

## ASTR 288C – Lecture 8

Tuesday, 26 October 2009

### Data Analysis III: Plotting

#### 1) Interactive Data Language (IDL)

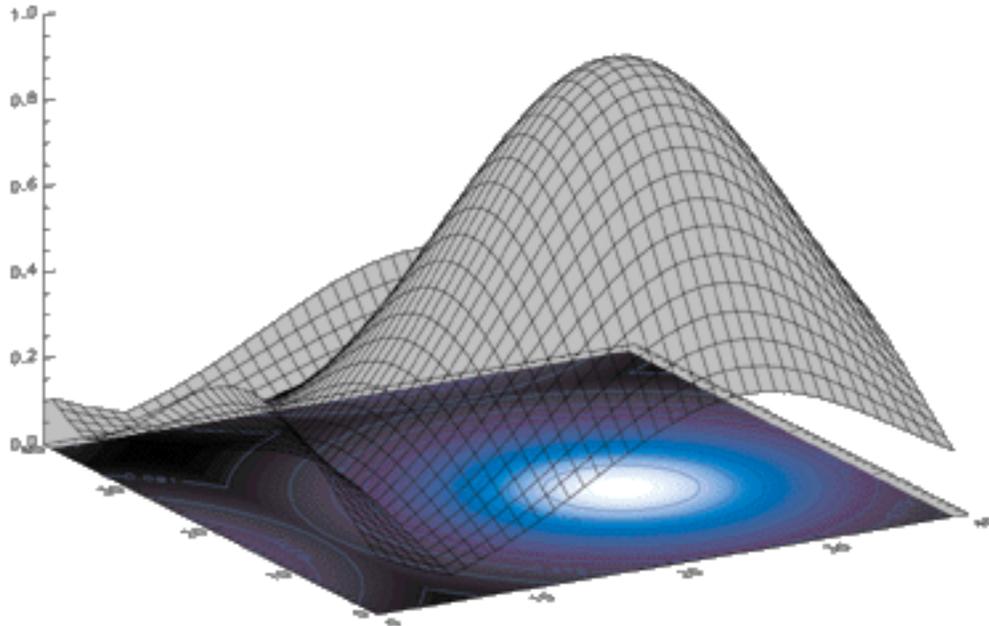
The “Interactive Data Language”, IDL, is a commercial software package that is widely used in physics, astronomy, space sciences, solar physics, and medical applications. IDL was created in the 1970s and has many things in common with Fortran and C.

#### Pros:

- The data syntax is relatively simple and easy to understand.
- The “GNU Data Language”, GDL, a free, public-license software package is available, which is an IDL clone. Most IDL programs will run with GNU.
- Access to many types of data: IDL is able to read most data formats, including FITS, ASCII, FITS, JPEG, etc.
- IDL can be used interactively (on the screen) or to create very complex routines and applications.
- IDL is probably the most powerful tool to analyze data. Routines are readily available for most problems that you will encounter.
- IDL is particularly powerful in terms of visualization of data. It can handle almost anything in 2D or 3D. IDL has a rich set of pre-built visualization tools.
- You can develop procedures that can be added to the library. If you need a procedure often, just call the procedure in your program (no need to type the code in every single program).
- Very powerful and versatile.
- Many IDL libraries exist (for free), such as the IDL Astronomy Library.
- Hundreds of math and statistics routines are available.
- Cross-platform syntax: if you write IDL code, you can run it on a PC, Mac, VMS, UNIX, SUN, etc. You can write a program and send it to colleagues, regardless what computing platform they use.
- Code debugging capabilities: IDL not only tells you where it encountered an error, it also gives hints what may be wrong.
- You can build “widgets” that allow you to interact with your data and graphical visualization products.
- Existing Fortran or C code can be dynamically linked to IDL code, and Fortran or C can call IDL routines.
- Operators and functions can work on an entire array.

**Cons:**

- Expensive! IDL is not free, although GDL is.
- Very powerful and versatile, which can make IDL a beast to handle with a steep learning curve.
- IDL plots to the screen can look different from IDL plots to a file.
- Not optimized for producing plots in the formats that astronomy journals want.



There are many books about IDL that are written at all different levels from beginner's introductions to detailed technical manuals. There are also many good tutorials, cookbooks, and guides available on the Web. One nice site is <http://www.astro.virginia.edu/class/oconnell/ast511/IDLexercises/>, but it is worthwhile googling to find Web sites that you find useful.

## Examples:

```
IDL> a = 2
IDL> b = 5.0
IDL> c = a + b
IDL> help, a, b, c
A      INT    =    2
B      FLOAT  =   5.00000
C      FLOAT  =   7.00000
```

Note that “a” is an integer, but “b”, and subsequently “c” are a floating point, keeping the variable with the highest precision. Variables do not need to be declared.

The real power of IDL is revealed when using its array-oriented language:

For example, declare a variable called “array” as an integer matrix with two dimensions of size 5 columns and 5 rows:

```
IDL> array = indgen (5, 5)
IDL> help, array.
ARRAY      INT    = Array[5, 5]
IDL> print, array
  0   1   2   3   4
  5   6   7   8   9
 10  11  12  13  14
 15  16  17  18  19
 20  21  22  23  24
```

In IDL, columns come first, in other words `array[column,row]`. Array indices start at zero, so the `array[1,3]` is the value in the second row (1) and the fourth column (3). Since IDL is an array-oriented language, any operation that is applied to an array variable will automatically affect every element of the array without having to utilize FOR loops. Example:

```
IDL> array = array * a
IDL> print, array
  0   2   4   6   8
 10  12  14  16  18
 20  22  24  26  28
 30  32  34  36  38
 40  42  44  46  48
```

You can “subscript” in IDL, access individual array elements

```
IDL> print, array [1,3]
      32
```

and perform operations on individual array elements:

```
IDL> test = array[1,3]*10
IDL> print, test
      320
```

**Example of X-ray light curve data:**

Date	L_x	L_x_err
[days]	[cgs]	[cgs]
8.7550	0.3296	0.13
11.465	0.4525	0.17
14.275	0.8877	0.23
16.785	1.2492	0.26
20.076	1.4000	0.26
25.280	1.5023	0.24
29.220	1.4210	0.22
33.681	1.4325	0.27
38.459	1.1971	0.24
43.128	1.1777	0.27
47.208	0.9340	0.20
51.530	0.7605	0.21
61.873	0.5783	0.18

### Example of a program to plot a light curve:

; Give the program a name (anything after a semicolon “;” is a comment):

```
pro lightcurve
```

; Write output to a PostScript file:

```
file='lightcurve'  
filename = file + '.ps'  
set_plot, 'ps'  
device, file=filename
```

; Read in data:

```
header = STRARR(2) ; array of strings  
array = FLTARR(3,13) ; array of floating point numbers  
OPENR, SN2008bo, 'lightcurve.dat', /GET_LUN  
READF, SN2008bo, header, array  
date = REFORM(array[0,*])  
Lx = REFORM(array[1,*])  
Lx_err = REFORM(array[2,*])
```

; Plot axis and data:

```
USERSYM, [0.8,-0.8,-0.8,0.8,0.8], [0.8,0.8,-0.8,-0.8,0.8], /FILL  
PLOT, date, Lx, XRANGE=[0,70], YRANGE=[0.01,2.0], $  
XTITLE='Date', $  
YTITLE='Luminosity', CHARSIZE=1.5
```

; Close Postscript file:

```
device, /close  
set_plot, 'x'
```

```
END
```

## How to Run IDL:

There are two ways you can run IDL.

- 1) **Command-line version:** Just type in "idl" (without the double quotes) at the unix command line prompt.

```
% idl
```

```
IDL Version 6.4.1, Mac OS X (darwin i386 m32). (c) 2007, ITT Visual  
Information Solutions
```

```
Installation number: 1249-7.
```

```
Licensed for use by: NASA/GSFC
```

```
IDL>
```

Create/edit your IDL program with an ASCII editor (*not a word processor such as Word or Open Office*) such as emacs, Xemacs, Textedit (Mac), Notepad (Windows), vi, gedit, etc.

The program should be saved to a file with the filename extension \*.pro.

```
; Give the program a name:
pro lightcurve

; Write output to a Postscript file:

file='lightcurve'
filename = file + '.ps'
set_plot, 'ps'
device, file=filename

; Read in data data:

header = STRARR(2)
array = FLTARR(3,13)
OPENR, SN2008bo, 'lightcurve.dat', /GET_LUN
READF, SN2008bo, header, array
date = REFORM(array[0,*])
Lx = REFORM(array[1,*])
Lx_err = REFORM(array[2,*])

; Plot axis and data:

USERSYM, [0.8,-0.8,-0.8,0.8,0.8], [0.8,0.8,-0.8,-0.8,0.8], /FILL
PLOT, date, Lx, XRANGE=[0,70], YRANGE=[0.01,2.0], $
XTITLE='Date', $
YTITLE='Luminosity', CHARSIZE=1.5

; Close Postscript file:

device, /close
set_plot, 'x'
[]
END
```

-1:\*\* test.pro All L32 (IDLWAVE Abbrev Fill)-----

Most editors will recognize that this is an IDL program and highlight the syntax in different colors.

Next, you need to compile the program by typing “.run” (or “.r”), or “.compile”. Note that there is a period (.) at the start of each command.

```
IDL> .r lightcurve.pro  
% Compiled module: ERR_BAR.  
% Compiled module: LIGHTCURVE.
```

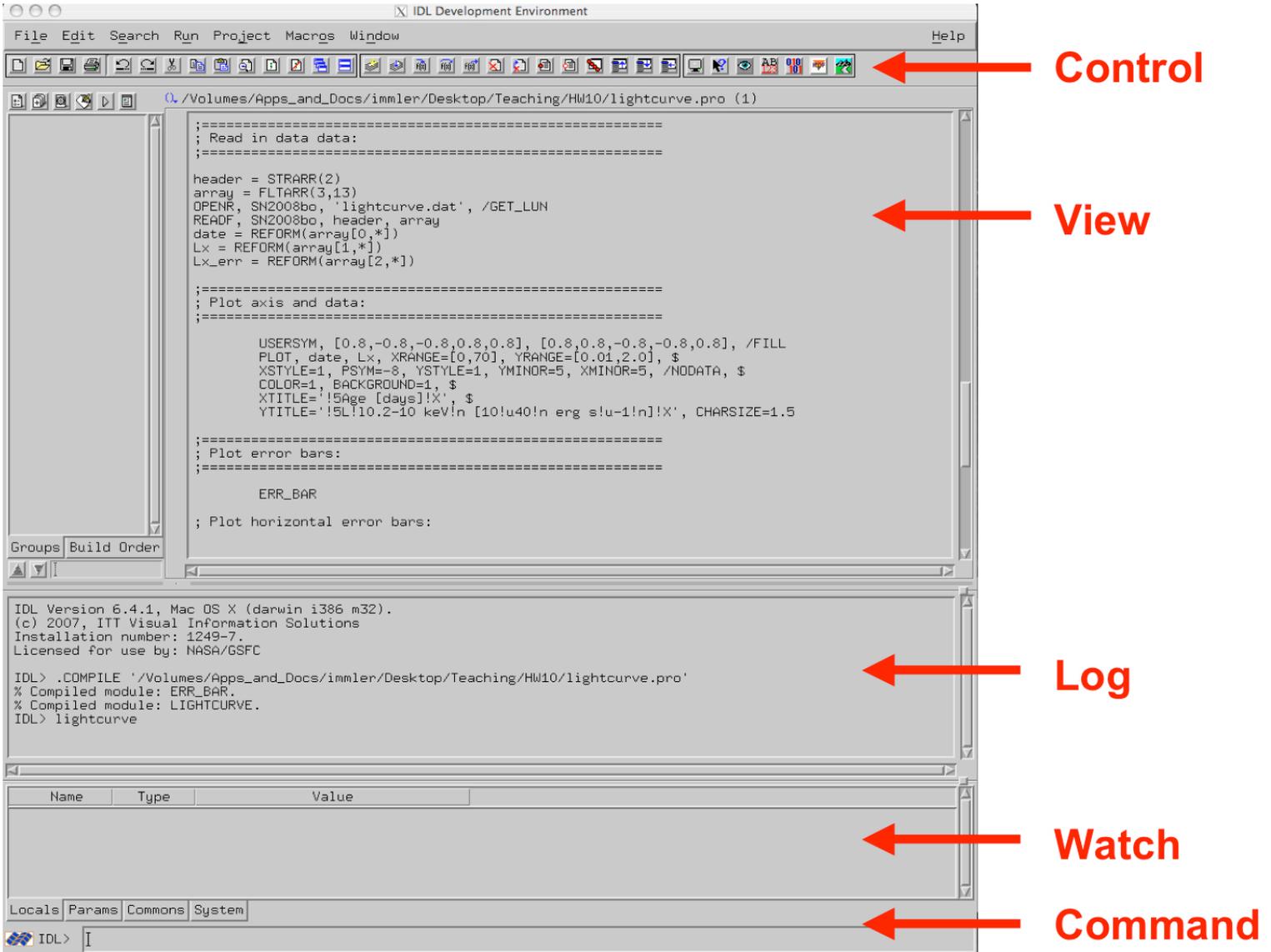
The above example has compiled two modules successfully.

Now run the execute (or run) the program that is in the memory of your computer (but not stored in a file):

```
IDL> lightcurve
```

If you don't get errors, the program ran successfully. The example above has created a plot and saved it into a file.

2) **GUI Version:** Type in “idlde”.  
This will bring up a screen that shows the menu-driven GUI version of IDL.



IDL also has a “**batch mode**” that allows you to execute a series of IDL routines. You can create an ASCII file with all IDL commands and simply run:

```
IDL> @batch
```

By doing so, each line in the batch file is read & executed before proceeding to the next line.

### **The IDL Astronomy User’s Library:**

The IDL Astronomy User’s Library contains ~500 routines that can be easily implemented into your IDL code. It is a repository for astronomy-related IDL programs that have been submitted by users. You can download the codes and modify them to fit your needs. If you have an astronomy problem that needs IDL coding, chances are that there is already a program that you can use or modify.

The IDL Astronomy User’s Library has programs in the areas of coordinate conversion, date conversions, data base procedures, file and disk I/O, table I/O, astrometry, FITS files I/O and manipulations, image manipulations, math & statistics, plotting procedures, web sockets, and non-astronomy procedures.

You can download the IDL Astronomy User’s Library from the link below and add it to your startup script (e.g., “.cshrc” if you are using a C-shell):

```
setenv IDL_ASTRO /usr/local/idlastronlib/data
```

Every time you want to call a program within that library, you type:

```
IDL_ASTRO/<filename>
```

### **How do you get Help?**

At the IDL prompt, type “?”. This will bring up the IDL help widget.

```
IDL> ?
```

### **Links and Useful Resources:**

<http://www.dfanning.com/> (Coyote’s Guide to IDL Programming)

<http://www.itvis.com/UserCommunity/UserForums.aspx> (IDL Forum)

<http://gnudatalanguage.sourceforge.net/> (GDL, a free IDL clone)

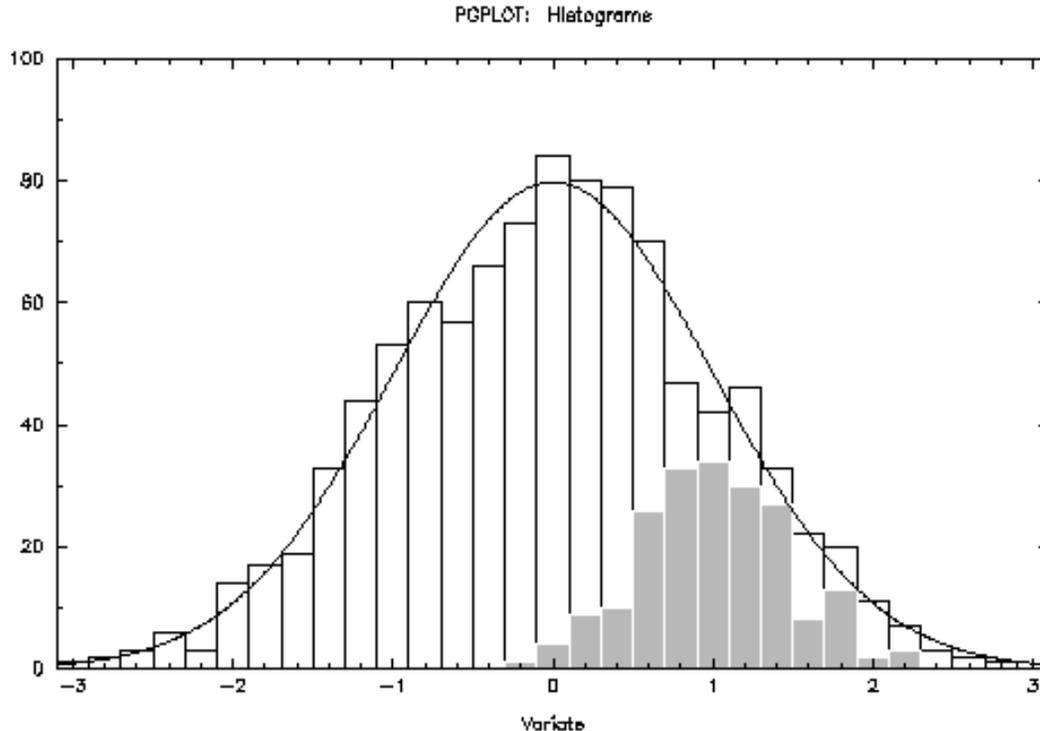
<http://www.itvis.com/ProductServices/IDL/ProductDocumentation.aspx>  
(IDL documentation)

<http://idlastro.gsfc.nasa.gov/> (IDL Astronomy User’s Library)

## 2) PGPLOT Graphics Subroutine Library

One software package that is widely used for plotting data, particularly from within software packages, is the PGPLOT Graphics Subroutine Library. PGPLOT was created at CALTEC and is a device-independent graphics package to make scientific graphs. The main purpose is to make publication quality plots with minimum effort on the part of the user. It is device independent and the output can be directed to any devices (files in many different formats, X windows, printers, etc). It runs on Unix, Mac, and Windows.

PGPLOT itself is written in Fortran 77 and uses (supplied) C and C++ libraries to run. PGPLOT can be called from within many programming environments (such as Fortran, C, FTOOLS, PERL, PYTHON, etc), which makes it very versatile and convenient to use. Below is an example of a plot produced by PGPLOT.



### Links and Useful Resources:

<http://www.astro.caltech.edu/~tjp/pgplot/> (PGPLOT main page)

<http://www.astro.caltech.edu/~tjp/pgplot/contents.html> (PGPLOT manual)

### 3) SM (née supermongo)

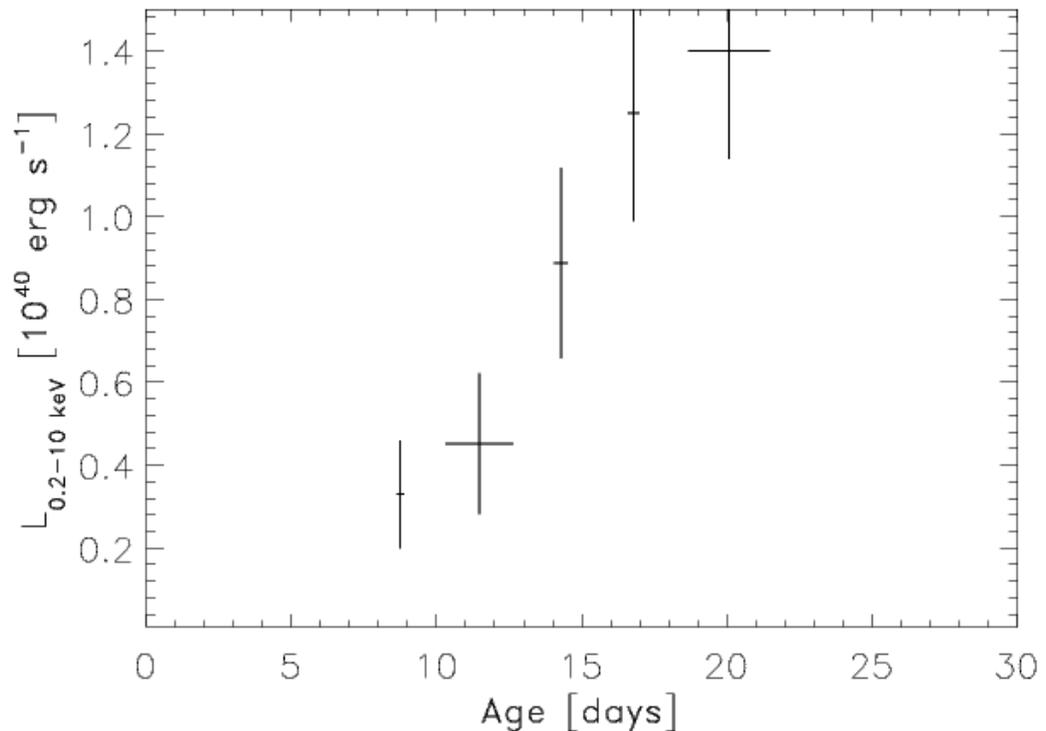
SM is an interactive plotting program with a flexible command language. In many ways it is similar to IDL, but it is smaller and has fewer features. This makes it less useful than IDL, but far easier to learn and use. However, there is still a learning curve for SM. SM can be called interactively or directly from Fortran or C programs. SM is widely used in astronomy because it was written by astronomers and is designed so that the default plotting format is one that is acceptable to all of the major astronomy journals. It also has the ability to do basic data analysis such as fitting functions to data, performing statistical tests, and computing Fourier transforms. It is being overtaken by IDL as the primary plotting package used in astronomy, but is likely to remain in use for many more years. The Web page for SM is at <http://www.astro.princeton.edu/~rhl/sm/>. To run SM one usually types “sm” at a unix command line. Note that SM is not to be confused with another astronomical plotting package called “mongo”.

## Lab Work

The purpose of today's lab exercise is to learn to run IDL, to make simple changes to an IDL procedure, and to make a simple plot.

- 1) Download the IDL procedure "lightcurve.pro" and the data file "lightcurve.dat" from the course Web site at <http://lheawww.gsfc.nasa.gov/~sholland/astr288c/lectures/lecture8/lightcurve.pro> and <http://lheawww.gsfc.nasa.gov/~sholland/astr288c/lectures/lecture8/lightcurve.dat>.
- 2) Log into your account and start IDL. Use either the interactive or command-line version, whichever you prefer.
- 3) Using IDL, read "lightcurve.pro" and execute the procedure "lightcurve". The file "lightcurve.ps" will be created in the directory that you are running in. The ".ps" indicates that this is a PostScript file. You can view PostScript files with the unix command
  - `gv lightcurve.ps &`

The plot that is produced by the original, unedited "lightcurve.pro" procedure should look like this. Note that it does not contain all of the data.



- 4) Rename this plot so that it is not overwritten the next time you run IDL

➤ `mv -i lightcurve.ps whatever.ps`

where “whatever.ps” can be whatever file name you wish. The name should end in “.ps” to indicate that this is a PostScript file.

- 5) The default lightcurve procedure that you downloaded only plots the first five data points. You will need to edit “lightcurve.pro” (either using a text editor or using the interactive IDL environment, *do not use a word processor*) so that it plots all of the data and adjust the upper limits of the axes so that all of the data appears on the plot.
  - a. Change the dimensions of the arrays that contain the data and the range of the for loop that draws the error bars so that they can handle all of the data.
  - b. Increase the upper limits on the axes so all the data are visible.
- 6) Add horizontal error bars for all of the data points. The first five already have horizontal error bars. The horizontal error bars represent the duration of each exposure. For example, the first exposure started at 8.62 days and ended at 8.89 days. Assume that each exposure is exactly one day long unless if other values are given in the IDL procedure.
- 7) Adjust the position of the plot’s title (Swift/XRT) so that is in the upper right-hand corner of the plot. The title needs to be entirely inside the box.
- 8) Print the new plot that is produced by IDL and hand it in as this week’s homework assignment. You may e-mail it to [Stephen.T.Holland@nasa.gov](mailto:Stephen.T.Holland@nasa.gov) before the class on 2 Nov 2009 if you prefer.

## Data

Date	L_x	L_x_err
[days]	[cgs]	[cgs]
8.7550	0.3296	0.13
11.465	0.4525	0.17
14.275	0.8877	0.23
16.785	1.2492	0.26
20.076	1.4000	0.26
25.280	1.5023	0.24
29.220	1.4210	0.22
33.681	1.4325	0.27
38.459	1.1971	0.24
43.128	1.1777	0.27
47.208	0.9340	0.20
51.530	0.7605	0.21
61.873	0.5783	0.18