompt mode.

c) The discussion here requires you to enter commands at the IDL prompt (which you will see in "Command Mode"). In "GUI Mode," enter commands in the input box just to the right of the 'IDL>' icon at the bottom of the GUI.

2. Any time something is not clear, don't hesitate to use the Help utilities described on the previous page. In the examples below, the IDL> that is listed is the IDL prompt — you do type this, IDL presents this to you automatically.

3. This tutorial assumes that you are working in a directory named idl in your login directory if you are on a Linux machine. To start IDL, issue the idl command at the Unix/Linux prompt.

4. When working in a Microsoft Windows environment, the first time you run IDL on a machine, you may be asked to select a WorkSpace directory. You should choose a directory for which you have permission to write files. You can check to see what directory you are currently in by typing:

```
IDL> cd, current=mydir
IDL> print, mydir
```

make sure you are in the directory you should be working in.

5. It should be kept in mind that this tutorial is only a glimpse at what you can do with IDL. Once you start using it, you will want to look at the online, hardcopy documentation, and/or the optional textbook for this course , Practical IDL Programming by Gumley, and explore.

6. Getting started. Try typing the following four commands at the IDL prompt:

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

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

b) To repeat a command, you can go up and down through previous commands using the arrow keys (in both Unix and the Microsoft Windows GUI). When you reach the command you want to repeat, hit <return> (this is similar to the keypad editing environment of the Unix tcsh and bash shells).

c) IDL programs can be stopped using <Ctrl>–C. (Hold down the control key and hit the letter c).

d) IDL can be aborted immediately using <Ctrl>–\. (All variables are lost and the state of open files will be uncertain).

7. 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.

8. 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).

9. Programs.

a) Here we will be making simple IDL programs that you can save and run at any time in the future.

i) Under Linux, start an emacs session in either a separate terminal window (assuming this window is pointing to your working directory) or in the IDL terminal by typing

```
IDL> $emacs myfile.pro &
```

where the '$' tells IDL this is an operating system command and the '&' tells the operating system to put this in background giving you back the IDL> prompt.

ii) Under the IDL GUI (in Windows), select the 'File' pulldown menu at the top left of the IDL GUI and

select 'New IDL Source File' – this will open an editing window in the IDL GUI.

b) Enter the following commands into your program – make sure the last line is end.

```
a = 25
b = 3
c = a * b
print, a, b, c
end
```

c) In Linux, save your program in the appropriate directory, then compile and execute your program with

```
IDL> .run myfile
```

which should produce the output

```
% Compiled module: $MAIN$.
      25       3      75
```

d) In Windows, don't worry about saving this program and just compile and execute your program by clicking the 'green arrow' button near the top of the GUI. This should produce the output

```
% Compiled module: $MAIN$.
      25       3      75
```

10. Variables and arithmetic.

a) You can explicitly type variables, or not. See Type Conversion of the *API Reference Guide/Functional List of IDL Routines/Statements* in the Help utility for information on type conversions.

b) 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
```

c) 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
```

d) Vectors can be component-wise 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
```

e) 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
```

f) 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)
```

g) 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)
```

h) Note that in versions of IDL earlier than Version 5.0, array variables used parentheses '( )' notation instead of square-bracket '[ ]' notation. As such, in the above program, the "for"-loop would have looked like

```
IDL>     a(i) = 2*i
```

i) IDL Version 5.0 and beyond will accept both the '( )' and the '[ ]' notation for array elements. You should always follow the square bracket notation for arrays in this course.

11. Matrices.

a) A matrix (which is 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 a[i, j] = 10 * i + j
IDL> endfor
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.

b) 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
```

c) Create the transpose:

```
IDL> Atrans = transpose(A)
IDL> print, Atrans
   1     1     4
```

```
   2     0     5
   0     0     6
```

d) Take the determinant:

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

e) 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
```

f) 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
```

g) 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
```

12. Plots of Y vs. X. Check out the PLOT and the OPLOT commands in file sampleplot.pro to plot and overplot Y vs. X data. This file can be downloaded from the Course Web Page. Please note that this is a fairly robust routine that I wrote – plotting can be quite easy to do in practice.

13. Surface plots.

a) IDL provides an interactive viewer for surface plots, called XSURFACE. The non-interactive procedure is called SURFACE.

b) To make a surface plot, IDL needs to have the function evaluated on a regular rectangular grid. There are typically 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.

14. Animation. One method of producing animation is to create a sequence of images and then display them in order. For line plots, you keep on using OPLOT commands by first *undrawing* the line by setting COLOR=0b in the OPLOT command, then by oplotting it with the normal color and continuing this cycle until the animation is complete.

15. Hardcopy. You can save your plots and other images by rendering them to a postscript file instead of to an X window or Microsoft Windows. An example is given in sampleplot.pro which can be downloaded from the Course Web Page. 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
```

16. Why do my graphics get erased?

a) 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.

b) 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 (however, Microsoft Windows does).

c) 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. Note that RETAIN = 2 is the default in the current versions of IDL.

D. Overview of Plotting Procedure sampleplot.pro.

1. The previously cited procedure (sampleplot.pro) demonstrates the power that IDL provides you in generating quality publishable graphics. This code, though a bit complicated, gives an example on how to write good IDL code.

2. In this procedure, the embedded procedure called SELHCWIDG brings use a GUI widget that allows the user to select both a "hardcopy" of the plot and/or an "image" file. The possible output hardcopy options are

a) **Normal postscript file**: These files can be sent directly to a postscript printer using the Unix/Linux lpr command. The created postscript file will have a file name suffix of '.ps' at the end.

b) **Encapsulated postscript file**: These files cannot be sent directly to a postscript printer, but instead, are designed to be imported into a word processing file such as a LaTeX file. The technique for doing this will be described in the next section of the notes. The created file has '.eps' as the file name suffix.

c) **Printer**: This option is only available on the Microsoft version of the code. Instead of making a hardcopy file that can be sent to a printer, IDL sends the plot to the default printer directly without creating a file.

d) **Terminal Only**: The plot is only drawn to the terminal screen and no hardcopy is generated. (This is the default option.)

3. The user of this plotting procedure, through the embedded SELHCWIDG procedure, can make image files of the following types (I recommend using the JPEG format since this format is widely compatible with a large number of software packages):

a) **JPG**: A JPEG image file which can be imported into a LaTeX file.

b) **BMP**: A Microsoft bitmap image file which can be imported into a LaTeX file.

c) **PNG**: An image file type designed for web use. These image files cannot be imported into a LaTeX file. This format is only recognized in the Microsoft version of IDL.

d) **TIFF**: Short for Tagged Image Format Files, TIFF is a popular high color-depth image protocol in the industrial graphics world. TIFF files tend to be much larger than JPEG or BMP files. As a result, these types of images are not widely used in the physical sciences.

e) **None**: No image file is generated. (This is the default option.)

4. One can also call this procedure from with another procedure and select the output type through the various keyword settings in the procedure call. If using the keywords settings, one can turn off the SELHCWIDG GUI widget input method by setting the \NOGUI in the procedure call.