Exercise: Using IDL

Author
Harald van der Werff

Data needed
aviris_flevoland_ref_scaled.bsq

Introduction

A large part of ENVI functionality comes from IDL (http://www.ittvis.com/idl/). IDL stands for Interactive Data Language and is an object-based programming language that is widely used for data analysis, visualization and cross-platform application development. The combination of IDL and ENVI is a powerful tool for image and spectral processing. Not only can self-made IDL scripts be run from ENVI, but also can IDL-based programs make use of routines available in ENVI. Moreover, ENVI routines can be used in so-called 'batch processing', which is performing a linear sequence of ENVI processing tasks in a non-interactive manner. You can write a batch mode routine (an IDL program) and call it from the ENVI menu system to perform the tasks, or you can start batch mode from the IDL command line.

The first part of this exercise consists of a 'walk through' which explains how to write IDL functions and how to apply these in ENVI using Band Math. After that, you will create a script to calculate rededge position for determining vegetation stress.

Compiling IDL code from ENVI

IDL code can be typed using your favorite text editor, as long as the text can be saved as a plain ASCII (text) file with file extension '.pro'. (Take care that on the Windows platform, some editors tend to save files with a hidden extension . txt: This will not work, the extension has to be '.pro'.)

IDL modules have to be compiled (rewritten in machine code) for use in ENVI, which can be done in two ways:

1. Use "File/Compile IDL Module" from the ENVI menu to load your IDL code.
2. An IDL program can be placed in the 'save_add' directory in the ENVI program directory. All modules in this directory are compiled when ENVI is started. When you do not have writing privileges for this directory, another directory can be selected by changing the preferences from the ENVI menu. Note that if you change a module, you still have to compile it again using "File/Compile IDL Module" from the ENVI menu. This option has mainly use when an IDL function is stable and not being (newly) developed.

Writing IDL functions

In this tutorial, we will build a simple function that calculates the Normalized Difference Vegetation Index (NDVI) using a red and a near-infrared image band.

IDL functions consist of a statement that declares the function and a statement that ends the function:

```
function my_ndvi,band_nir,band_red

end
```

In the declaration of a function comes the name of the function, which is 'my_ndvi' in this example. Also listed and separated by comma's are the variables that will contain the input data. In this example, the input variables are 'band_nir' and 'band_red'. While these two variables are used to get data into the program, a separate statement has to be added to return data from the IDL module to ENVI:

```
function my_ndvi,band_nir,band_red

  return, result
end
```

The function is now ready to get data and to return the result, the only thing that needs to be added are the mathematical expression to calculate NDVI:

```
function my_ndvi,band_nir,band_red

  numerator = band_nir - band_red
  denominator = band_nir + band_red
  result = numerator / denominator

  return, result
end
```

In this equation, we first subtract and add the NIR and Red image bands to make the numerator and denominator. In a separate statement, the numerator is divided by the denominator.

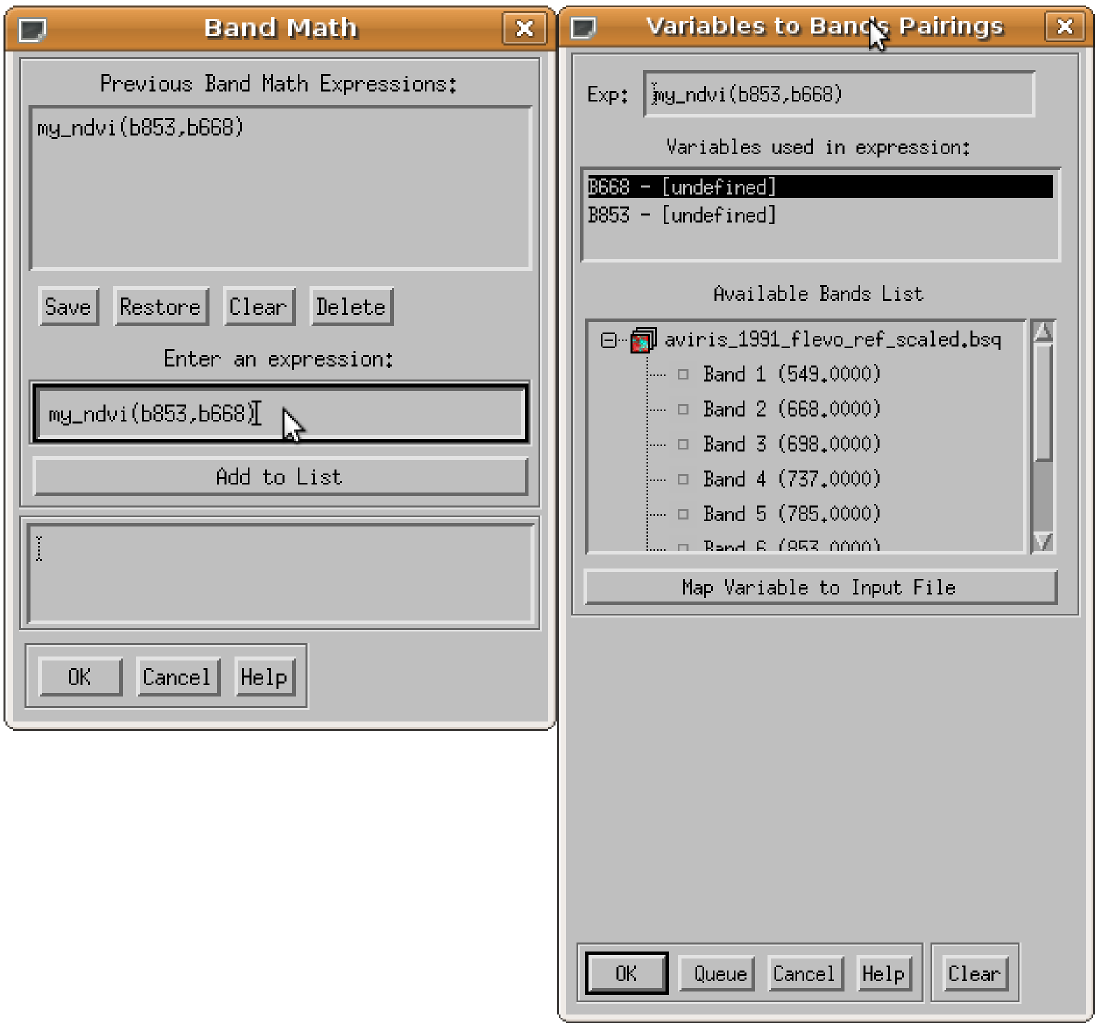
Although this is a working example, it is possible to shorten the code:

```
function my_ndvi,band_nir,band_red
  result = (band_nir - band_red) / (band_nir + band_red)
  return, result
end
```

Mathematical convention is that multiplication and division are done before addition or subtraction. Adding the brackets forces the statement to first to the addition and subtraction (making the numerator and denominator values) and only then do the ratio to create the resulting NDVI value.

The function can be saved as 'my_ndvi.pro' and compiled using ENVI.

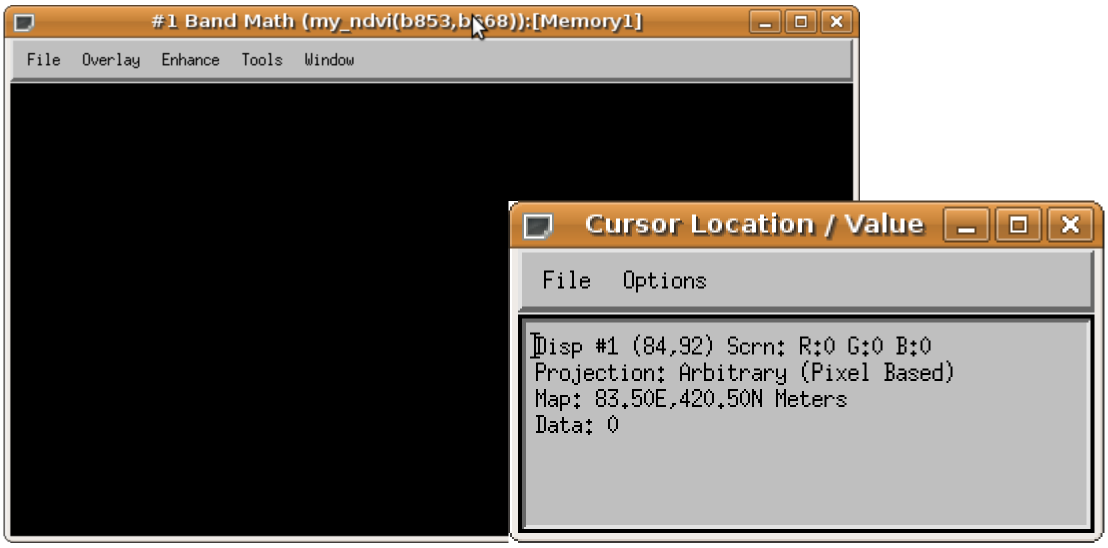
To test the function, open in ENVI image 'aviris_flevo_ref_scaled.bsq'. Call the function you just made from Band Math and assign image bands to the variables by selecting a variable name with the mouse and then selecting an image band:


Band Math
Previous Band Math Expressions:
Save
Restore
Clear
Delete
Enter an expression:
Add to List
OK
Cancel
Help
Variables to Bands Pairings
Variables used in expression:
Available Bands List
Map Variable to Input File
Queue

```
my_ndvi(b853,b668)
```

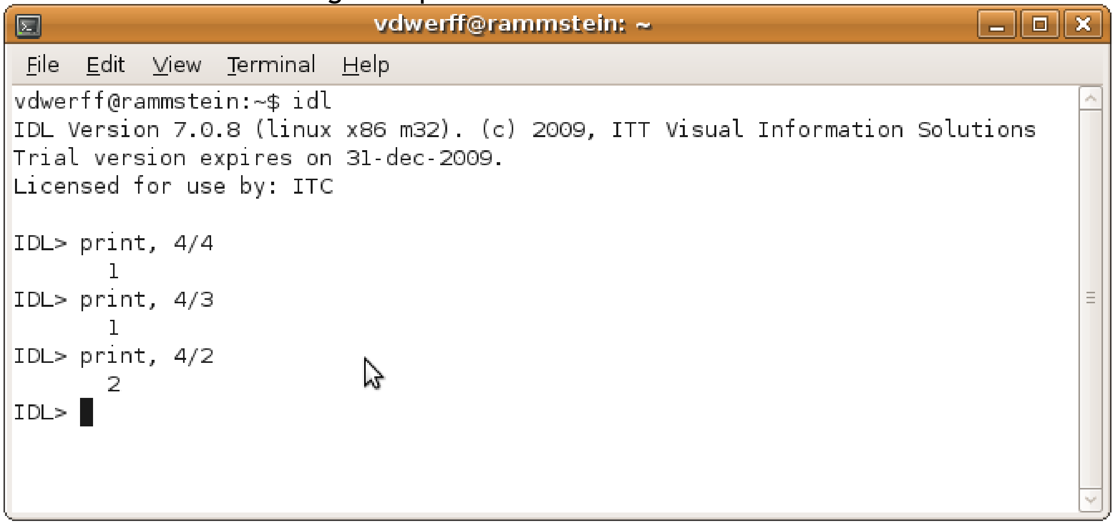
```
Exp: my_ndvi(b853,b668)
B668 - [undefined]
B853 - [undefined]
aviris_1991_flevo_ref_scaled.bsq
Band 1 (549.0000)
Band 2 (668.0000)
Band 3 (698.0000)
Band 4 (737.0000)
Band 5 (785.0000)
Band 6 (853.0000)
```

The result of the calculation is returned as an image and appears in the ENVI 'Available Band List'. Examine the result, it looks like something has gone wrong, the entire image is black! You can examine the pixel values by using the 'Cursor Location/ Value' tool in ENVI:



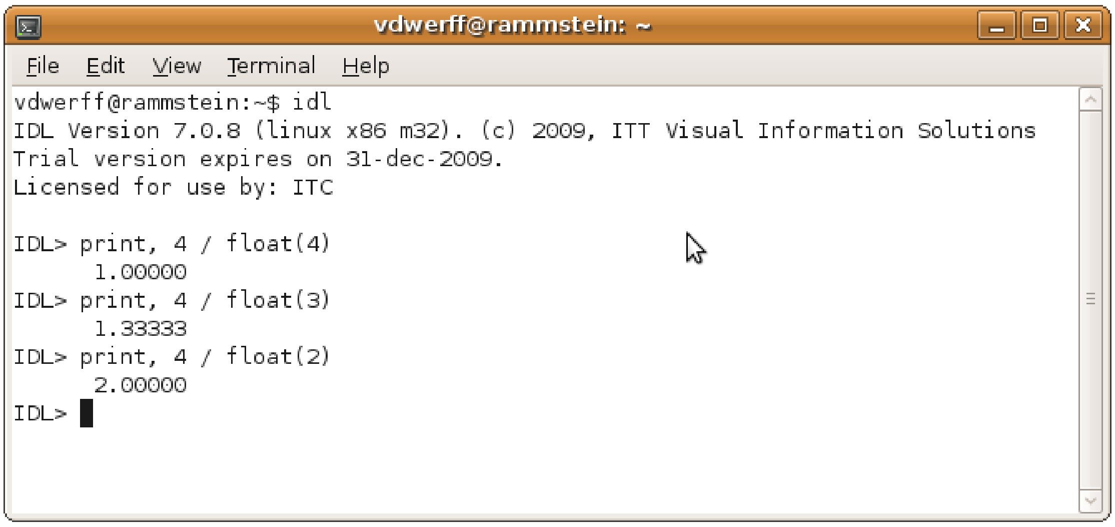
```
Disp #1 (84,92) Scrn: R:0 G:0 B:0
Projection: Arbitrary (Pixel Based)
Map: 83.50E,420.50N Meters
Data: 0
```

This is a result of an other mathematical conventions. The input image is stored in bytes, which are whole numbers (e.g. 0, 5, 21, 36, 255). When a whole numbers is divided by another whole number, the result will also be rounded off to a whole number. See the following example:



```
vdwerff@rammstein:~$ idl
IDL Version 7.0.8 (linux x86 m32). (c) 2009, ITT Visual Information Solutions
Trial version expires on 31-dec-2009.
Licensed for use by: ITC

IDL> print, 4/4
       1
IDL> print, 4/3
       1
IDL> print, 4/2
       2
IDL>
```

You would expect the result of division '4 / 3' to be '1.33333', but it is rounded to '1'. The rounding can be avoided when one of the variables, or both, are real or 'floating point' numbers (e.g. 1.2, 0.5, 3.0). In the following example, the denominator is converted from a whole number to a real number using the function `float()`:



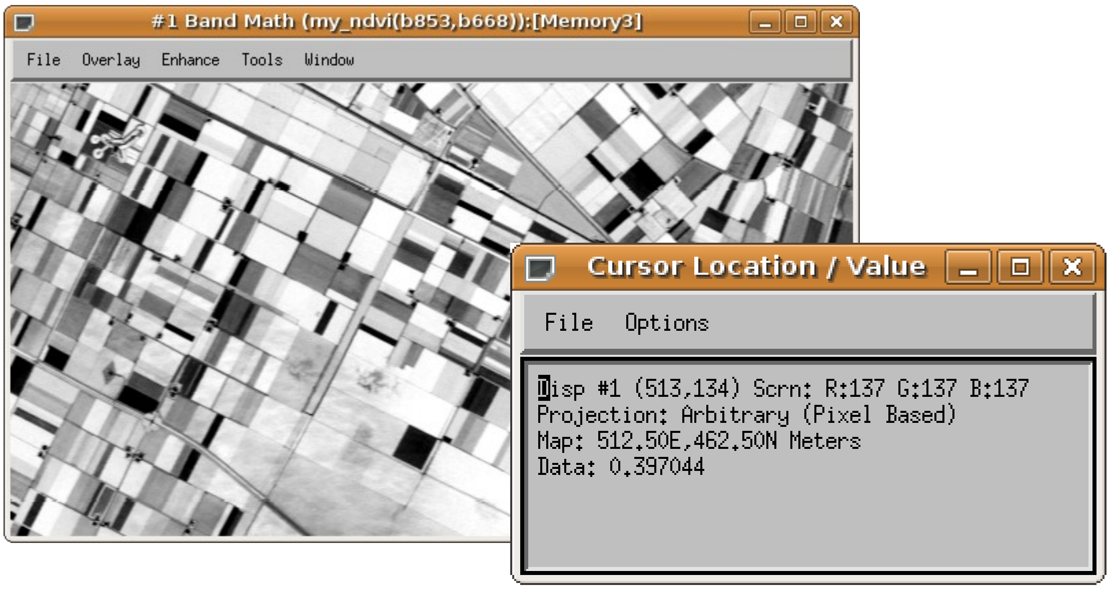
```
vdwerff@rammstein:~$ idl
IDL Version 7.0.8 (linux x86 m32). (c) 2009, ITT Visual Information Solutions
Trial version expires on 31-dec-2009.
Licensed for use by: ITC

IDL> print, 4 / float(4)
      1.00000
IDL> print, 4 / float(3)
      1.33333
IDL> print, 4 / float(2)
      2.00000
IDL>
```

As we do not know in advance if an input image for our function will consist of whole numbers (integer, byte) or floating point numbers (float, double or complex), we have to modify the function so it will always make this conversion for us:

```
function my_function, band1,band2
  result = (band_nir-band_red) / float(band_nir+band_red)
  return, result
end
```

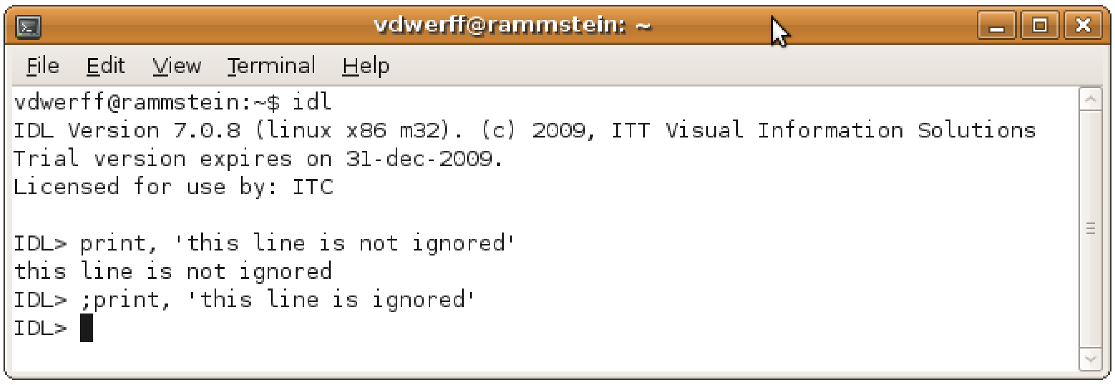
This addition makes the denominator a floating point number before the division is done, automatically making the result also a floating point number.

When the IDL source file has changed, it needs to be re-compiled to get the latest version into computer memory. Use ENVI to compile the code and repeat the application of the function to the AVIRIS image. You will see that the result has now real numbers which show much more detail:


#1 Band Math (my_ndvi(b853,b668)):[Memory3]
File Overlay Enhance Tools Window
Cursor Location / Value
File Options

```
Disp #1 (513,134) Scrn: R:137 G:137 B:137
Projection: Arbitrary (Pixel Based)
Map: 512.50E,462.50N Meters
Data: 0.397044
```

A last tip is to add comments to your code to explain other users of your IDL function, and to help yourself remember what each line in the code means. A line can in the code be declared as 'comment' by starting the line with a semi-colon:



```
vdwerff@rammstein:~$ idl
IDL Version 7.0.8 (linux x86 m32). (c) 2009, ITT Visual Information Solutions
Trial version expires on 31-dec-2009.
Licensed for use by: ITC

IDL> print, 'this line is not ignored'
this line is not ignored
IDL> ;print, 'this line is ignored'
IDL>
```

The final function should hence look something like this:

```
function my_function, band1,band2
  ; this function calculates the NDVI value
  ; of a NIR and red image band
  ; the result is automatically casted
  ; to floating point values
  result = (band_nir-band_red) / float(band_nir+band_red)
  return, result
end
```

Compare the result of this function with the default NDVI module (transform - NDVI). Do you notice any differences? Does the function work correct?

A routine to calculate red edge position
In the second part of this exercise, a function has to be written that calculates the rededge position.

The values of the AVIRIS reflectance image used in this exercise have been rescaled. This is often done by data providers to store the data as integer values rather than floating point values. Unsigned integers are a 16-byte data type while floating points are a 32-bit, taking up exactly twice as much disk space. To reduce the disk usage but keep a enough high dynamic range of data values, the image is rescaled by a factor 100, i.e., 100% reflectance = 100*100 DN = 10,000 DN in image values.

The red edge is calculated in two steps. First, calculate the reflectance at the inflexion point, interpolated from the nearby spectral bands:

$$R_{\Re} = \left(R_{670} + R_{780} \right) / 2$$

Secondly, calculate the red-edge wavelength (L_{re}):

$$L_{\Re} = 700 + 40 * \left(\left(R_{\Re} - R_{700} \right) / \left(R_{740} - R_{700} \right) \right)$$

Write a function that calculates the red edge position from the four image bands. Do not forget to add comments to the code.

Use the image 'aviris_flevo_ref.bsq' as input. Investigate the output and calculate image statistics (statistics – image statistics).

Assignment: Submit in a short report the source code of the red edge function, with comments, and a short comment on the minimum, maximum and mean red-edge values produced by this function.