HypPy Spectral Math Manual[1]

for the Spectral Library Viewer & programming

Wim Bakker

January 9, 2020

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

Contents

Abstract

Chapter 1

The Spectrum Class

1.1 Introduction

The Spectrum was created for making manipulation of spectra easy. The idea is that Spectrum object contains information about wavelength as well as values be it reflectance, radiance or other data. The Spectrum class offers easy access to the data, while at the same time it offers an easy way to operate on the spectral data.

The list below summarizes some of the properties of the Spectrum object:

- the Spectrum object contains wavelengths and values
- an index can be used to select values or make subsets of the spectrum
- mathematical operators like +, -, * and / can be used on spectra
- spectral indices can be easily build
- the Spectrum object contains a wealth of functions
- when working with spectra having different wavelengths the spectra are automatically resampled on the fly

1.2 Spectrum data

1.3 Spectrum index

S[]

1.4 Spectrum as iterator

for...

1.5 Spectrum as string

repr, str

1.6 Spectrum as function

__call__

1.7 Spectrum filtering

smooth, median, gradient, integrate, convolve

1.8 Spectrum resampling

resample, interpolate, cut

1.9 Spectrum plotting

plot, wplot, pplot, lplot

1.10 Spectrum high level functions

hull, localmax, shoulders, asymmetry

Chapter 2

Methods of the Spectrum class

The following methods are defined in the class Spectrum:

__abs__(self) Abs operator.

__add__(self, other) Add operator.

__call__(self, wav) Returns the band with given wavelength.

There is no check on units. There is no check on whether the returned band is close to the requested band.

The difference with the index is that indices can contain slices while function calls can not.

__coerce__(self, other) Coerce one spectrum into another if needed.

This determines the union of the sets of wavelengths of both spectra and resamples both spectra into the unioned wavelengths.

Wavelengths of the union are sorted afterwards.

__div__(self, other) Division operator.

__eq__(self, other) Operator equal.

__ge__(self, other) Operator greater than or equal.

__getitem__(self, i) The Spectrum object can handle many things as indices. Indices can be integers, floats, strings, slices, tuples and sequences.

Integers are normal indices:

```
>>> S[251]
0.83959899999999998
```

Floats are assumed to be wavelengths and are translated to the index with the nearest wavelength (see wavelength2index):

```
>>> S[1.0]
0.83959899999999998
```

If the "wavelengths" are band names then the band name can be searched using strings:

```
>>> S.pelkey()['olindex']
0.023354440658353237
```

(the function pelkey produces a Spectrum with band names instead of wavelengths)

Slices are translated to ranges of a spectrum:

```
>>> S[100:200]
0.5273 0.776515
0.5293 0.778634
0.5313 0.780073
...
```

Floats can be used in slices:

```
>>> S[1.0:2.0]
1.0 0.839599
1.006 0.83894
1.012 0.839281
1.018 0.839616
...
```

Tuples can be used to select multiple parts from the spectrum:

```
>>> S[1.0:2.0, 2.5:]
1.0 0.839599
...
1.995 0.724337
2.496 0.399651
...
2.976 0.057666
```

A sequence can be used to select a list of particular wavelengths from a spectrum:

```
>>> S[[0.5, 1.0, 1.5, 2.5, 3.0]]
0.4993 0.752254
1.0 0.839599
1.4985 0.66482
2.496 0.399651
2.976 0.057666
```

Combinations of the above are also possible:

```
>>> S[0.8:1.0:10, 270, [400, 410, 3.0]]
0.8015 0.844845
0.871 0.843062
0.964 0.842001
1.0985 0.84682
1.865 0.803714
1.965 0.748131
2.976 0.057666
```

__gt__(self, other) Operator greater than.

__init__(self, wavelength=None, spectrum=None, name=None, description=None)
Constructor for the class Spectrum. Wavelength and spectrum must be iterables.

The wavelengths are assumed to be sorted.

The class Spectrum can be used to construct new spectra on the fly. Here is an example Math Expression for down-sampling a given spectrum S to a spectrum using the ASTER center wavelengths and bandwidths. Only the VNIR and SWIR wavelengths are used here:

```
Spectrum([0.56, 0.66, 0.81, 1.65, 2.165, 2.205, 2.26, 2.33, 2.395],
[S[0.52:0.60].mean(), S[0.63:0.69].mean(), S[0.76:0.86].mean(), S[1.6:1.7].mean(),
S[2.145:2.185].mean(), S[2.185:2.225].mean(), S[2.235:2.285].mean(),
S[2.295:2.365].mean(), S[2.36:2.43].mean()]).plot()
```

__iter__(self) Return the iterator itself (copy)

__le__(self, other) Operator less than or equal.

__len__(self) Returns the number of elements in the spectrum.

Example:

```
>>> len(S)
473
```

__lt__(self, other) Operator less than.

__mul__(self, other) Multiply operator

__ne__(self, other) Operator not equal.

__neg__(self) Negative operator.

__pos__(self) Positive operator.

__pow__(self, other) Power operator.

__radd__(self, other) Radd operator (right add).

__rdiv__(self, other) Rdiv operator (right division).

__repr__(self) Returns representation of spectrum.

All the wavelength/spectrum pairs will be returned as a string separated by newlines:

```
>>> repr(S)
'0.2211 0.175487
...
2.976 0.057666'
```

__rmul__(self, other) Rmul operator (right multiply).

__rpow__(self, other) Rpow operator (right power).

__rsub__(self, other) Rsub operator (right subtract).

__str__(self) Returns the Spectrum converted into string.

All the wavelength/spectrum pairs will be returned as a string of one line:

```
>>> str(S)
' 0.2211 0.175487 ... 2.976 0.057666'
```

__sub__(self, other) Subtract operator.

absolute(self) Numpy function absolute.

add(self, other) Numpy function add.

all(self, *args) Numpy function all.

any(self, *args) Numpy function any.

arccos(self) Numpy function arccos.

arccosh(self) Numpy function arccosh.

arcsin(self) Numpy function arcsin.

arcsinh(self) Numpy function arcsinh.

arctan(self) Numpy function arctan.

arctan2(self, other) Numpy function arctan2.

arctanh(self) Numpy function arctanh.

astype(self, t) Convert spectral values to type t.

For type codes see for instance numpy.typeDict

Example, convert a spectrum to byte values:

```
>>> T = (255.99*S).astype('u1')
```

asymmetry(self, n=None, level=0.5) Calculate asymmetry of peaks.

If n is given, only the asymmetry of the n highest peaks are returned.

Asymmetry is calculated from the distances of the peak to the left and right shoulders.

Example:

```
>>> S.peaks().asymmetry(5)
2.175 −0.027896809228
2.215 0.768096263038
2.466 −0.253204867227
2.528 0.338614909345
2.816 −0.0931167428635
```

A negative asymmetry means that the left side is narrower than the right side (the peak is leaning towards the left).

The returned spectrum is a pSpectrum (point spectrum).

clip(self, a, b, *args) Numpy function clip.

Clip values to between minimum a and maximum b.

convolve(self, kernel) Convolve spectrum with a linear kernel.

Kernel must contain exactly 3 values.

Example, average (smooth) spectrum S:

```
>>> T = S.convolve([1/3.,1/3.,1/3.])
```

Example, calculate gradient (edge filter) of S:

```
>>> T = S.convolve([−1, 0, 1])
```

copy(self) Numpy function copy.

cos(self) Numpy function cos.

cosh(self) Numpy function cosh.

cumprod(self, *args) Numpy function cumprod.

cumsum(self, *args) Numpy function cumsum.

cut(self, w1=None, w2=None) Cut spectrum between w1 and w2. w1 and w2 will always be the end points and may be interpolated.

The behavior of a "cut" is different from the behavior of a slice:

```
>>> S[1.1:1.2]
1.0985 0.84682
...
1.1985 0.846374

>>> S.cut(1.1, 1.2)
1.1 0.846604
1.1035 0.8461
...
1.1985 0.846374
1.2 0.8469995
```

You could see a slice as a "quick and dirty cut" and a cut as a "precise cut".

deg2rad(self) Numpy function deg2rad.

divide(self, other) Numpy function divide.

dpeaks(self) Calculate absorption peaks from a spectrum.

The function calculates the distance between the spectrum and its convex hull.

The calculation is done using the geometry part of the Python module 'shapely'. If shapely is not installed this function will raise a NotImplementedError exception.

equal(self, other) Numpy function equal.

exp(self) Numpy function exp with base e (2.7182818284590451).

exp2(self) Numpy function exp2 with base 2.

fit(self, n=1) Fit a polynomial with order n through spectrum.

The resulting interpolated spectrum has exactly the same wavelengths as the input spectrum.

float(self) Convert spectral values to floating point numbers.

gradient(self) returns a first order derivative of a spectrum.

greater(self, other) Numpy function greater.

greater_equal(self, other) Numpy function greater_equal.

hull(self) Returns convex hull as a spectrum.

Example:

```
>>> H = S.hull()
```

Note that the hull may contain less points than the original spectrum:

```
>>> len(S)
473
>>> len(H)
23
```

hypot(self, other) Numpy function hypot.

index2wavelength(self, 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 microns, nanometers or anything else...

int(self) Convert spectral values to integers.

integrate(self) Integrate spectrum.

Calculates the area under a spectrum using the scipy.integrate.trapz function.

interpol(self, w) Linear interpolation of spectrum at wavelength w.

Example:

```
>>> S[[101]]
0.5293 0.778634

>>> S[[102]]
0.5313 0.780073

>>> S.interpol(0.53)
0.77913765000000001
```

less(self, other) Numpy function less.

less_equal(self, other) Numpy function less_equal.

localmax(self, n=None) Return the local maxima of the spectrum.

If n is given, return the n largest local maxima.

Returns a pSpectrum (point spectrum).

localmaxfit(self, n=None) Fit the n largest local maxima with parabola.

localmaxidx(self, n=None) Return indices of the local maxima.

If n is given, return only the n largest local maxima.

localmin(self, n=None) Return the local minima of the spectrum.

If n is given, return the n smallest local minima.

Returns a pSpectrum (point spectrum).

localminfit(self, n=None) Fit the n smallest local minima with a parabola.

localminidx(self, n=None) Return array of indices of the n smallest local minima.

log(self) Numpy function log.

log10(self) Numpy function log10 with base 10.

log2(self) Numpy function log2 with base 2.

long(self) Convert spectral values to long integers.

max(self, *args) Numpy function max.

Here's a trick to calculate the maximum value of all the bands of a pixel:

```
i1[...].max(axis=2)
```

This takes the entire datacube and calculates the maximum in the direction of the third axis (axis=2).

maximum(self, other) Numpy function maximum.

mean(self, *args) Numpy function mean.

medfilt(self, n=3) Smooth spectrum with a median filter.

Uses scipy.signal.medfilt

Note that median filtering may introduce flat areas in a spectrum which means that local extrema may not be found anymore.

min(self, *args) Numpy function min.

minimum(self, other) Numpy function minimum.

minwav(self, n=None) Calculate the minimum wavelength.

Equivalent to: 1 - S.nohull().localminfit(n)

Returns a pSpectrum (point spectrum) containing wavelength/depth pairs.

multiply(self, other) Numpy function multiply.

negative(self) Numpy function negative.

next(self) Return next value of the iterable.

Values are 2-tuples containing a wavelength and a spectral value.

Example:

```
>>> for w, s in S:
        print w, s


0.2211 0.175487
0.2291 0.199842
0.2361 0.210632
0.2421 0.214605
...
```

nohull(self, mode='div') Returns the continuum removed as a spectrum.

Two modes are implemented: 'div' (divide) and 'sub' (subtract).

Mode 'div' is equivalent to: S / S.hull()

Mode 'sub' is equivalent to: 1 + (S - S.hull())

Mode 'div' (divide) is the default.

nonan(self) Remove NaNs from spectrum.

not_equal(self, other) Numpy function not_equal.

peaks(self) Calculate absorption peaks from a spectrum.

Equivalent to: S.hull() - S

pelkey(S) Returns Summary Products of a spectrum.

Reference: Pelkey et al., 2007, J. Geophys. Res., Crism multispectral summary products: Parameterizing mineral diversity on Mars from reflectance.

The result is a Spectrum with band names rather than wavelengths.

Currently 39 summary products are implemented.

plot(self, *args, **kwargs) Plot a spectrum as lines.

Example, plot spectrum and its convex hull in one window:

```
>>> S.plot()
>>> S.hull().plot()
```

power(self, other) Numpy function power.

pplot(self, *args, **kwargs) Plot spectrum as points.

prod(self, *args) Numpy function prod.

rad2deg(self) Numpy function rad2deg.

reciprocal(self) Numpy function reciprocal.

resample(self, w=None, step=None) Resample spectrum to wavelengths w or in steps with size step.

Example, resample spectrum S into wavelengths of spectrum T:

```
>>> R = S.resample(T.w)
```

Example, resample spectrum S into steps of 1 nanometer:

```
>>> Q = S.resample(step=0.001)
>>> len(S)
473
>>> len(Q)
2755
```

That resampling means loss of information can be ”proven” with:

```
>>> (S−T).absolute().sum()
0.061246582669433898
```

resample2aster(self) Resample spectrum to ASTER bands. This does not use the real SRF of ASTER but uses an approximation by a block function.

Example, resample spectrum S into ASTER bands:

```
>>> R = S.resample2aster()
```

resample_bandwidth(self, w=None, bandwidth=None) Resample spectrum to wavelengths w using band widths in bandwidth. This assumes that the spectral response function (SRF) is a block function.

Example, resample spectrum S into wavelengths of ASTER:

```
>>> R = S.resample_bandwidth(
                [0.56, 0.66, 0.81, 1.65, 2.165, 2.205, 2.26, 2.33, 2.395],
                [0.08, 0.06, 0.1, 0.1, 0.04, 0.04, 0.05, 0.07, 0.07])
```

round(self, *args) Numpy function round.

second(self) Returns a second order derivative of a spectrum.

shoulders(self, n=None, level=0.5) Calculates shoulders of a spectrum.

If n is given then only the shoulders of the first n highest peaks are returned. A level of 0.5 gives full width half maximum (FWHM).

The shoulders are return as a pSpectrum (point spectrum). The shoulders are not necessarily in sorted order.

sign(self) Numpy function sign.

sin(self) Numpy function sin.

sinh(self) Numpy function sinh.

smooth(self, n=1) Smooth spectrum.

If n is given, smooth spectrum n times.

S.smooth() is equivalent to S.convolve([1/4., 1/2., 1/4.])

S.smooth(2) is equivalent to S.convolve([1/4., 1/2., 1/4.]).convolve([1/4., 1/2., 1/4.])

sort(self) Returns a spectrum sorted by wavelength.

sqrt(self) Numpy function square root.

square(self) Numpy function square.

std(self, *args) Numpy function std.

subtract(self, other) Numpy function subtract.

sum(self, *args) Numpy function sum.

tan(self) Numpy function tan.

tanh(self) Numpy function tanh.

var(self, *args) Numpy function var.

wavelength2index(self, wavelength) Returns the index of the band with the closest wavelength number.

Wavelength is the required wavelength.

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

Note that the "closest" wavelength may not be the expected wavelength:

```
>>> S.wavelength2index(5.0)
472
>>> S.index2wavelength(472)
2.976
>>> S.wavelength[472]
2.976
```

In the example above we request a wavelength of 5.0 but get a wavelength close to 3.0.

where(self) Numpy function where.

The where function returns indices of values that are true.

Example, plot all values of S where S is larger than 0.84:

```
>>> S[(S>0.84).where()].plot()
```

wplot(self, *args, **kwargs) Plot wavelengths as vertical lines.

Example, plot a spectrum and indicate the 5 deepest absorption peaks with dotted vertical lines:

```
>>> S.peaks().plot()
'plot'
>>> S.peaks().localmax(5).wplot()
'plot'
```

wshift(self, shift=0, mult=1) Shift spectrum to higher/lower wavelengths.

For instance, shift the whole spectrum 10 nanometer to the higher wavelengths:

```
>>> T = S.wshift(0.01)

>>> S
0.2211 0.175487
0.2291 0.199842
...
```

```
>>> T
0.2311 0.175487
0.2391 0.199842
...
```

Convert a spectrum to nanometers:

```
>>> T = S.wshift(mult=1000)
>>> T
221.1 0.175487
229.1 0.199842
...
```

Chapter 3

Examples

3.1 Programming examples

3.2 Spectral Library Viewer examples