# IDL

AST2210

Autumn 2017

In this course you will use the programming language IDL (Interactive Data Language) for one of the labs specifically (the Hinode lab), and optionally for other labs. IDL is similar to MATLAB and Python both in syntax and that it is vectorized and interactive. IDL is used extensively at this institute, particularly in the solar physics group. This document will tell you how to get access to IDL and give you information on functions in IDL we use which is not part of the standard IDL library. For finding more information/tutorials on using IDL, remember that Google is your friend! Otherwise, don't hesitate to ask or send an email.

## Accessing IDL

To get access to IDL you need to remotely log on to the "beehive" machine at the institute. In all examples below replace *username* with your own username and *machinename* with "beehive".

If you are working from home or La Palma you first need to login to *tsih.uio.no*, and from there to one of the machines listed above. This simply means that you run the below commands, and when you are at the terminal prompt of "tsih", you repeat the ssh-command, but now with "beehive" instead.

**From linux** Run *ssh -Y username@machinename.uio.no* from a terminal window.

**From OSX** You may need to install XQuartz if you are running 10.8+. Then run *ssh -Y username@machinename.uio.no* from a terminal window.

**From Windows** On Windows you have two options. Option 2 is more documented and is recommended if you have no clue what you are doing and just want to follow an easy guide. Option 1 is slightly more technical to setup but after setup provides a faster way to remotely log on to the institute, and is less hassle once up and running - so is most recommended overall.

1. Install Xming, when installing choose the option 'Don't install an SSH client'. Install Putty. When using Putty remember to enable X11 forwarding under Connections/SSH/X11. Type *username@machinename.uio.no* in the 'Hostname' box.
2. First remote log on to an IFI machine following these instructions: link. Then start X-win32 following the instructions under 'Xterm-innstillinger' in the link above. Finally type *ssh -Y USERNAME@machinename.uio.no* in the terminal window.

## Starting IDL

IDL is started by typing the following commands in a terminal window in the directory where your IDL files are:

```
tcsh
source ~ainard/ast2210–files/ast2210.tcshrc
sswidl
```

## IDL tips'n'tricks

Below follow tips and general advice on running IDL in this course, like plotting and saving images. This is not a complete guide to IDL, but meant as a resource for things more specific to the course, and students are advised to check IDL documentation other places as well (a simple tutorial link is provided towards the end, and otherwise the web is full of examples - google!).

### The IDL command prompt and saving session variables

IDL has a command prompt, in which you enter commands and define variables successively, with all previously entered commands and assigned variables stored for **the current session**. To save work, you have to either save the output in some format like a text file, or save the session-variables. For now you will probably only need to save some variables from session to session, which can be done as such:

```
IDL> a = 1.0
IDL> save, /variables, filename = 'test_session.sav'
```

The variable $a$ will now be saved (together with any other variable of the current session) in the file test_session.sav in your working directory. To retrieve the variable $a$ in the next session and to print it to in order to verify it has been loaded, write:

```
IDL> restore,  filename = 'test_session.sav'
IDL> print, a
      1.00000
```

Where we see that $a$ has been loaded and still has the correct value.

### Batch files and procedures

Retyping and memorizing code is not the way to program - hence for larger tasks you should create procedures or batch files to run code that you can then edit. In IDL you can either create a batch file or a procedure file. The first can be called anything, and needs no extension. The second has extension `.pro`.

A batch file is run line-by-line - batch mode, hence the name. This means each line in the file is run as if you typed each of the commands in the command-prompt of IDL. An example batch file called `test_file` could for example look like this:

```
a = 10.0
b = 20.0
average = (a + b)/2.0
print, average
```

You run a batch file in batch mode by typing the file name, preceded by the @ symbol. For the above example the output will look like this

```
IDL> @test_file
       15.0000
```

Where the output, the value of `average`, is as expected.

The variables defined and run in the batch file are now still defined in the current session, such that

```
IDL> print, a
       10.0000
IDL> print, b
       20.0000
IDL> print, average
       15.0000
```

still works.

Batch files are not as good for running for loops and other multi-line statements, for this, a procedure is more suited (although you can extend commands over several lines using the $ symbol).

A simple procedure called `test_prog.pro` containing a for-loop may look like this

```
my_array = [1, 2, 3, 4, 5]       ; An array I made
my_sum = 0.0                     ; This will store the sum of
                                 ; my array
for i=0,4 do begin
          ; I loop through all 5 indexes of the array
          ; and sum the elements
          my_sum = my_sum + my_array[i]
endfor

; Finally I print the sum
print, 'The sum of the array is:', my_sum

end
```

Note the final `end` statement that has to be included at the end of the procedure.

And to run it in an IDL session you would type and receive the output:

```
IDL> .run test_prog.pro
% Compiled module: $MAIN$.
The sum of the array is:       15.0000
```

## Arrays, Matrices and indexing

Arrays/Matrices in IDL are indexed from 0. Black and white images like in the Hinode lab will be stored as 2-D matrices, or in the two dimensions of a 3-D matrix if stored together. For example, in the Hinode lab the images you have downloaded are saved in the 3-D matrix `data`. The help command lists the type (INT = integer) and what kind of variable data is, and its dimensions as it is an Array:

```
IDL> help,data
DATA            INT       = Array[4096, 2048, 6]
```

We can here see that each image has a pixel size of $4096 \times 2048$, saved along the first two dimensions, and that there are 6 images in total, with the image index running along the third dimension. To access the entire first image (index 0) but not the others one can access all entries in the first two dimensions using the $*$ symbol:

```
IDL> help, data[*,*,0]
<Expression>    INT       = Array[4096, 2048]
```

In the Hinode lab you are also asked to calculate some statistics for the images, this is easily achived by using the functions `min`, `min`, `avg` and `stddev`:

So, the statistics for a simple array `a` are as an example found by:

```
IDL> a = [1, 2, 3, 4, 5, 6]
IDL> print, min(a)
       1
IDL> print, max(a)
       6
IDL> print, avg(a)
% Compiled module: AVG.
      3.50000
IDL> print, stddev(a)
      1.87083
```

## Some plotting advice

To save plots to eps and pdf files, use the `savep` procedure, example:

```
savep, 'filename-without-ending'
[Misc. plotting commands]
savep
```

To save plots as png-files, use the `write_png` procedure, example:

```
[Misc. plotting commands]
write_png, 'filename.png', tvrd()
```

To plot an image (as an example the first one saved in the 3D matrix `data` from the Hinode lab), you can use `plot_image`:

```
IDL> plot_image, data[*,*,0]
```

The function `plot_image` has many optional inputs you can set, such as `min`, `max`, `scale`, `xtitle`, `ytitle` and `title`, which you can try out as such for example:

```
IDL> plot_image, data[*,*,0], min = 600, max = 1500, xtitle = 'this is the x axis
     title'
```

To inspect smaller parts of an image, use indexing of the x- and y-axis in the array, for example to look at a chunk of elements from 500 to 1000 in both x and y, simply plot the indexed image like below.

```
IDL> plot_image, data[500:1000,500:1000,0]
```

## Saving your images

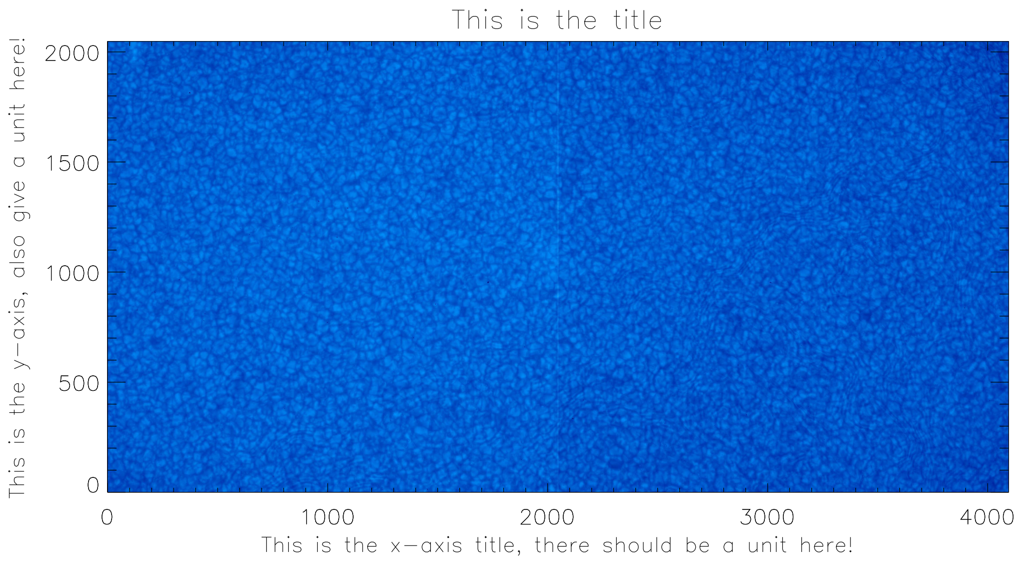
There is a new function called `save_img` available (it should be loaded for you automatically when sourcing the ast2210.tcshrc file) for saving your image plots to .eps or .pdf format.

The function can be called using the same statements as `plot_image` (in fact it for the most part simply transports the output from a `plot_image` to an .eps/.pdf file). Additionally, you can also set a different color table than the default greyscale, using the keyword `color_table`. For a list of available color tables, see here. The different color tables are specified using an integer, as specified on the webpage. If `color_table` is not set, it defaults to Black-White Linear. Feel free to try some out if you want fancy images - though it may very well be that some of the simpler (like the default) are probably easier on the eyes, and best for picking out details.

An example call may look like below.

```
IDL> save_img, data[*,*,0], 'my_image', type = 'pdf', color_table = 2, title = 'This
     is the title', xtitle = 'This is the x-axis title, there should be a unit here!'
     , ytitle = 'This is the y-axis, also give a unit here!'
% LOADCT: Loading table B-W LINEAR
```

The image/array and the filename (without extension) for output must always be specified, and in the case above they are (`data[*,*,0]`) and `my_image` respectively. Additionally the output type is specified to pdf (remember the " marks for string inputs), if you want output as an .eps file, leave `type` unspecified or set it to 'eps'. Note: when the image output is specified to 'pdf', a temporary file of .eps format is created, then deleted once converted to pdf. Further the color-table is set to Blue-White (`color_table` = 2), also a main title, an x-axis title and a y-axis title are given. The image can be seen in Figure 0.1.


This is the title
2000
1500
1000
500
0
0
1000
2000
3000
4000
This is the x–axis title, there should be a unit here!
This is the y–axis, also give a unit here!

Figur 0.1: An example output from `saveimg`.

## Resources

A helpful tutorial for IDL can be found at the Boston University webpage under `http://www.bu.edu/tech/support/research/training-consulting/online-tutorials/idl/`. Here of course not everything will apply completely to our course, but the basic stuff should be helpful.

## Common IDL pitfalls!

- Integer division.
- Integer overflow does not give an error message!
- If you code crashes IDL stop where the error was, this may be inside a function where your variables are not visible, to go back (to 'main') type `retall`.