Programmers' Manual[1]

Wim Bakker

June 22, 2011

[1]This document is distributed under the GNU General Public License

Contents

Chapter 1

Introduction

This document describes the third major revision of envi module (hence the number 2 in envi2 :-). The general-purpose image processing programs that were created using the envi2 module are documented in the User Manual. The Programmers' manual is meant for programmers who want to be able to read, process and write ENVI[1] data using Python.

The envi2 module was designed with the following considerations in mind:

- it should offer read and write capability for the most important ENVI data files.

  Currently, the envi2 module supports reading and writing all ENVI standard image files (bip, bsq, bil), ENVI classification files and ENVI spectral libraries.

- it should transparently map bil, bip and bsq to one internal format (bip).

  All the formats supported by ENVI (bip, bil and bsq) are mapped into one format (bip) by using the array transpose function of Numpy. Numpy supports what it calls 'views' on the arrays, which means that the operation does not create a new transposed array but that it does the transpose operation on the fly on the data whenever data is read or written.

- it should be memory friendly.

  Data files are mapped as arrays using the Numpy memory map function (memmap). Image data is not kept in memory, but all the read and write operations on the array are going directly to file.

  All this means that the real data can be kept on file, while in Python programs the data looks exactly like any other data array.

- it should offer functionality for working with wavelengths next to working with band numbers.

  The methods wavelength2index and index2wavelength will allow you to do that.

[1]ENVI® is a trademark of ITT Industries.

- it should be able to sort wavelengths in the image.

  In some ENVI images, bands are not sorted on wavelength. This may be the case when the image contains bands from different detectors and the detectors have overlapping wavelength ranges or the ranges are not stored in the order of wavelength range. This poses all kinds of problems for viewing the image data or pixel spectra, and this may pose problems on the processing of these images. The envi2 module is able to map wavelengths in sorted order.

  Sorting wavelengths and taking into account the bad band list can be done using two options (sort_wavelengths and use_bbl).

- it should be able to use the bad band list (bbl) and skip bad bands.

  Lastly, the envi2 module can take the bad band list into account, such that the bad bands do not show up in the array data. Bad bands are indicated in the ENVI header file by the keyword 'bbl'. A zero (0) for particular band means that the data in this band should not be trusted, because of sensor failure or excessive noise. A one (1) for a band means that it is a good band.

All in all, the envi2 module offers a view on the image data stored on file and the array data may look quite different from the data on disk (different internal format, different order of wavelengths, different number of bands). Hence, we sometimes use the term 'view' for the images as well.

For the programmer all this means that his or her life has become a lot easier. You don't have to worry about memory usage, you simply leave that to the operating system. Internally, all images have the same structure (bip), so you don't have to worry about the real structure anymore.

Chapter 2 describes the envi2 module and the image processing tools in full detail.

Prerequisites

The software described in this document was developed using the Python programming language version 2.6 [5].

The core of the envi2 module (chapter 2) relies heavily on the array processing functionality of the Numerical Python (NumPy) package [3].

User interfaces (see User Manual) were developed using Tkinter. The Tkinter module ("Tk interface") is the standard Python interface to the Tk GUI toolkit [7].

The continuum removal program (see User Manual) uses a modified version of the Quick Hull algorithm [2] for finding the convex hull.

Display programs (see User Manual) use the Image and ImageTk[2] modules from the Python Imaging Library [4] for displaying images, and Matplotlib (a.k.a. PyLab) for making 2D graphs of spectra [1].

The thermal correction (see User Manual) uses the least squares estimator from the Scientific Python (SciPy) package [6].

[2]On Linux, Image and ImageTk are two separate modules!

Chapter 2

The envi2 module

The envi2 module is a module that contains functionality for easy read and write access to ENVI image files and ENVI spectral library files.

The module consists of a number of submodules, header, image, speclib, spectral, envilist and resample.

When the envi2 module is imported into Python the first four modules, header, image, speclib end spectral are automatically loaded. One statement is needed for loading the functionality:

```
>>> import envi2
```

The envilist and resample modules are used by the other modules and there is no need to load them.

If the resample function is needed in your Python program then you can import it like this:

```
>>> import envi2.resample
```

In the following chapters the submodules will be discussed.

Chapter 3

The header module

The header module is used for reading and writing ENVI header files. ENVI images and spectral libraries typically consist of two files. One contains the data, while the header file contains the metadata of the image or library data. The ENVI header is represented as a Python object of class Header. Objects of this class contain attributes that directly reflect the attributes as they can be found in the ENVI header files. There are a number of things that are different.

One, the names of the attributes in the ENVI header can contain spaces, which is very inconvenient in Python. A choice was made to replace the spaces in ENVI attribute names with an underscore. For instance, the 'data type' attribute in the ENVI file header will be named 'data_type' in Python.

Two, some of the values inside the ENVI header may change appearance in Python. For instance, IDL records, as they appear inside ENVI headers, such as values of the wavelength attribute, are changed into Python lists. The translation between IDL records and Python lists is handled by the module envilist.

There hardly ever is a need to create separate Header objects as most of the interaction of ENVI type files is done via the factory functions Open and New from the image module.

An ENVI header can be read and printed like this:

```
>>> h = envi2.Header('Python/MARS/data/ORB0422_4_jdat')
>>> print h
description = ['ENVI File', 'Created [Thu May 19 15:02:32 2005]']
samples = 128
lines = 366
bands = 352
header_offset = 0
file_type = ENVI Standard
data_type = f
interleave = bsq
sensor_type = Unknown
byte_order = 0
wavelength_units = Unknown
wavelength = [0.92630000000001, 0.94057999999997, 0.95486000000004, 0.9691499999996,...]
bbl = [0, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...]
```

The constructor of the Header class will look for the header file. Either the name of the image file or the name of the header file can be entered and header file will be found.

Compare the information above with the contents of the ENVI header file, ORB0422_4_jdat.hdr:

```
ENVI
description = {
  ENVI File, Created [Thu May 19 15:02:32 2005]}
samples = 128
lines   = 366
bands   = 352
header offset = 0
file type = ENVI Standard
data type = 4
interleave = bsq
sensor type = Unknown
byte order = 0
wavelength units = Unknown
wavelength = {
 0.926300, 0.940580, 0.954860, 0.969150, ...}
; All three detectors, SWIR1+SWIR2+VNIR
bbl = {0, 1, 1, 1, 1, 1, 1, 1, 1, 1, ...}
```

In Python the attributes of the Header object can be accessed directly using the dot notation on the header object *h*:

```
>>> print "Samples:", h.samples
Samples: 128
>>> print "Lines:", h.lines
Lines: 366
>>> print "Bands:", h.bands
Bands: 352
>>> print "Data type:", h.data_type
Data type: f
```

Again, note the use of the underscore in the 'data_type' attribute.

In addition to the ENVI header attributes the Header object contains a number of other attributes that are used for housekeeping.

- attrlist

  This attribute contains the list of attributes that were read from the header. These attributes are also automatically written when needed.

- datatypedict

  This Python dictionary contains a mapping between the ENVI data types and the corresponding Python (numpy) data types. Table 3.1 gives an overview of the codes used.

- revdatatypedict

  This dictionary contains the reverse mapping from Python data types to ENVI data types.

- magic

  Contains the 'magic number.' For ENVI header files this should always be 'ENVI.'

- itoi

  The envi2 module maps the image files on disk to virtual images in Python. These virtual images are also called 'data views.' The virtual image may have a different number of bands (because of the bad band list) or a different order of bands (because it may sort the bands according to wavelength) than the real image on disk. The itoi (index to index) maintains this mapping from virtual band indices to the band index on disk. It should be noted that Python starts numbering bands at 0 while ENVI starts numbering bands at 1.

ENVI data type	**Description**	**Numpy data type**
1	8 bit byte (unsigned in ENVI!)	'u1'
2	16-bit signed integer	'h'
3	32-bit signed long integer	'i'
4	32-bit floating point	'f'
5	64-bit double precision floating point	'd'
6	2x32-bit complex, real-imaginary pair of double precision	'F'
9	2x64-bit double precision complex, real-imaginary pair of double precision	'D'
12	16-bit unsigned integer	'H'
13	32-bit unsigned long integer	'I'
14	64-bit signed long integer	'l'
15	64-bit unsigned long integer	'L'

Table 3.1: Corresponding ENVI and Numpy data types.

Table 3.2 lists the fields that can be used in the ENVI header. In Python code, replace the spaces in ENVI header field names by underscores. In values the spaces should not be replaced. Lists in the ENVI header are enclosed in curly brackets, while lists in Python code are enclosed in square brackets. Strings in the ENVI header are not enclosed in string quotes, while in Python they are.

In addition to these standard keywords, additional modules may define their own keywords for the header file. ENVI normally ignores these extra keywords. Comment can be added to the ENVI file by starting the line with a semi-colon ';'.

The header information is represented by the Header class.

3.1 The Header class

The Header Class contains the ENVI header file information plus some additional house-keeping information. Normally there is no need to directly interact with the Header object. For the information about the image use the bands, lines, samples and wavelength attributes from the Image objects.

The following methods are defined in the header class. Once again, most of the time there is no need to call these directly as most of the communication runs via the Image object.

ENVI field name	Description
band names	list of names for each band of an image.
bands	the number of bands per image file.
bbl	lists the bad bands, 0 for bad bands and 1 for good bands.
byte order	the order of the bytes, 0 for Intel, 1 for everything else.
class lookup	lists RGB color definitions for each respective class.
class names	list of class names, one for each class.
classes	defines the number of classes, including the unclassified.
complex function	the type of image to display in complex data files. Real, Imaginary, Power, Magnitude, or Phase.
data gain values	gain values for each band.
data ignore value	used only in ENVI programming.
data offset values	offset values for each band.
data type	the type of data representation (see table 3.1)
default bands	indicates which band numbers to automatically load.
default stretch	determines the default stretch.
dem band	filename of a DEM that you associate with an image.
dem file	index of the DEM band.
description	describes the image or the processing performed.
file type	the ENVI-defined file type. Currently supported are: ENVI Standard, ENVI Spectral Library, ENVI Classification.
fwhm	lists full-width-half-maximum values for each band.
geo points	geographic corners for non-georeferenced files.
header offset	the number of bytes to skip in the file.
interleave	refers to whether the data are BSQ, BIP, or BIL.
lines	the number of lines in the image.
map info	lists projection information.
pixel size	x and y pixel size in meters for non-georeferenced files.
major frame offsets	bytes to skip in a major frame.
minor frame offsets	bytes to skip in a minor frame.
projection info	describes user-defined projection information.
reflectance scale factor	factor for scaling reflectance to 0–1.
rpc info	rational polynomial coefficient geolocation info.
samples	the number of samples (pixels) per image line.
sensor type	instrument type.
spectra names	(spectral library) contains a list of spectrum names.
wavelength	the center wavelength values of each band in an image.
wavelength units	indicates the wavelength units, Micrometers, Nanometers
x start and y start	coordinates of the top-left pixel. Default values (1,1).
z plot average	the number of pixels in the x and y directions to average for Z plots.
z plot range	the default minimum and maximum values for Z plots.
z plot titles	specific x and y axis titles for Z plots.

Table 3.2: ENVI header file keywords.

```
__init__(h, fname=None, hdr=None, sort_wavelengths=True, use_bbl=True, **keys)
```

This is the contructor for creating new Header objects. In the case that either of sort_wavelengths or use_bbl are set to true then the header is set up in such a way that the image object does not one-to-one reflect the data on disk. In that case, the image data can be seen as a virtual image in which a number of bands may be left out (use_bbl) or the order of the bands may have changed (sort_wavelengths).

By using the hdr argument a new header can be constructed from an existing Header or Image object. The header is constructed by reading the header file first, then the header object and after that the header information can be overwritten or extended by using keyword arguments.

If the header supplied with the hdr keyword is a header from a virtual image, then the new header will *not* be virtual and it will not have an itoi attribute.

Images will only be virtual if they are read from a file and if either the sort_wavelengths or use_bbl arguments are set to True.

```
__str__(h)
```

Header objects can be printed by using a simple print statement. This method is useful for debugging.

```
copy(h)
```

Method for making a copy of a Header object.

```
read(h, fname)
```

This Method is used to read the header information from an ENVI header file.

```
to_attrlist(self, attr)
```

This method is used internally by the Header object to keep track of which of its attributes came from or should go to an ENVI header file.

```
write(h, fname)
```

Method used for writing out the information of a Header object to an ENVI header file.

Header objects also contain the following attributes. These Python dictionaries are used for translating ENVI data types to Numpy data types and vice versa. See also table 3.1.

```
datatypedict = {1:'u1', 2:'h', 3:'i', 4:'f', 5:'d', 6:'F', 9:'D', 12:'H', 13:'I', 14:'l', 15:'L'}

revdatatypedict = {'D':9, 'F':6, 'I':13, 'H':12, 'L':15, 'u1':1, 'd':5, 'f':4, 'i':3, 'h':2, 'l':14}
```

Chapter 4

The image module

This is the main module of the envi2 package. It defines a number of classes for ENVI type images. See table 4.1 for the list of image classes.

Class name	**Description**
Image	abstract superclass
Image1Band	superclass of the Classification class
ImageBIL	for BIL images
ImageBIP	for BIP images and 1-band images
ImageBSQ	for BSQ images
Classification	for classification images

Table 4.1: Image classes.

Note that the image object represents a 'view' and may not actually reflect the data on disc because the wavelengths may be sorted and because the bad band list may be taken into account.

Regardless of whether the original image format was BSQ, BIL or BIP, the view is mapped into BIP format (y, x, band). Also new images are mapped onto a BIP view regardless there output format. Why BIP? Well, why not?

The Image class attributes bands and wavelength reflect the bands and wavelengths of the virtual image, not the one on disk. The original values from the ENVI header file can still be found in the associated header object of the image.

The band attribute may be less than the header.band attribute if a bad band list is used. (if the image was opened with use_bbl=True) Always use the band attribute from the image, not from the header. The header object reflects the metadata as it was read from the header file of the image, while the attributes of the image object reflect the metadata of the 'view' on the image file.

Table 4.2 lists the attributes of Image objects.

The methods of the image classes will be discussed in the following sections.

Image attribute	**Description**
data	the data, mostly of type memmap, 2D (1-band) or 3D (data cube)
header	the metadata from the original header file
samples	number of samples (x)
lines	number of lines (y)
bands	number of bands in the view(z or b)
shape	shape of the data array, basically the same as data.shape
wavelength	list of wavelengths, if supplied in the header.
band_names	list of band names, if supplied in the header.
bbl	bad band list, if supplied in the header.

Table 4.2: Image attributes. Note that these represent the view, not the original image!

4.1 __init__(self, header)

This is the constructor of the image objects. It takes one argument, which is the header object that contains all the information for constructing the image object.

4.2 Getting and setting values

The standard Python __getitem__ and __setitem__ methods were defined for image objects. This means that data can be conveniently retrieved and stored by using an index on the Image objects. Furthermore, the number of indices can vary from 1 to 3, where a different number of indices has a different meaning to the Image object. One-band images are also treated as 3-dimensional datasets, in which the third dimension (the bands) has length one.

Using one single index typically adresses one specific band form the image. Using two indices adresses one pixel (spectrum) in the image. And lastly, three indices are used to address one value from one location and band in the image. Indices can also be slices to indicate a range of indices. Furthermore, the ellipsis (...) can be used to address the whole image.

When using image objects, regardless of their original data type (bip, bil or bsq), the data are mapped to BIP type data. Table 4.3 lists the meaning of the indices.

Indices can also address a range. Some more advanced indexing examples are listed in table 4.4.

Image objects can be easily iterated. Every next iteration gives the next data band. The following example shows that principle:

```
>>> import envi2
>>> im = envi2.Open('/tmp/sub3')
>>> len(im)
158
>>> for band in im:
                # process one band at the time here
```

Number of indices	Order	Meaning
1	im[b]	b is the band number
2	im[y, x]	y is the line number (row), x is the sample number (column)
3	im[y, x, b]	line, sample, band

Table 4.3: Number of Image indices.

Code	Description	Dimension
im[23]	the entire band 23	2D
im[:, :, 23]	the entire band 23	2D
im[10:20]	bands 10 to 20	3D
im[100, 200]	spectrum from y=100, x=200	1D
im[0, :]	all spectra from line 0 (spectral slice)	2D
im[100, 200, 300]	value from y=100, x=200, band=300	scalar
im[100, :, 300]	all values from y=100, band=300 (x-profile)	1D
im[...]	the entire image (ellipsis)	3D
im[:, :, :]	the entire image (slices)	3D

Table 4.4: Some examples of Image indices.

All data that is returned is in the form of a numpy array. The returned array will be in BIP format (y, x, band), in which all the dimensions are optional, but y is always before x and band, and x is always before band.

In addition to using indices on the image also the following methods can be used for retrieving and setting data.

- get_band(self, b) — Get band data with band index b. Returns a 2D array.
- get_spectrum(self, j, i)

  Get the spectrum at location j, i. Returns a 1D array.
- get_value(self, j, i, b) — get a value at location j, i, b. Returns a scalar.
- set_band(self, b, value)

  Set band data with band index b. Value should be a 2D array with the same size as the image.
- set_spectrum(self, j, i, value)

  Set the spectrum at location j, i. Value should be a 1D array with the same number of bands as the (virtual) image.
- set_value(self, j, i, b, value)

  Set a value at location j, i, b. Value must be a scalar.

4.3 Distance measures

For convenience the image objects have a number of predefined spectral distance measures. All these functions take two locations and calculate the spectral distance between these two locations.

The locations t1 and t2 should be 2-tuples (x, y). The convention is different from the indices used on the Image objects, where the order y, x is used.

The following functions are defined:

- spectral_angle(im, t1, t2)

  Calculate the spectral angle between spectra at locations t1 and t2.

- euclidean_distance(im, t1, t2)

  Calculate the Euclidean distance between spectra at locations t1 and t2.

- intensity_difference(im, t1, t2)

  Calculate the intensity difference between spectra at locations t1 and t2.

- bray_curtis_distance(im, t1, t2)

  Calculate the Bray-Curtis distance between spectra at locations t1 and t2.

- spectral_information_divergence(im, t1, t2)

  Calculate the spectral information divergence between spectra at locations t1 and t2.

4.4 Flushing data to disk

When writing new images, the data can be forced to be written to disk with the flush() method. The function sync() has the same effect, but may disappear with newer versions of numpy.

4.5 Wavelength to index

The following methods are used for translating indices to wavelength and vice versa:

- index2wavelength(im, index)

  Returns the wavelength of the band with this index.

  The wavelength is returned in the units that are used in the header file. These may be Micrometers or Nanometers.

- wavelength2index(im, wavelength)

  Returns the index of the band that has the closest wavelength number. The parameter wavelength is the required wavelength. The user is responsible for checking the returned value because there is no guarantee that the requested wavelength is actually present in the image or that the returned wavelength is fit for the purpose it was intended for.

4.6 Methods for virtual images

The following methods are used for the translation between the virtual image and the actual image data on disk. Virtual images may differ from the actual data if either sort_wavelengths or use_bbl was set to True.

- real_band(self, b)

  Translates virtual band number to real band number on disk using the index-to-index (itoi) lookup-table from the header object.

  Argument b can be a number or a list or an array. First band is 0.

- virtual_spectrum(self, s)

  Translates a real spectrum to virtual spectrum by filtering out bad bands as they are given in the header file.

  Note that the units used should reflect the units used in the header file. Currently there is no check on whether the given units make any sense with relation to the wavelengths supplied with the image.

4.7 The Image classes

As of 19 May 2010 the Image1Band class is *not* used for 1-band images anymore. One-band images are now treated as 3D datasets, in which the band dimension is always 1, and returned as ImageBIP objects! This makes the processing of images more uniform as no distinctions need to be made for 1-band images.

The class **Image1Band** is intended for 1-band images only. However, currently it is only used as an (abstract) superclass of the Classification class. It behaves very much like its Image superclass, but there are a few differences. First of all, the image data of a Classification image is treated in a 2-dimensional way and therefore has at most two indices for addressing values in the data. There can be 1 or 2 indices. If a third index is given then this is ignored, even if it's not zero! In the case of one index this can only be the ellipsis (...), which is used to address the entire dataset. This means that im[...] is equivalent to im[:, :]. Indices must be given in the order y, x. returned values are scalars or numpy arrays.

The class **ImageBIL** is used for representing ENVI BIL type images. The data is mapped onto a BIP structure, which means that the data must be addressed with indices in the order y, x, b, just like for all other image types. This class inherits most functionality from the Image superclass.

The class **ImageBIP** is used for representing ENVI BIP type images. This class inherits most functionality from the Image superclass.

The class **ImageBSQ** is used for representing ENVI BIL type images. The data is mapped onto a BIP structure, which means that the data must be addressed with indices in the order y, x, b, just like for all other image types. This class inherits most functionality from the Image superclass.

The class **Classification** is used for representing ENVI Classification images. In ENVI, classification images are 1-band images with a color lookup-table for the classes. The attributes that always must be present for ENVI Classification images are listed in table 4.5.

Keyword	Value
file type	must be set to 'ENVI Classification'
classes	must be set to the number of classes
class lookup	list of color values per class, 3 R,G,B values per class
class names	(optional) list of class names

Table 4.5: Attributes of Classification files.

The following piece of code shows how a Classification file can be created by using the factory function New (see also next section).

```
import numpy
import envi2
from envi2.constants import *

im1 = envi2.Open('/tmp/sub3')

im2 = envi2.New('/tmp/sub3class',
                hdr=im1, bands=1, data_type='u1',
                file_type=ENVI_Classification,
                classes=256,
                class_lookup=(numpy.random.random(3*256)*256).astype('i') )
```

In this example most of the header information for the new image is taken from the first image im1. Some attributes are changed for the purpose of creating a classification image, the number of bands is set to 1, the data type is set to byte ('u1', at most 256 classes), the file type is changed to ENVI Classification (this constant is available in envi2.constants), the number of classes is set to 256 and for the class lookup table 3 times 256 random numbers are created. The result will be a classification image in which every class will have a random color assigned to it.

4.8 Factory functions

These factory function are the functions used for creating the image objects of the classes described above. These are the fucntions used for reading existing data and for creating new ENVI data files. There hardly ever is a need for directly creating the image objects directly.

The following factory functions can be used. The Open function is used for opening existing ENVI data file. The Open function returns one of the image objects Image1Band, ImageBIL, ImageBIP, ImageBSQ or Classification.

- Open(fname, as_type=None, hdr=None, sort_wavelengths=True, use_bbl=True)

  Factory function Open returns one of the image objects of class Image1Band, ImageBIP, ImageBIL, ImageBSQ or Classification.

  The data is basically memory mapped on file, which means that not much extra memory will be used for images. There is one exception, if the argument as_type is supplied then the data will be copied into memory.

A header or image object can be supplied in the hdr argument. This can be useful in the case the header file is corrupted or missing.

The argument sort_wavelengths can be used to make a view on the data in such a way that the bands in the image appear to be sorted on wavelength. This is useful for a file in which detector ranges overlap or are in which wavelength do not appear in the right order.

The argument use_bbl can be used to make a view on the data in which the bad bands are not visible in the image. This is useful to skip the data which is marked as bad in the header file.

The argument as_type should be any of the types defined in the dtype of numpy.

The argument hdr should be of type Image or of type Header.

sort_wavelengths and use_bbl should be True or False (default is True).

Images opened using the Open function can only be read, not written. To protect existing images they are always opened read-only.

- New(fname, hdr=None, **keys)

  Factory function New returns one of the image objects of class Image1Band, ImageBIP, ImageBIL, ImageBSQ or Classification.

  The data is basically memory mapped on file, which means that not much extra memory will be used for images.

  A header or image object should be supplied in the hdr argument for setting up the header information for the newly constructed image.

  Any key=value pair supplied to the function will end up in the Envi header of the image file. This can be useful for controlling the Envi header. The value can be number, string or list. Lists will be translated into curly bracket items and in the Envi header.

  Use file_type=constants.ENVI_Clasification for creating ENVI classification images.

  Note that for created images no views are created, everything is going to disk as is. No mapping by sort_wavelength or use_bbl is assumed.

  Images created with the New function are opened read-write.

Newly created images must have a minimum number of attributes in order to be recognized as valid images. The following code creates a byte image of 100 by 100 by 100 of BIP format.

```
im = envi2.New('/tmp/dummy',
               bands=100, lines=100, samples=100,
               data_type='u1', interleave='bip')
```

The following code gives an example of converting an image from BIL to BIP format:

```
>>> import envi2

>>> im1 = envi2.Open('/tmp/cup95eff.int')

>>> im2 = envi2.New('/tmp/cup95eff_bip', hdr=im1, interleave='bip')
>>> im2[...] = im1[...]
>>> del im1, im2
```

- the first line imports the envi2 module
- the second line opens the existing image. The input image is in BIL format but we don't need to specify this because it is read from the header.
- the third line uses the header of the first image to construct a new output image. The interleave value is overwritten with 'bip'; this will set up the new image for BIP format.
- line four copies the data from the input image to the output image. Because the structure of all images are always mapped mapped to BIP the data can be copied one-to-one; under the hood, the transpose function will take care of the real structure of the images.
- the last line closes the images and in effect writes out all pending data to disk.

Chapter 5

The spectral module

The spectral module defines some distance functions. These functions are used by the Image objects as described above. The distance is calculated on the spectral vectors and can be seen as a distance measure in feature space. The following functions are implemented:

- bray_curtis_distance(s1, s2)

  Bray Curtis distance—also called Sorensen distance—is a normalization method that is commonly used in botany, ecology and environmental sciences. It views the space as grid similar to the city block distance. The Bray curtis distance has a nice property that if all coordinates are postive, its value is between zero and one. Zero bray curtis represent exact similar coordinate. If both objects are in the origin, the Bray curtis distance is undefined. Normalization is done using absolute difference divided by the summation.

- euclidean_distance(s1, s2)

  The Euclidean distance or Euclidean metric is the ”ordinary” distance between two points that one would measure with a ruler, and is given by the Pythagorean formula.

- intensity_difference(s1, s2)

  The intensity difference is simply the difference between the lengths of the two vectors.

- spectral_angle(s1, s2)

  The spectral angle—or angular separation—is the angle between the two spectral vectors in feature space.

- spectral_information_divergence(s1, s2)

  SID originates from the concept of divergence in information theory and measures the discrepancy of probabilistic behaviors between two pixel vectors.

The spectra s1 and s2 are assumed to have the same number of bands and the same wavelengths. If this is not the case then the resample function in the module resample should be used in advance.

Chapter 6

The speclib module

The speclib module defines one single class which is used for reading and writing ENVI Spectral Library files.

The objects of the Speclib class can take indices for convenient adressing of the data. For that purpose the special functions __getitem__() and __setitem__() are defined. The speclib object takes either 1 or 2 indices.

- 1 index — assumes we want a spectrum.
- 2 indices — assumes we want a value.

Indices follow the BIP principle and must be given as (spectrum, band). All indices can be slices. A single ellipsis (...) means the whole dataset. Spectrum names can also be used as indices.

The constructor of the Speclib class takes the following arguments:

```
__init__(self, fname, mode='r', as_type=None, hdr=None, **keys)
```

- fname — file on disk.
- mode — 'r' for reading, 'w' for writing (default 'r').
- as_type

  read spectrum as the type given by the attribute 'as_type'. This should be a numpy type. If no as_type is supplied the data is returned as the original data type on disk.
- hdr — used for constructing new spectral libraries.
- key=value

  any remaining key=value pair will end up in the header. This can be used for setting up new headers from scratch.

The following code shows how to plot all the spectra of one spectral library in one plot (which appears to be not very useful). This assumes you have installed Matplotlib (PyLab) on your machine.

```
>>> import envi2
>>> speclib = envi2.Speclib('/usr/local/itt/envi46/spec_lib/usgs_min/usgs_min.sli')
>>> len(speclib)
481
>>> from pylab import *
>>> for spec in speclib:
        plot(spec)
```

The following code plots the first 10 spectra using wavelengths on the x-axis.

```
>>> for spec in speclib[:10]:
        plot(speclib.header.wavelength, spec)
```

Spectra in the spectral library can be addressed by their full names. The following retrieves an Illite spectrum from the library.

```
>>> spec = speclib['a-illite.spc Ammonio-Illite/Smec GDS87']
```

Because not all names will be known in advance by the user there is also a function for finding spectra names that contain a certain string. For instance, to find all the index numbers of spectrum containing the string ‘illite’ you can apply the following:

```
>>> speclib.spec_match('illite')
[2, 215, 216, 217, 218, 219]
```

The spec_match function can be used, for instance, to plot all the Illite spectra in one plot:

```
>>> for spec in speclib[speclib.spec_match('illite')]:
        plot(speclib.header.wavelength, spec)
```

Also for the Speclib class some distance meaures are defined which can be used for analysis of the spectra. See the Image class for a full description.

The Speclib class also defines the methods wavelength2index and index2wavelength for finding the closest band number to a given wavelength and, vice versa, to get the wavelength that belongs to a certain band.

A spectrum from the spectral library can be resampled using the resample method. The resampling is uses linear interpolation.

The following code is an example of how the resample function could be used to resample a spectrum from the spectral library into the wavelengths used by an image.

```
>>> import envi2

>>> im = envi2.Open('/tmp/sub3')

>>> speclib = envi2.Speclib('/usr/local/itt/envi46/spec_lib/usgs_min/usgs_min.sli')

>>> print im.header.wavelength_units
Nanometers

>>> print speclib.header.wavelength_units
Micrometers

>>> spec_illite = speclib.resampled(speclib.spec_match('illite', matchall=False),
                                    im.wavelength / 1000.0)

>>> plot(im.wavelength, im[10, 20])

>>> plot(im.wavelength, spec_illite)
```

- The first line imports the envi2 module.
- The second line opens the ENVI image. Because we're working with a spectral we will assume that it is a reflectance image.
- Line three opens the USGS mineral spectral library file.
- Line four and five print the units that are used for the wavelengths in both files. Observe that the image uses nanometers while the spectral library uses micrometers. Unfortunately, this keyword is not always set properly in the ENVI header and is often set to Unknown.
- In line six the index of the first Illite spectrum is searched and this spectrum is resampled into the wavelengths used in the image. Observe that the wavelengths of the image need to be scaled by a factor of 1000 because the spectral library uses micrometers for the wavelengths.
- In line seven we plot the spectrum of pixel x, y = 20, 10 from the image against the wavelengths of the image.
- In the last line we plot the illite spectrum. Observe that the Illite spectrum, spec_illite, is now in the same wavelength range as the image itself.

For compatibility with the ascspeclib module (a module designed for reading ASCII spectral libraries) some functions were added to the Speclib class. These functions are listed below.

- wavelength(s) — Return wavelengths of spectrum s.
- wavelength_units(s) — Return wavelength units of spectrum s. This function makes an educated guess of the wavelength units.
- spectrum(s) — Return spectral values of spectrum s.
- name(s) — Return name of spectrum s.
- names() — Return names of all the spectra in the library.
- description(s) — Return description of spectrum s.

Wherever possible, use these functions rather than the other attributes of the Speclib and Header objects.

Chapter 7

The resample module

The envi2.resample module contains only one function, the resample function, `resample(s1, w1, w2)`. This functions takes one spectrum s1 with wavelengths w1 and returns a resampled spectrum for wavelengths w2. The returned spectrum has the same length as w2.

The wavelengths are assumed to be sorted arrays. The spectrum s1 may have to be sorted in advance of using the resample function. If the option `sort_wavelength=True` was used on an ENVI image then the resample function can be safely used on the spectral data.

Bilinear interpolation is performed on s1.

In the following example two ways of calculating a resampled spectrum are shown. The spectrum spec3 is resampled using the 'resampled' function of the spectral library. The spectrum spec4 is resampled using the 'resample' function from the envi2.resample module. The resulting spectral angle of spec3 and spec4 with spec1 is the same. The spectrum spec1 is a spectrum taken from the image at location x=286 and y=291.

```
>>> import envi2

>>> im = envi2.Open('/usr/local/itt/envi46/data/cup95eff.int')
>>> spec1 = im[291, 286]

>>> sl = envi2.Speclib('/usr/local/itt/envi46/spec_lib/usgs_min/usgs_min.sli')

>>> spec2 = sl[sl.spec_match('kaolinite', matchall=False)]
>>> spec3 = sl.resampled(sl.spec_match('kaolinite', matchall=False), im.wavelength)

>>> import envi2.spectral
>>> print envi2.spectral.spectral_angle(spec1, spec3)
0.12274693547

>>> import envi2.resample
>>> spec4 = envi2.resample.resample(spec2, sl.header.wavelength, im.wavelength)

>>> print envi2.spectral.spectral_angle(spec1, spec4)
0.12274693547
```

The result is the same. In fact, under the hood, the spectral library method 'resampled' calls the funtion envi2.resample.resample function.

Chapter 8

The envilist module

The module envilist is used by the header module. It contains two functions to_envi_list(l) and to_python_list(s) for translating information between curly brackets in the ENVI header to Python lists and vice versa. The function find_matching_bracket(s, start=0) is used for parsing header information.

- to_envi_list(l) — converts list l to curly-bracket string.
- to_python_list(s) — converst a curly-bracket string s to Python list.
- find_matching_bracket(s, start=0) — find matching curly bracket.

For the programmer there is usually no need to use these functions.

Chapter 9

The constant module

This module contains a number of constants. The following constants are defined:

```
# ENVI file types
ENVI_Standard = 'ENVI Standard'
ENVI_Classification = 'ENVI Classification'
ENVI_Speclib = 'ENVI Spectral Library'

# ENVI interleave values
ENVI_bip = 'bip'
ENVI_bil = 'bil'
ENVI_bsq = 'bsq'
```

This module is not loaded by default and should be loaded like this:

```
>>> import envi2.constants

>>> print envi2.constants.ENVI_bip
bip
```

Chapter 10

The __init__.py module

This special file is needed to make Python recognize the envi2 directory as a Python module. By default the following (sub-)modules are loaded:

```
from spectral import *

from header import *

from image import *

from speclib import *
```

Other modules, like the resample and constants modules, must be imported separately.

Bibliography

[1] Matplotlib Library for 2D Plots. http://matplotlib.sourceforge.net/. Last visited June 2009.

[2] Modified Quick Hull—Python Code. http://members.home.nl/wim.h.bakker-/python/quickhull2d.py. Last visited June 2009.

[3] NumPy Scientific Computing. http://numpy.scipy.org/. Last visited June 2009.

[4] Python Imaging Library. http://www.pythonware.com/products/pil/. Last visited July 2009.

[5] Python Programming Language Official Website. http://python.org/. Last visited June 2009.

[6] SciPy—Scientific Tools for Python. http://scipy.org/. Last visited June 2009.

[7] TkInter Wiki. http://wiki.python.org/moin/TkInter. Last visited June 2009.