IDL Basics

Jian-Yang Li, Fernando Roig
jyli@astro.umd.edu, froig@on.br

What is IDL

- An interactive computer language, similar to Matlab.
  - Command line input, execute and output results immediately, like interpret language, Matlab.
  - Programming language, can be used to write procedures, functions, and programs (scripts), like C or Fortran.
  - Expandable with user-supplied packages, such as Goddard library (astrolib), JHU/APL library.

Variables

- Like any other computer language, variables are the basic data unit.
  - IDL is not case sensitive.
  - Variable names start with a letter, containing letters, numbers, underscore, but not special characters such as !, @, &, etc.
  - Variable names can't be IDL reserved words such as le, gt, begin, etc.
- Type of variables can be byte (8-bit unsigned integer), integers, long integer, floating point, double precision, string, complex, structure, etc.
- Declaration is not necessary. IDL will interpret according to context.

Scalars and Arrays

- An IDL variable can be a scalar or an array.
  - IDL processes an array as a single variable. Usually you don't have to write loops.
  - Arrays are indexed like in C, not Fortran.
    - Indices starts with 0
    - For two dimensional array, the first index is column number (x-axis), the second index is row number (y-axis).
  - To quote the specific element(s) in an array:
    - For one element: a[1, 2]
    - For the 6th to the 9th elements in row 10: a[5:8, 10]
    - For all elements in column 15: a[15, *]
    - The square bracket can be replaced by bracket ()

Commands to Process Variables

- Launch the help window of IDL by typing question mark (?) in the command line.
- help, *variable_name*
  - Print out the type of variables, and values for simple types.
  - If *variable_name* is a structure, use keyword /struct to print out the names of its tags.
  - It can also be used to print information about currently compiled procedures and/or functions, use keyword /func and /proc
- print, *variable_name1, variable_name2, …*
  - Print out the value of variables.
  - For arrays, the values of all elements will be printed out.
- Most arithmetic operators in C++ can be used for numerical type variables, including ++, +=.
- Plus sign (+) can also be used on string variables to concatenate strings.
- > and < signs are special:
  - *variable2* = *variable1* < 5, will force all elements in *variable1* that are greater than 5 to 5, and assign the resultant array to *variable2*.

Generating Arrays

- *variable* = fltarr(*number*)
  - Generate a 1-D floating point array with *number* of elements, and initialize the array to all 0s.
- *variable* = fltarr(*column_number*, *row_number*)
  - Generate a 2-D floating point array with the specified numbers of columns and rows.
  - It can take more input parameters to generate multi-dimensional arrays.
- *variable* = findgen(*number*)
  - Generate a 1-D floating point index starting from 0 to *number*-1.
  - Can take two or more input parameters as fltarr does to generate multi-dimensional arrays.
- Corresponding functions to generate other data types
  - Byte type: bytarr, bindgen
  - Integer type: intarr, indgen
  - Double precision: dblarr, dindgen
  - String: strarr

Images

- An image is simply a 2-D array, with each element representing a pixel
  - By default, IDL displays image such that the first element of an array a[0,0] is at the lower left corner, and row runs to the right, columns runs up.
- IDL displays images in 256 gray levels (8-bit), and indexed to the color table that's currently loaded.

Commands to Process Images – 1

- window, *window_index*, xs=200, ys=300
  - Open a window with index *window_index*, and x-size 200 pixels, y-size 300 pixels
- wdelete, *window_index*
  - Delete the window with index specified by *window_index*
- wset, *window_index*
  - Set the window with index *window_index* to active status for output
- erase
  - Erase the content in currently active window
- tv, *byte_array*
  - Display an image without brightness rescaling. Since IDL only display 256 gray levels, you should parse a byte array to tv.
- tvscl, *array*
  - It takes the range of the values in the array, and rescale them to 0-255, then displays it in the active window.

Commands to Process Images – 2

- profiles, *image_name*
  - Interactively display row or column profiles of an image at the position of cursor.
- *output* = rebin(*input_image*, *x-size*, *y-size*)
  - Re-bin the image to the size specified by *x-size* and *y-size*.
  - But note, *x-size* and *y-size* have to be integer factors of the original x-size and y-size, respectively. i.e., either 2x, 3x, 5x, or ½, 1/3, ¼, but not 2.5x or 2/5.
- *output* = congrid(*input_image*, *x-size*, *y-size*)
  - Similar to rebin, but now you can specify arbitrary sizes for *x-size* and *y-size*.

Plotting

- plot, *x_variale*, *y_variable*
  - Plot *y_variable* as a function of *x_variable*
  - Useful keywords, look at help for details
    - psym=2, line=2
    - xtitle='*x-title*', ytitle='*y-title*', title='*title*'
    - xrange=[0.0, 100.0], yrange=[0.0, 1.5]
    - /noerase
- oplot, *x_variable*, *y_variable*
  - Over plot without erasing the previous plot.
  - Some keywords for plot also work for oplot. Check the help.
- Other plotting commands: plots, ploterr, oploterr, errplot

Basic Arithmetic Operations

- Addition, subtraction, multiplication, and division are all the same as in C or Fortran. But now you can do the whole array at once.
  - *variable3 = variable1 + variable2*, where *variable1* and *variable2* can be either scalars or arrays of the same dimensions, and *variable3* has the same dimensions as *variable1* and *variable2*
- Logical expressions
  - *variable1* gt *variable2*, will return a scalar or an array where the elements are 1 if the corresponding elements of *variable1* is greater than *variable2*, or 0 if *variable1* is not greater than *variable2*.
  - Other similar operators are: ge (greater or equal), lt (less than), le (less than or equal), eq (equal), ne (not equal)

Array Searching

- *result* = where(*array_expression*)
  - It will return an array containing the indices of the non-zero elements in *array_expression*.
  - The returned indices can be used to index the non-zero elements of the input array
- For example, print out the non-zero elements in an array:
  - array1 = [10, 0, 23, 44, 0]
  - print, where(array1) will print out 0, 2, 3, which are the indices of all non-zero elements in array1
  - print, array1[where(array1)] will print, 10, 23, 44
- It can also be used for multi-dimensional arrays:
  - print, array1[where(array1 gt 2)] will print out all elements of array1 that are greater than 2, and array1 can be a 1-D, 2-D, 3-D, or any dimensional array.

Other Useful Array Operation

- *result* = n_elements(*array*)
  - Returns the total number of elements in the input *array*, regardless the dimensions
- *results* = size(*array*)
  - Returns the number of dimensions, number of elements in each dimension, variable type, etc., of input *array*.
- *result* = total(*array*)
  - Returns the sum of all elements in the input *array*.
- *results* = mean(*array*)
  - Returns the average of input *array*.
- *results* = median(*array*)
  - Returns the median of input *array*.

To Save Plot or Displayed Image

- Screen capture – this is not a function provided by IDL, but you don't want to forget about it.
  - For a PC running on Win2k or WinXP, press alt+PrintScreen to copy the current window to system clipboard.
  - Or have the IDL display window on the top, and press ctrl+c
- Save current plot to an image file, such as a gif
  - write_gif, *file_name*, tvrd()
- Output the plot or image to a postscript file directly:
  - set_plot, 'ps'
  - device, file=*filename*
  - To save to an encapsulated postscript (eps), use keyword /encapsulated on device command.
  - You can set up the output file such as the size, orientation, etc., with device command. Check its help
  - Now all your graphic output goes to the ps file. After finish drawing, you need to close the file first, by typing device, /close
  - Then set the display back to your screen. The command is different for different systems: for PC-Windows, type set_plot, 'win', for Unix-Linux, type set_plot, 'x'

Programming – 1

- if *expression* then *statement* else *statement*
  or
  if *expression* then begin
  statements
  endif else begin
  statements
  endelse
- for *variable=init*, *limit* , *increment* do *statement*
  or
  for *variable=init*, *limit* , *increment* do begin
  statements
  endfor

Programming – 2

- while *expression* do *statement*
  or
  while *expression* do begin
  statements
  endwhile
- repeat *statement* until *expression*
  or
  repeart begin
  statements
  endrep until *expression*

Procedures and Functions

- IDL supports subroutines, in the forms of procedures and functions. They should all be stored in .pro files.
  - Procedures don't return any values to the caller.
  - A procedure has the form like:

    pro *name_of_procedure*, *input1, input2, …*
    statements
    end
  - To call a procedure, directly call its name, and supply any input it needs.
  - Functions return values to the caller
  - A function has the form like:

    function *name_of_function*, *input1, input2, …*
    statements
    return, *expression*
    end
  - To call a function, use the form *results=name_of_function*(*inputs…*)
  - Now you know that the commands plot, tvscl, etc. are procedures, and n_elements, total , etc. are functions

Scripts/Programs

- A script is a top level program, similar to the main() function in C. It is stored in a .pro file.
- It is simply a series of commands. It is ended with a single statement end in the last line.
- All variables that are defined and used in a script will be visible to command line, after the script has been run.
  - This is different from procedures and functions, where all variables defined inside them are local, meaning invisible to any high level callers, or other subroutines.
- To run a script, type .run *name_of_script_file* in command line (see next slides).

Some Other Useful Command

- save, *variable_name1, variable_name2, …,* filename=*string*
  - Save the variables to an IDL data file specified by the string variable
  - If no *variable_name* is specified, all variables in current session will be stored.
- restore, *string*
  - Restore all variables stored in the file specified by *string*
- exit, to exit IDL
- stop
  - Used in procedures, functions, or programs. The execution will halt at this command, and return the control to command line.
  - This is a very useful command to debug programs.
- Commands starting with a dot (.) are used by IDL as control commands. You can't put them into programs or subroutines.
  - .run *file_name*: for procedures and functions, it compiles them; for scripts/programs, it runs them
  - .compile *file_name*: compile a procedure, function, or script, but it doesn't run any program
  - .go: start execution at the beginning of a previously compiled program
  - .cont: continue to execute the current program that has stopped because of an error, a stop statement, or a keyboard interrupt.
  - .reset: reset IDL, and re-initialize IDL as if you just start IDL. But it doesn't change current directory.

Resources

- Always refer to IDL help
- IDL online turorial
  - http://www.ittvis.com/tutorials/index.asp
- Most commonly used user contributed packages
  - IDL library browser
    - http://www.astro.washington.edu/deutsch/idl/htmlhelp
  - Astrolib library from Goddard
    - http://idlastro.gsfc.nasa.gov/homepage.html
  - JHU/APL library
    - http://fermi.jhuapl.edu/s1r/idl/s1rlib/local_idl.html