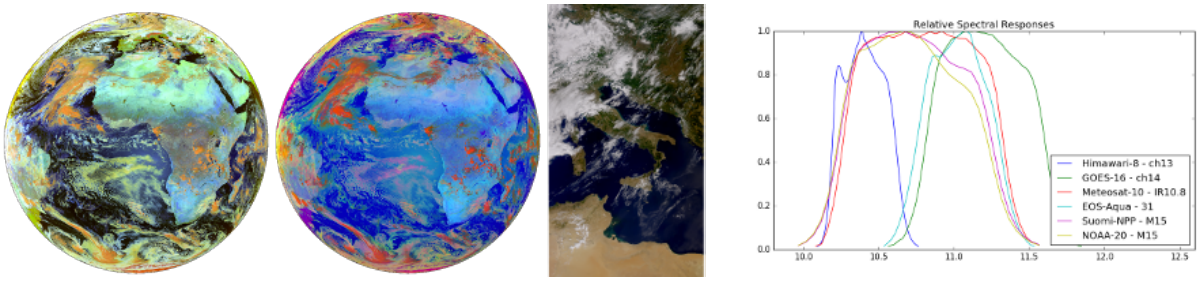
Pyspectral Documentation

Release 0.13.1.dev0+g427ddc7.d20231128

Adam Dybbroe

Nov 28, 2023

CONTENTS



Given a passive sensor on a meteorological satellite PySpectral provides the relative spectral response (rsr) function(s) and offers some basic operations like convolution with the solar spectrum to derive the in band solar flux for instance. The focus is imaging sensors like AVHRR, VIIRS, MODIS, ABI, AHI and SEVIRI. But more sensors are included and if others are needed they can be easily added. With PySpectral it is possible to derive the reflective and emissive parts of the signal observed in any NIR band around 3-4 microns where both passive terrestrial emission and solar backscatter mix the information received by the satellite. Furthermore PySpectral allows correcting true color imagery for background (climatological) Rayleigh scattering and aerosol absorption.

The source code can be found on the github page.

With PySpectral and SatPy (or previously mpop) it is possible to make RGB colour composites like the SEVIRI day solar or the day microphysical RGB composites according to the MSG Interpretation Guide (EUMETSAT). Also with SatPy integration it is possible to make atmosphere corrected true color imagery.

CHAPTER

ONE

SATELLITE SENSORS SUPPORTED

Below we list the satellite sensors for which the relative spectral responses have been included in Pyspectral.

Table 1: Satellite sensors supported

Satellite sensor	Filename	Link to the original RSR
Meteosat-8 seviri	*rsr_seviri_Meteosat-8.h5*	GSICS
Meteosat-9 seviri	*rsr_seviri_Meteosat-9.h5*	GSICS
Meteosat-10 seviri	*rsr_seviri_Meteosat-10.h5*	GSICS
Meteosat-11 seviri	*rsr_seviri_Meteosat-11.h5*	GSICS
GOES-16 abi	*rsr_abi_GOES-16.h5*	GOES-R
GOES-17 abi	*rsr_abi_GOES-17.h5*	GOES-S
GOES-18 abi	*rsr_abi_GOES-18.h5*	GOES-T
GOES-19 abi	*rsr_abi_GOES-19.h5*	GOES-U
Himawari-8 ahi	*rsr_ahi_Himawari-8.h5*	JMA
Himawari-9 ahi	*rsr_ahi_Himawari-9.h5*	JMA
GEO-KOMPSAT-2A ami	*rsr_ami_GEO-KOMPSAT-2A.h5*	NWPSAF-GeoKompsat-2A-ami
FY-4A agri	*rsr_agri_FY-4A.h5*	NSMC-fy4a
Envisat aatsr	*rsr_aatsr_Envisat.h5*	ESA-Envisat
TIROS-N to NOAA-19 avhrr	e.g. *rsr_avhrr3_NOAA-19.h5*	GSICS
Metop-A avhrr/3	*rsr_avhrr3_Metop-A.h5*	GSICS
Metop-B avhrr/3	*rsr_avhrr3_Metop-B.h5*	GSICS
Metop-C avhrr	*rsr_avhrr3_Metop-C.h5*	GSICS (Acquired via personal contact)
EOS-Terra modis	*rsr_modis_EOS-Terra.h5*	GSICS
EOS-Aqua modis	*rsr_modis_EOS-Aqua.h5*	GSICS
Sentinel-3A slstr	*rsr_slstr_Sentinel-3A.h5*	ESA-Sentinel-SLSTR
Sentinel-3B slstr	*rsr_slstr_Sentinel-3B.h5*	ESA-Sentinel-SLSTR
Sentinel-3A olci	*rsr_olci_Sentinel-3A.h5*	ESA-Sentinel-OLCI
Sentinel-3B olci	*rsr_olci_Sentinel-3B.h5*	ESA-Sentinel-OLCI
Sentinel-2A msi	*rsr_msi_Sentinel-2A.h5*	ESA-Sentinel-MSI
Sentinel-2B msi	*rsr_msi_Sentinel-2B.h5*	ESA-Sentinel-MSI
NOAA-20 viirs	*rsr_viirs_NOAA-20.h5*	NESDIS
Suomi-NPP viirs	*rsr_viirs_Suomi-NPP.h5*	GSICS
Landsat-8 oli	*rsr_oli_Landsat-8.h5*	NASA-Landsat-OLI
FY-3D mersi-2	*rsr_mersi-2_FY-3D.h5*	CMA (Acquired via personal contact)
HY-1C cocts	*rsr_cocts_HY-1C.h5*	(Acquired via personal contact)
Metop-SG-A1 MetImage	*rsr_metimage_Metop-SG-A1.h5*	NWPSAF-MetImage
Meteosat-12 fci	*rsr_fci_Meteosat-12.h5*	NWPSAF-Meteosat-12-fci
MTG-I1 fci (NB! Identical to Meteosat-12 fci)	*rsr_fci_MTG-I1.h5*	NWPSAF-Meteosat-12-fci

CHAPTER

TWO

INSTALLATION

Installation of the latest stable version is always done using:

```
$>  pip install pyspectral
```

Some static data are part of the package. Downloading of this data is handled automagically by the software when data are needed. However, the data can also be downloaded manually once and for all by calling dedicated download scripts. The latter is helpful when running PySpectral in an operational environment. See further below on how the static data downloads are handled.

You can also choose to download the PySpectral source code from github:

```
$> git clone git://github.com/pytroll/pyspectral.git
```

and then run:

```
$> python setup.py install
```

or, if you want to hack the package:

```
$> python setup.py develop
```

CHAPTER

THREE

STATIC DATA

PySpectral make use of and requires the access to two different kinds of static ancillary data sets. First, relative spectral responses for a large number of satellite imaging sensors are available in a unified hdf5 format. Secondly, PySpectral comes with a set of Look-Up-Tables (LUTs) for the atmospheric correction in the short wave spectral range.

Both these datasets downloads automagically from zenodo.org when needed.

On default these static data will reside in the platform specific standard destination for storing user data, via the use of the appdirs package. On Linux this will be under *~/.local/share/pyspectral*. See further down.

3.1 The spectral response data

PySpectral reads relative spectral response functions for various satellite sensors. The original agency specific spectral response data are stored in various different formats (e.g. ascii, excel, or netCDF), with varying levels of detailed information available, and usually not following any agreed international standard.

These data are often available at the satellite agencies responsible for the instrument in question. That is for example EUMETSAT, NOAA, NASA, JMA and CMA. The Global Space-based Inter-Calibration System (GSICS) Coordination Center (GCC) holds a place with links to several relevant instrument response data. But far from all data are available through that web-site.

Therefore, in order to make life easier for the *PySpectral* user we have defined one common internal HDF5 format. That way it is also easy to add support for new instruments. Currently the relative spectral responses for the following sensors are included:

- Suomi-NPP and NOAA-20 VIIRS
- All of the NOAA/TIROS-N and Metop AVHRRs
- Terra/Aqua MODIS
- Meteosat 8-11 SEVIRI
- Sentinel-3A/3B SLSTR and OLCI
- Envisat AATSR
- GOES-16/17 ABI
- Himawari-8/9 AHI
- Sentinel-2 A&B MSI
- Landsat-8 OLI
- Geo-Kompsat-2A / AMI
- Fengyun-4A / AGRI

The data are automagically downloaded and installed when needed. But if you need to specifically retrieve the data independently the data are available from the pyspectral rsr repository and can be downloaded using a script that comes with *PySpectral*. For instance, to download the data into the default directory:

```
python ~/.local/bin/download_rsr.py
```

Instead if you want to download the data to a specific directory and using verbose mode (to get some log information on the screen):

```
python ~/.local/bin/download_rsr.py -v -o /tmp
```

It is still also possible to download the original spectral responses from the various satellite operators instead and generate the internal HDF5 formatted files yourself. However, this should normally never be needed. (For SEVIRI on Meteosat download the data from eumetsat and unzip the Excel file.)

3.2 Look-Up-Tables for atmospheric correction in the SW spectral range

Look-Up-Tables (LUTs) with simulated black surface top of atmosphere reflectances over the $400-600nm$ wavelength spectrum for various aerosol distributions and a set of standard atmospheres for varying sun-satellite viewing are available in HDF5 on zenodo.org. The LUTs are downloaded automagically when needed and placed in the same directory structure as the spectral response data (see above). The data can also be downloaded manually using a dedicated download script. To download all LUTs you can do like this:

```
python ~/.local/bin//download_atm_correction_luts.py
```

If you only need LUTs for a few aerosol distributions, for example *desert aerosols* and *marine clean aerosols* (and using verbose mode to get some log info on the screen):

```
python ~/.local/bin//download_atm_correction_luts.py -a desert_aerosol marine_
↪clean_aerosol -v
```

3.3 Configuration file

A default configuration file *pyspectral.yaml* is installed automatically and being part of the package under the *etc* directory. In many cases the default settings will allow one to do all what is needed. However, it can easily be overwritten by making your own copy and set an environment variable pointing to this configuration file, as e.g.:

```
$> PSP_CONFIG_FILE=/home/a000680/pyspectral.yaml; export PSP_CONFIG_FILE
```

So, in case you want to download the internal *PySpectral* formatted relative spectral responses as well as the atmospheric correction LUTs once and for all, and keep them somewhere else. Change the configuration in *pyspectral.yaml* so it looks something like this:

```
rsr_dir = /path/to/internal/rsr_data
rayleigh_dir = /path/to/rayleigh/correction/luts
download_from_internet = True
```

Then download the data:

```
python ~/.local/bin/download_rsr.py
```

```
python ~/.local/bin//download_atm_correction_luts.py
```

And then adjust the *pyspectral.yaml* so data downloading will not be attempted anymore:

```
rsr_dir = /path/to/internal/rsr_data
rayleigh_dir = /path/to/rayleigh/correction/luts
download_from_internet = False
```

CHAPTER

FOUR

USAGE

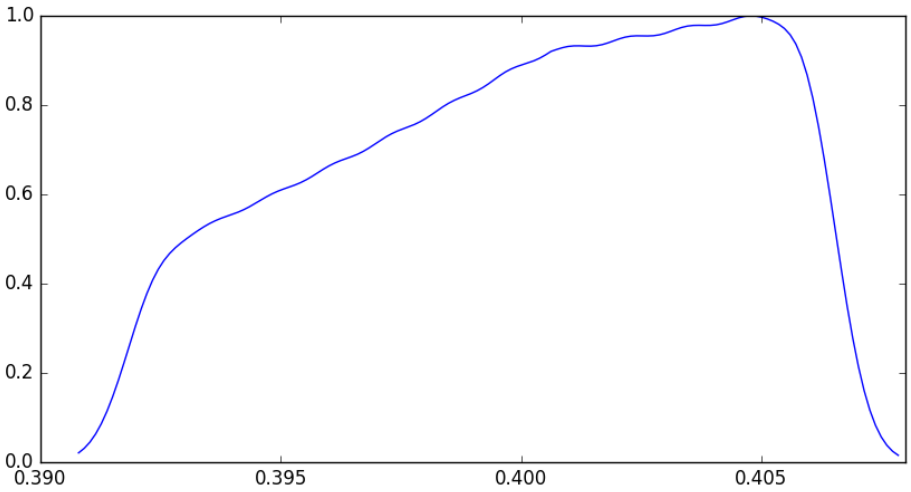
Get the spectral responses for Sentinel-3A OLCI (and turn on debugging to see more info on what is being done behind the curtan):

```
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> from pyspectral.utils import debug_on
>>> debug_on()
>>> olci = RelativeSpectralResponse('Sentinel-3A', 'olci')
```

You will see that if you haven't run this kind of code before, pyspectral will download the spectral responses for all satellites supported from Zenodo.

Now, you can work with the data as you wish, make some simple plot for instance:

```
>>> [str(b) for b in olci.band_names]
['Oa01', 'Oa02', 'Oa03', 'Oa04', 'Oa05', 'Oa06', 'Oa07', 'Oa08', 'Oa09', 'Oa10
↪', 'Oa11', 'Oa12', 'Oa13', 'Oa14', 'Oa15', 'Oa16', 'Oa17', 'Oa18', 'Oa19',
↪'Oa20', 'Oa21']
>>> print("Central wavelength = {wvl:7.6f}".format(wvl=olci.rsr['Oa01']['det-1
↪']['central_wavelength']))
Central wavelength = 0.400303
>>> import matplotlib.pyplot as plt
>>> dummy = plt.figure(figsize=(10, 5))
>>> import numpy as np
>>> rsr = olci.rsr['Oa01']['det-1']['response']
>>> resp = np.where(rsr < 0.015, np.nan, rsr)
>>> wl_ = olci.rsr['Oa01']['det-1']['wavelength']
>>> wvl = np.where(np.isnan(resp), np.nan, wl_)
>>> dummy = plt.plot(wvl, resp)
>>> plt.show()
```



A simple use case:

```
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> from pyspectral.solar import SolarIrradianceSpectrum
>>> modis = RelativeSpectralResponse('EOS-Aqua', 'modis')
>>> solar_irr = SolarIrradianceSpectrum(dlambda=0.005)
>>> sflux = solar_irr.inband_solarflux(modis.rsr['20'])
>>> print("Solar flux over Band: {sflux}".format(sflux=round(sflux, 6)))
Solar flux over Band: 2.002928
```

And, here is how to derive the solar reflectance (removing the thermal part) of the Aqua MODIS 3.7 micron band:

```
>>> from pyspectral.near_infrared_reflectance import Calculator
>>> import numpy as np
>>> sunz = np.array([80.])
>>> tb3 = np.array([290.0])
>>> tb4 = np.array([282.0])
>>> refl37 = Calculator('EOS-Aqua', 'modis', '20', detector='det-1', solar_flux=2.
↪0029281634299041)
>>> print("Reflectance = {r:7.6f}".format(r=np.ma.round(refl37.reflectance_from_tbs(sunz,
↪ tb3, tb4), 6)[0]))
Reflectance = 0.251249
```

And, here is how to derive the atmospheric (Rayleigh scattering by atmospheric molecules and atoms as well as Mie scattering and absorption of aerosols) contribution in a short wave band:

```
>>> from pyspectral import rayleigh
>>> rcor = rayleigh.Rayleigh('GOES-16', 'abi')
>>> import numpy as np
>>> sunz = np.array([[50., 60.], [51., 61.]])
>>> satz = np.array([[40., 50.], [41., 51.]])
>>> azidiff = np.array([[160, 160], [160, 160]])
>>> redband =  np.array([[0.1, 0.15], [0.11, 0.16]])
>>> refl = rcor.get_reflectance(sunz, satz, azidiff, 'ch2', redband)
>>> print([np.round(r, 6) for r in refl.ravel()])
[3.254637, 6.488645, 3.434351, 7.121556]
```

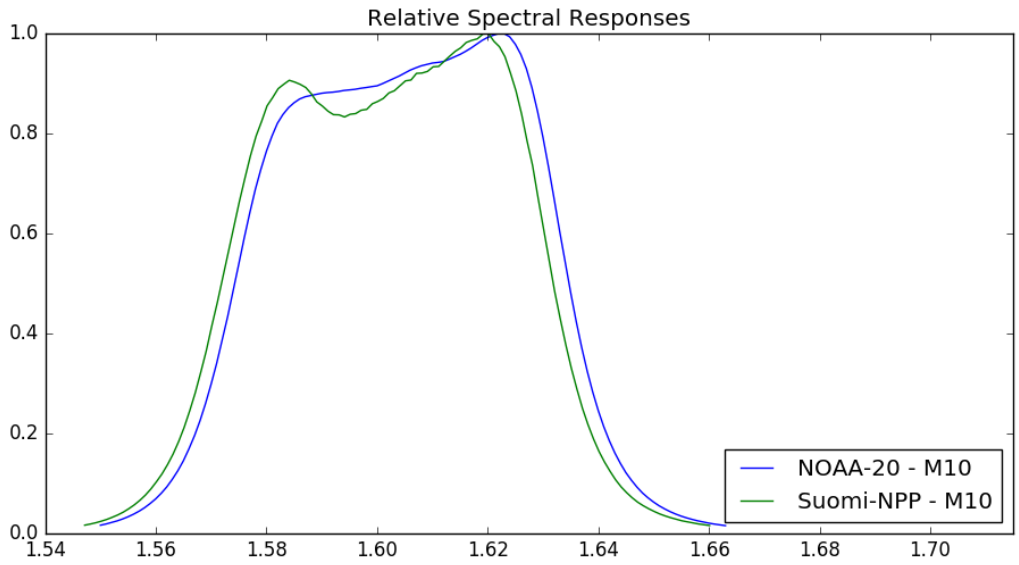
CHAPTER

FIVE

SIMPLE PLOTTING OF SPECTRAL RESPONSES

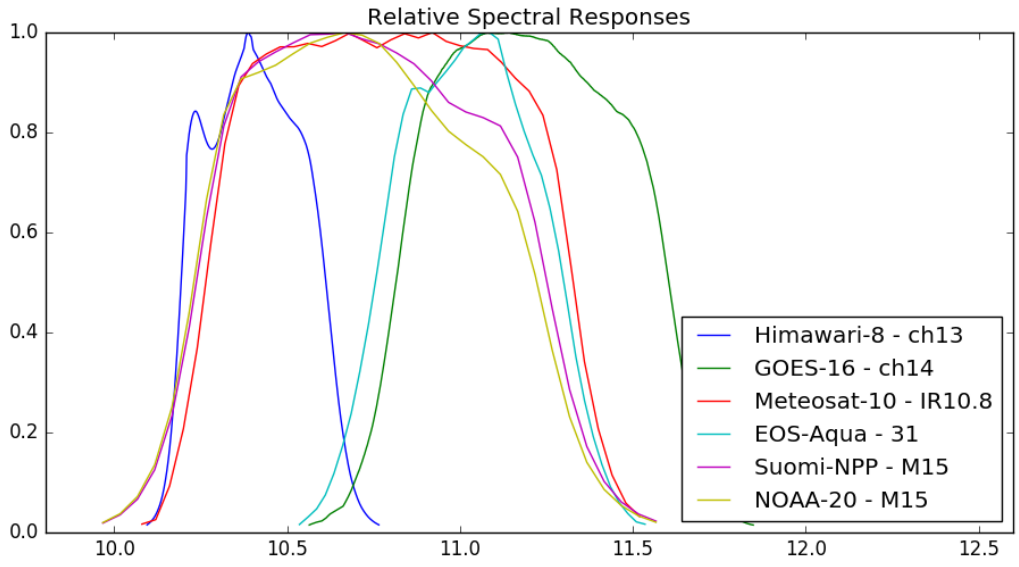
Plot VIIRS spectral responses (detector 1 only) for band M10 on JPSS-1 and Suomi-NPP:

```
python composite_rsr_plot.py -p NOAA-20 Suomi-NPP -s viirs -b M10
```



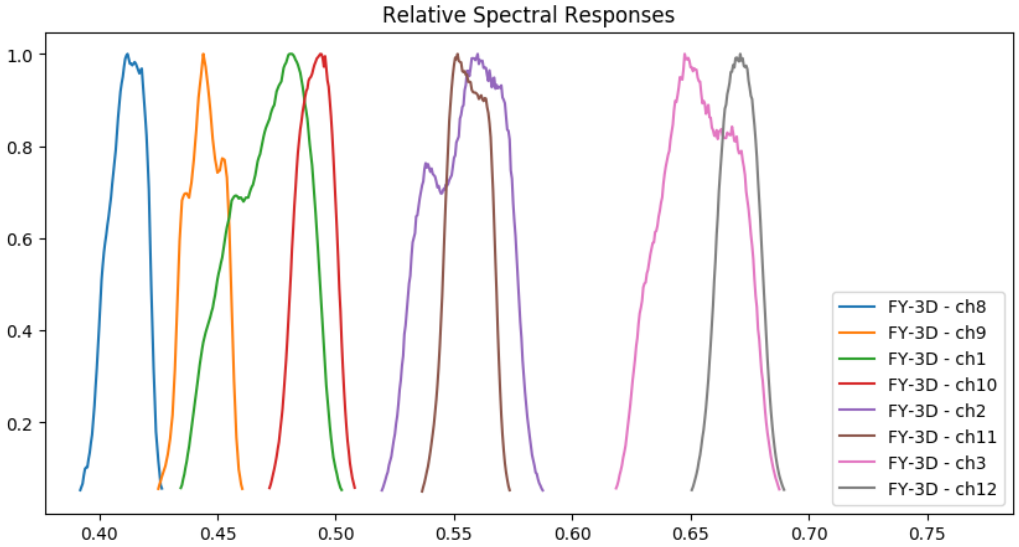
Plot relative spectral responses for the spectral channel closest to the $10.8\mu m$ for several platforms and sensors:

```
python composite_rsr_plot.py --platform_name Himawari-8 GOES-16 Meteosat-10 EOS-Aqua␣
↪Sentinel-3A Suomi-NPP NOAA-20 --sensor ahi abi seviri modis olci slstr viirs --
↪wavelength 10.8
```



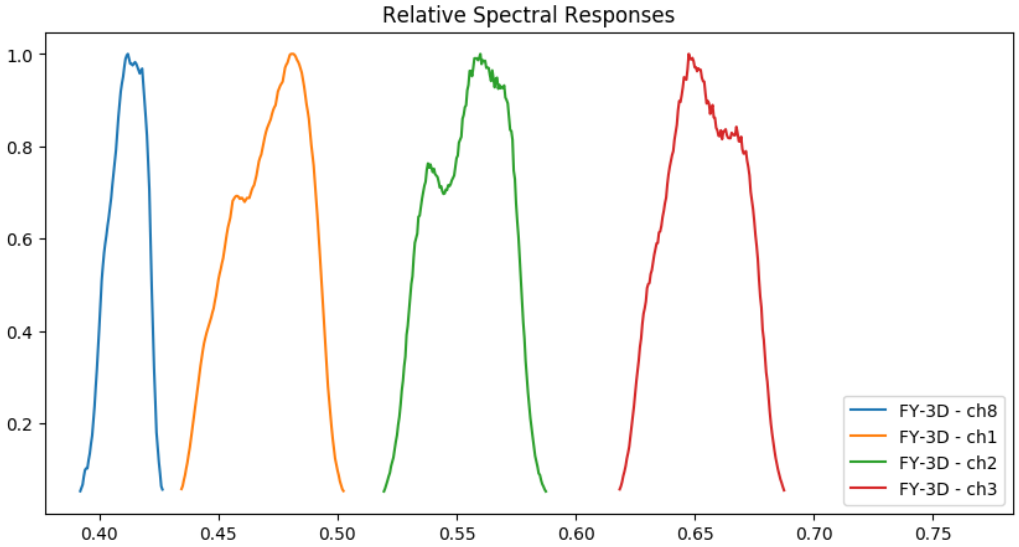
Some sensor bands are quite close, and that requires the search for bands in the spectral range to use rather small wavelengths increments. Therefore you might experience the plotting to a bit slow on default. Here an example with MERSI-2 on FY-3D:

```
python bin/composite_rsr_plot.py -p FY-3D -s mersi-2 -r 0.4 0.7 -t 0.05
```



It is possible to specify a different wavelength resolution/increments by using a flag. However, when you do that it might affect how pyspectral identify bands. If the resolution is too coarse several close bands may be considered one and the same. In the below case it is probably not a good idea to lower the resolution as one can see (several MESI-2 bands are now missing):

```
python bin/composite_rsr_plot.py -p FY-3D -s mersi-2 -r 0.4 0.7 -t 0.05 --wavelength_
↪resolution 0.05
```


Relative Spectral Responses
1.0
0.8
0.6
0.4
0.2
0.40
0.45
0.50
0.55
0.60
0.65
0.70
0.75
FY-3D - ch8
FY-3D - ch1
FY-3D - ch2
FY-3D - ch3

CHAPTER

SIX

EXAMPLE WITH SEVIRI DATA

Let us try calculate the 3.9 micron reflectance for Meteosat-10:

```
>>> sunz = 80.
>>> tb3 = 290.0
>>> tb4 = 282.0
>>> from pyspectral.near_infrared_reflectance import Calculator
>>> refl39 = Calculator('Meteosat-10', 'seviri', 'IR3.9')
>>> print('{refl:4.3f}'.format(refl=refl39.reflectance_from_tbs(sunz, tb3, tb4)[0]))
0.555
```

You can also provide the in-band solar flux from outside when calculating the reflectance, saving a few milliseconds per call:

```
>>> from pyspectral.solar import (SolarIrradianceSpectrum, TOTAL_IRRADIANCE_SPECTRUM_
↪2000ASTM)
>>> solar_irr = SolarIrradianceSpectrum(TOTAL_IRRADIANCE_SPECTRUM_2000ASTM, dlambda=0.
↪0005)
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> seviri = RelativeSpectralResponse('Meteosat-10', 'seviri')
>>> sflux = solar_irr.inband_solarflux(seviri.rsr['IR3.9'])
>>> refl39 = Calculator('Meteosat-10', 'seviri', 'IR3.9', solar_flux=sflux)
>>> print('{refl:4.3f}'.format(refl=refl39.reflectance_from_tbs(sunz, tb3, tb4)[0]))
0.555
```

By default the data are masked outside the default Sun zenith-angle (SZA) correction limit (85.0 degrees). The masking can be adjusted via *masking_limit* keyword argument to *Calculator*, and turned of by defining *Calculator(..., masking_limit=None)*. The SZA limit can be adjusted via *sunz_threshold* keyword argument: *Calculator(..., sunz_threshold=88.0)*.

6.1 Integration with SatPy

The SatPy package integrates PySpectral so that it is very easy with only a very few lines of code to make RGB images with the 3.9 reflectance as one of the bands. Head to the PyTroll gallery pages for examples. SatPy for instance has a *snow* RGB using the 0.8 micron, the 1.6 micron and the 3.9 micron reflectance derived using PySpectral.

CHAPTER

SEVEN

DEFINITIONS AND SOME RADIATION THEORY

In radiation physics there is unfortunately several slightly different ways of presenting the theory. For instance, there is no single custom on the mathematical symbolism, and various different non SI-units are used in different situations. Here we present just a few terms and definitions with relevance to PySpectral, and how to possible go from one common representation to another.

7.1 Symbols and definitions used in PySpectral

λ	Wavelength (μm)
$\nu = \frac{1}{\lambda}$	Wavenumber (cm^{-1})
λ_c	Central wavelength for a given band/channel (μm)
ν_c	Central wavelength for a given band/channel (cm^{-1})
$\Phi_i(\lambda)$	Relative spectral response for band i as a function of wavelength
E_λ	Spectral irradiance at wavelength λ ($W/m^2\mu m^{-1}$)
E_ν	Spectral irradiance at wavenumber ν ($W/m^2(cm^{-1})^{-1}$)
B_λ	Blackbody radiation at wavelength λ ($W/m^2\mu m^{-1}$)
B_ν	Blackbody radiation at wavenumber ν ($W/m^2(cm^{-1})^{-1}$)
L_ν	Spectral radiance at wavenumber ν ($W/m^2sr^{-1}(cm^{-1})^{-1}$)
L_λ	Spectral radiance at wavelength λ ($W/m^2sr^{-1}\mu m^{-1}$)
L	Radiance - (band) integrated spectral radiance (W/m^2sr^{-1})

7.2 Constants

k_B	Boltzmann constant ($1.38064881e-23$)
h	Planck constant ($6.626069571e-34$)
c	Speed of light in vacuum ($2.997924581e8$)

7.3 Central wavelength and central wavenumber

The central wavelength for a given spectral band (i) of a satellite sensor is defined as:

$$\lambda_{ci} = \frac{\int_0^\infty \Phi_i(\lambda)\lambda \mathrm{d}\lambda}{\int_0^\infty \Phi_i(\lambda)\mathrm{d}\lambda}$$

Likewise the central wavenumber is:

$$\nu_{ci} = \frac{\int_0^\infty \Phi_i(\nu)\nu \mathrm{d}\nu}{\int_0^\infty \Phi_i(\nu)\mathrm{d}\nu}$$

And since $\nu = 1/\lambda$, we can express the central wavenumber using the spectral response function expressed in wavelength space, as:

$$\nu_{ci} = \frac{\int_0^\infty \Phi_i(\lambda)\frac{1}{\lambda^3} \mathrm{d}\lambda}{\int_0^\infty \Phi_i(\lambda)\frac{1}{\lambda^2}\mathrm{d}\lambda}$$

And from this we see that in general $\nu_c \neq 1/\lambda_c$.

Taking SEVIRI as an example, and looking at the visible channel on Meteosat-8, we see that this is indeed true:

```
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> from pyspectral.utils import convert2wavenumber, get_central_wave
>>> seviri = RelativeSpectralResponse('Meteosat-8', 'seviri')
>>> cwl = get_central_wave(seviri.rsr['VIS0.6']['det-1']['wavelength'], seviri.rsr['VIS0.
↪6']['det-1']['response'])
>>> print(round(cwl, 6))
0.640216
>>> rsr, info = convert2wavenumber(seviri.rsr)
>>> print("si_scale={scale}, unit={unit}".format(scale=info['si_scale'], unit=info['unit
↪']))
si_scale=100.0, unit=cm-1
>>> wvc = get_central_wave(rsr['VIS0.6']['det-1']['wavenumber'], rsr['VIS0.6']['det-1'][
↪'response'])
>>> round(wvc, 3)
15682.622
>>> print(round(1./wvc*1e4, 6))
0.637648
```

In the PySpectral unified HDF5 formatted spectral response data we also store the central wavelength, so you actually don't have to calculate them yourself:

```
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> print("Central wavelength = {cwl}".format(cwl=round(seviri.rsr['VIS0.6']['det-1'][
↪'central_wavelength'], 6)))
Central wavelength = 0.640216
```

7.4 Wavelength range

Satellite radiometers do not measure outgoing radiance at only the central wavelength, but over a range of wavelengths defined by the spectral response function of the instrument. A given instrument channel is therefore often defined by a wavelength range in the form $\lambda_{min}, \lambda_c, \lambda_{max}$. Where λ_{min} and λ_{max} are defined by the first and last points in the spectral response function where the response is greater than a given threshold. Pyspectral has a utility function to enable users to compute these wavelength ranges:

```
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> from pyspectral.utils import get_wave_range
>>> seviri = RelativeSpectralResponse('Meteosat-8', 'seviri')
>>> wvl_range = get_wave_range(seviri.rsr['HRV']['det-1'], 0.15)
>>> print(round(num, 3) for num in wvl_range)
[0.456, 0.715, 0.996]
```

7.5 Spectral Irradiance

We denote the spectral irradiance E which is a function of wavelength or wavenumber, depending on what representation is used. In PySpectral the aim is to support both representations. The units are of course dependent of which representation is used.

In wavelength space we write $E(\lambda)$ and it is given in units of $W/m^2\mu m^{-1}$.

In wavenumber space we write $E(\nu)$ and it is given in units of $W/m^2(cm^{-1})^{-1}$.

To convert a spectral irradiance E_{λ_0} at wavelength λ_0 to a spectral irradiance E_{ν_0} at wavenumber $\nu_0 = 1/\lambda_0$ the following relation applies:

$$E_\nu = E_\lambda \lambda^2$$

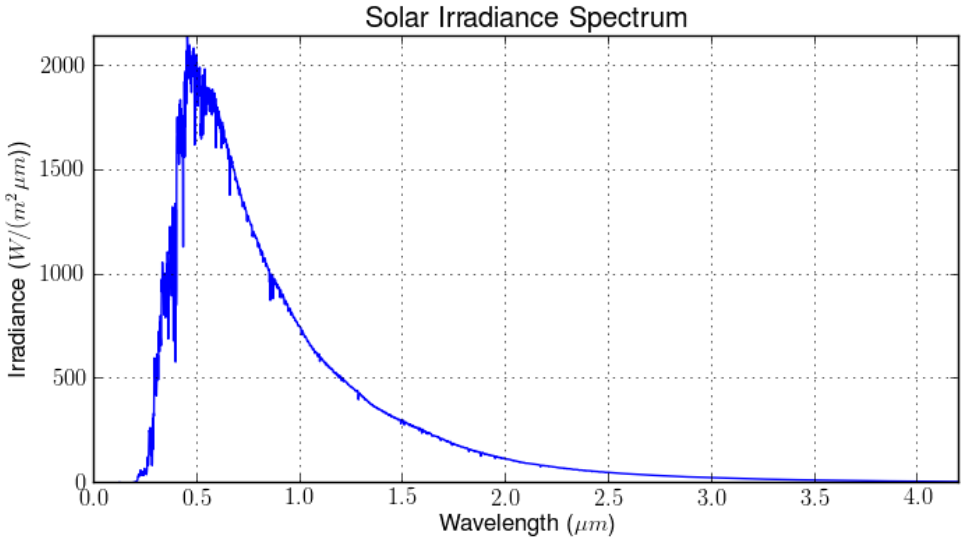
And if the units are not SI but rather given by the units shown above we have to account for a factor of 10 as:

$$E_\nu = E_\lambda \lambda^2 * 0.1$$

7.6 TOA Solar irradiance and solar constant

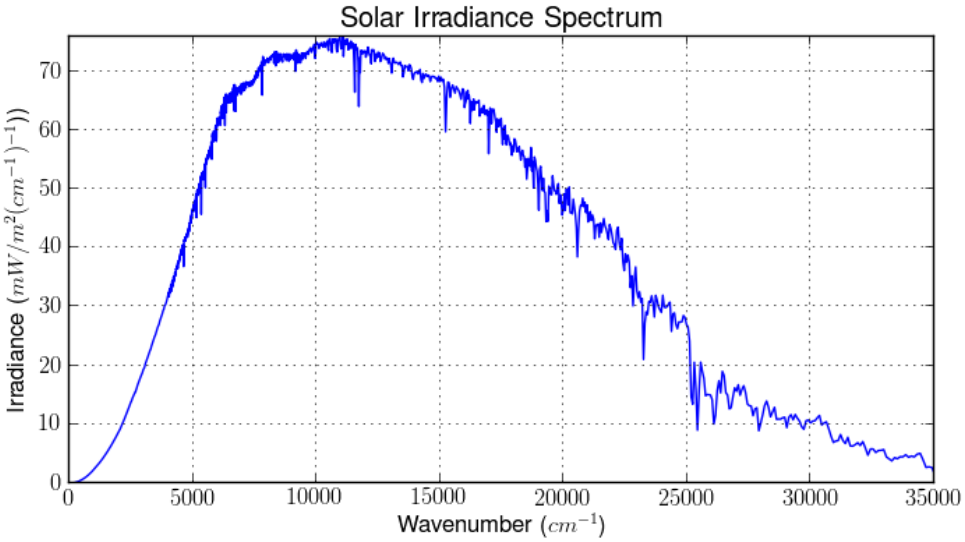
First, the TOA solar irradiance in wavelength space:

```
>>> from pyspectral.solar import SolarIrradianceSpectrum
>>> solar_irr = SolarIrradianceSpectrum(dlambda=0.0005)
>>> print("Solar irradiance = {}".format(round(solar_irr.solar_constant(), 3)))
Solar irradiance = 1366.091
>>> solar_irr.plot('/tmp/solar_irradiance.png')
```



The solar constant is in units of W/m^2. Instead when expressing the irradiance in wavenumber space using wavenumbers in units of cm^{-1} the solar flux is in units of mW/m^2:

```
>>> solar_irr = SolarIrradianceSpectrum(TOTAL_IRRADIANCE_SPECTRUM_2000ASTM,␣
↪dlambda=0.0005, wavespace='wavenumber')
>>> print(round(solar_irr.solar_constant(), 5))
1366077.16482
>>> solar_irr.plot('/tmp/solar_irradiance_wnum.png')
```



7.7 In-band solar flux

The solar flux (SI unit $\frac{W}{m^2}$) over a spectral sensor band can be derived by convolving the top of atmosphere solar spectral irradiance and the sensor relative spectral response. For band i:

$$F_i = \int_0^\infty \Phi_i(\lambda) E(\lambda) \mathrm{d}\lambda$$

where $E(\lambda)$ is the TOA spectral solar irradiance at a sun-earth distance of one astronomical unit (AU).

In python code it may look like this:

```
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> from pyspectral.utils import convert2wavenumber, get_central_wave
>>> seviri = RelativeSpectralResponse('Meteosat-8', 'seviri')
>>> rsr, info = convert2wavenumber(seviri.rsr)
>>> from pyspectral.solar import SolarIrradianceSpectrum
>>> solar_irr = SolarIrradianceSpectrum(dlambda=0.0005, wavespace='wavenumber')
>>> print("Solar Irradiance (SEVIRI band VIS008) = {sflux:12.6f}".format(sflux=solar_irr.
↪inband_solarflux(rsr['VIS0.8'])))
Solar Irradiance (SEVIRI band VIS008) = 63767.908405
```

7.8 Planck radiation

Planck's law describes the electromagnetic radiation emitted by a black body in thermal equilibrium at a definite temperature.

Thus for wavelength λ the Planck radiation or Blackbody radiation $B(\lambda)$ can be written as:

$$B_\lambda(T) = \frac{2hc^2}{\lambda^5} \frac{1}{e^{\frac{hc}{\lambda k_B T}} - 1}$$

and expressed as a function of wavenumber ν:

$$B_\nu(T) = 2hc^2\nu^3 \frac{1}{e^{\frac{hc\nu}{k_B T}} - 1}$$

In python it may look like this:

```
>>> from pyspectral.blackbody import blackbody_wn
>>> wavenumber = 90909.1
>>> rad = blackbody_wn((wavenumber, ), [300., 301])
>>> print("{0:7.6f} {1:7.6f}".format(rad[0], rad[1]))
0.001158 0.001175
```

Which are the spectral radiances in SI units at wavenumber around $909 cm^{-1}$ at temperatures 300 and 301 Kelvin. In units of $mW/m^2(cm^{-1})^{-1}\ sr^{-1}$ this becomes:

```
>>> print("{0:7.4f} {1:7.4f}".format((rad*1e+5)[0], (rad*1e+5)[1]))
115.8354 117.5477
```

And using wavelength representation:

```
>>> from pyspectral.blackbody import blackbody
>>> wvl = 1./wavenumber
>>> rad = blackbody(wvl, [300., 301])
>>> print("{0:10.3f} {1:10.3f}".format(rad[0], rad[1]))
9573177.494 9714687.157
```

Which are the spectral radiances in SI units around $11\mu m$ at temperatures 300 and 301 Kelvin. In units of $mW/m^2\ m^{-1}sr^{-1}$ this becomes:

```
>>> print("{0:7.5f} {1:7.5f}".format((rad*1e-6)[0], (rad*1e-6)[1]))
9.57318 9.71469
```

7.9 The inverse Planck function

Inverting the Planck function allows to derive the brightness temperature given the spectral radiance. Expressed in wavenumber space this becomes:

$$T_B = T(B_\nu) = \frac{hc\nu}{k_B} log^{-1}\{\frac{2hc^2\nu^3}{B_\nu} + 1\}$$

With the spectral radiance given as a function of wavelength the equation looks like this:

$$T_B = T(B_\lambda) = \frac{hc}{\lambda k_B} log^{-1}\{\frac{2hc^2}{B_\lambda \lambda^5} + 1\}$$

In python it may look like this:

```
>>> from pyspectral.blackbody import blackbody_wn_rad2temp
>>> wavenumber = 90909.1
>>> temp = blackbody_wn_rad2temp(wavenumber, [0.001158354, 0.001175477])
>>> print([round(t, 8) for t in temp])
[299.99998562, 301.00000518]
```

This approach only works for monochromatic or very narrow bands for which the spectral response function is assumed to be constant. In reality, typical imager channels are not that narrow and the spectral response function is not constant over the band. Here it is not possible to de-convolve the planck function and the spectral response function without knowing both, the spectral radiance and spectral response function in high spectral resolution. While this information is usually available for the spectral response function, there is only one integrated radiance per channel. That makes the derivation of brightness temperature from radiance more complicated and more time consuming - in preparation or in execution. Depending on individual requirements, there is a bunch of feasible solutions:

7.9.1 Iterative Method

A stepwise approach, which starts with a guess (most common temperature), calculate the radiance that would correspond to that temperature and compare it with the measured radiance. If the difference lies above a certain threshold, adjust the temperature accordingly and start over again:

(i) set uncertainty parameter ΔL

(ii) set $T_j = T_{firstguess}$

(iii) calculate $B(T_j)$

(iv) if $(B(T_j) - L_{measure}) > \Delta L$ then adjust T_j and go back to iii

(v) T_j matches the measurement within the defined uncertainty

Advantages

- no pre-computations
- accuracy easily adaptable to purpose
- memory friendly
- independent of band
- independent of spectral response function

Disadvantages

- slow, especially when applying to wide bands and high accuracy requirements
- redundant calculations when applying to images with many pixels

7.9.2 Function Fit

Another feasible approach is to fit a function Φ in a way that $|T - \Phi(L_{measure})|$ minimizes. This requires pre-calculations of data pairs T and $L(T)$. Finally an adequate function Φ (dependent on the shape of $T(L(T))$) is assigned and used to calculate the brightness temperature for one channel.

Advantages

- fast approach (especially in execution)
- minor memory request (one function per channel)

Disadvantages

- accuracy determined in the beginning of the process
- complexity of Φ depends on $T(L(T))$

7.9.3 Look-Up Table

If the number of possible pairs T and $L(T)$ is limited (e.g. due to limited bit size) or if the setting for a function fit is too complex or does not fit into a processing environment, it is possible to just expand the number of pre-calculated pairs to a look-up table. In an optimal case, the table cover every possible value or is dense enough to allow for linear interpolation.

Advantages

- fast approach (but depends on table size)
- (almost) independent of function

Disadvantages

- accuracy dependent on value density (size of look-up table)
- can become a memory issue

CHAPTER

EIGHT

DERIVATION OF THE 3.7 MICRON REFLECTANCE

It is well known that the top of atmosphere signal (observed radiance or brightness temperature) of a sensor band in the near infrared part of the spectrum between around $3-4\mu m$ is composed of a thermal (or emissive) part and a part stemming from reflection of incoming sunlight.

With some assumptions it is possible to separate the two and derive a solar reflectance in the band from the observed brightness temperature. Below we will demonstrate the theory on how this separation is done. But, first we need to demonstrate how the spectral radiance can be calculated from an observed brightness temperature, knowing the relative spectral response of the the sensor band.

8.1 Brightness temperature to spectral radiance

If the satellite observation is given in terms of the brightness temperature, then the corresponding spectral radiance can be derived by convolving the relative spectral response with the Planck function and divding by the equivalent band width:

$$L_{3.7} = \frac{\int_0^\infty \Phi_{3.7}(\lambda) B_\lambda(T_{3.7}) \mathrm{d}\lambda}{\widetilde{\Delta\lambda}} \tag{8.1}$$

where the equivalent band width $\widetilde{\Delta\lambda}$ is defined as:

$$\widetilde{\Delta\lambda} = \int_0^\infty \Phi_{3.7}(\lambda) \mathrm{d}\lambda$$

$L_{3.7}$ is the measured radiance at $3.7\mu m$, $\Phi_{3.7}(\lambda)$ is the $3.7\mu m$ channel spectral response function, and B_λ is the Planck radiation.

This gives the spectral radiance given the brightness temperature and may be expressed in $W/m^2 sr^{-1} \mu m^{-1}$, or using SI units $W/m^2 sr^{-1} m^{-1}$.

```
>>> from pyspectral.radiance_tb_conversion import RadTbConverter
>>> import numpy as np
>>> sunz = np.array([68.98597217,  68.9865146 ,  68.98705756,  68.98760105, 68.98814508])
>>> tb37 = np.array([298.07385254, 297.15478516, 294.43276978, 281.67633057, 273.
↪7923584])
>>> viirs = RadTbConverter('Suomi-NPP', 'viirs', 'M12')
>>> rad37 = viirs.tb2radiance(tb37)
>>> print([np.round(rad, 7) for rad in rad37['radiance']])
[369717.4972296, 355110.6414922, 314684.3507084, 173143.4836477, 116408.0022674]
>>> rad37['unit']
'W/m^2 sr^-1 m^-1'
```

In order to get the total radiance over the band one has to multiply with the equivalent band width.

```
>>> from pyspectral.radiance_tb_conversion import RadTbConverter
>>> import numpy as np
>>> tb37 = np.array([298.07385254, 297.15478516, 294.43276978, 281.67633057, 273.
↪7923584])
>>> viirs = RadTbConverter('Suomi-NPP', 'viirs', 'M12')
>>> rad37 = viirs.tb2radiance(tb37, normalized=False)
>>> print([np.round(rad, 8) for rad in rad37['radiance']])
[0.07037968, 0.06759911, 0.05990353, 0.03295971, 0.02215951]
>>> rad37['unit']
'W/m^2 sr^-1'
```

By passing `normalized=False` to the method the division by the equivalent band width is omitted. The equivalent width is provided as an attribute in SI units (m):

```
>>> from pyspectral.radiance_tb_conversion import RadTbConverter
>>> viirs = RadTbConverter('Suomi-NPP', 'viirs', 'M12')
>>> viirs.rsr_integral
1.903607e-07
```

Inserting the Planck radiation:

$$L_{3.7} = \frac{2hc^2 \int_0^\infty \frac{\Phi_{3.7}(\lambda)}{\lambda^5} \frac{\mathrm{d}\lambda}{e^{\frac{hc}{\lambda k_B (T_{3.7})}} - 1}}{\widetilde{\Delta\lambda}} \tag{8.2}$$

The total band integrated spectral radiance or the in band radiance is then:

$$L_{3.7} = 2hc^2 \int_0^\infty \frac{\Phi_{3.7}(\lambda)}{\lambda^5} \frac{\mathrm{d}\lambda}{e^{\frac{hc}{\lambda k_B (T_{3.7})}} - 1} \tag{8.3}$$

This is expressed in wavelength space. But the spectral radiance can also be given in terms of the wavenumber ν, provided the relative spectral response is given as a function of ν:

$$L_{\nu(3.7)} = \frac{\int_0^\infty \Phi_{3.7}(\nu) B_\nu(T_{3.7}) \mathrm{d}\nu}{\widetilde{\Delta\nu}}$$

where the equivalent band width $\widetilde{\Delta\nu}$ is defined as:

$$\widetilde{\Delta\nu} = \int_0^\infty \Phi_{3.7}(\nu) \mathrm{d}\nu$$

and inserting the Planck radiation:

$$L_{\nu(3.7)} = \frac{\frac{2h}{c^2} \int_0^\infty \Phi_{3.7}(\nu) \frac{\nu^3 \mathrm{d}\nu}{e^{\frac{hc}{\lambda k_B T_{3.7}}} - 1}}{\widetilde{\Delta\nu}}$$

8.2 Determination of the in-band solar flux

The solar flux (SI unit $\frac{W}{m^2}$) over a spectral sensor band can be derived by convolving the top of atmosphere spectral irradiance and the sensor relative spectral response curve, so for the $3.7 \mu m$ band this would be:

$$F_{3.7} = \int_0^\infty \Phi_{3.7}(\lambda) S(\lambda) \mathrm{d}\lambda \tag{8.4}$$

where $S(\lambda)$ is the spectral solar irradiance.

```
>>> from pyspectral.rsr_reader import RelativeSpectralResponse
>>> from pyspectral.solar import SolarIrradianceSpectrum
>>> viirs = RelativeSpectralResponse('Suomi-NPP', 'viirs')
>>> solar_irr = SolarIrradianceSpectrum(dlambda=0.005)
>>> sflux = solar_irr.inband_solarflux(viirs.rsr['M12'])
>>> print(np.round(sflux, 7))
2.2428119
```

8.3 Derive the reflective part of the observed 3.7 micron radiance

The monochromatic reflectivity (or reflectance) ρ_λ is the ratio of the reflected (backscattered) radiance to the incident radiance. In the case of solar reflection one can write:

$$\rho_\lambda = \frac{L_\lambda}{\mu_0 L_{\lambda 0}}$$

where L_λ is the measured radiance, $L_{\lambda 0}$ is the incoming solar radiance, and μ_0 is the cosine of the solar zenith angle θ_0.

Assuming the solar radiance is independent of direction, the equation for the reflectance can be written in terms of the solar flux $F_{\lambda 0}$:

$$\rho_\lambda = \frac{L_\lambda}{\frac{1}{\pi}\mu_0 F_{\lambda 0}}$$

For the $3.7\mu m$ channel the outgoing radiance is due to solar reflection and thermal emission. Thus in order to determine a $3.7\mu m$ channel reflectance, it is necessary to subtract the thermal part from the satellite signal. To do this, the temperature of the observed object is needed. The usual candidate at hand is the $11\mu m$ brightness temperature (e.g. VIIRS I5 or M12), since most objects behave approximately as blackbodies in this spectral interval.

The $3.7\mu m$ channel reflectance may then be written as (we now operate with the in band radiance given by (8.3))

$$\rho_{3.7} = \frac{L_{3.7} - \epsilon_{3.7}\int_0^\infty \Phi_{3.7}(\lambda) B_\lambda(T_{11}) \mathrm{d}\lambda}{\frac{1}{\pi}\mu_0 F_{3.7,0}}$$

where $L_{3.7}$ is the measured radiance at $3.7\mu m$, $\Phi_{3.7}(\lambda)$ is the $3.7\mu m$ channel spectral response function, B_λ is the Planck radiation, and T_{11} is the $11\mu m$ channel brightness temperature. Observe that $L_{3.7}$ is now the radiance provided by (8.3).

If the observed object is optically thick (transmittance equals zero) then:

$$\epsilon_{3.7} = 1 - \rho_{3.7}$$

and then, with the radiance $L_{3.7}$ derived using (8.2) and the solar flux given by (8.4) we get:

$$\rho_{3.7} = \frac{L_{3.7} - \int_0^\infty \Phi_{3.7}(\lambda) B_\lambda(T_{11}) \mathrm{d}\lambda}{\frac{1}{\pi}\mu_0 F_{3.7,0} - \int_0^\infty \Phi_{3.7}(\lambda) B_\lambda(T_{11}) \mathrm{d}\lambda} \tag{8.5}$$

In Python this becomes:

```
>>> from pyspectral.near_infrared_reflectance import Calculator
>>> import numpy as np
>>> refl_m12 = Calculator('Suomi-NPP', 'viirs', 'M12')
>>> sunz = np.array([68.98597217,  68.9865146 ,  68.98705756,  68.98760105, 68.98814508])
```

(continues on next page)

(continued from previous page)

```
>>> tb37 = np.array([298.07385254, 297.15478516, 294.43276978, 281.67633057, 273.
↪7923584])
>>> tb11 = np.array([271.38806152, 271.38806152, 271.33453369, 271.98553467, 271.
↪93609619])
>>> m12r = refl_m12.reflectance_from_tbs(sunz, tb37, tb11)
>>> print(np.any(np.isnan(m12r)))
False
>>> print([np.round(refl, 6) for refl in m12r])
[0.214329, 0.202852, 0.17064, 0.054089, 0.008381]
```

We can try decompose equation (8.5) above using the example of VIIRS M12 band:

```
>>> from pyspectral.radiance_tb_conversion import RadTbConverter
>>> import numpy as np
>>> sunz = np.array([68.98597217,  68.9865146 ,  68.98705756,  68.98760105, 68.98814508])
>>> tb37 = np.array([298.07385254, 297.15478516, 294.43276978, 281.67633057, 273.
↪7923584])
>>> tb11 = np.array([271.38806152, 271.38806152, 271.33453369, 271.98553467, 271.
↪93609619])
>>> viirs = RadTbConverter('Suomi-NPP', 'viirs', 'M12')
>>> rad37 = viirs.tb2radiance(tb37, normalized=False)
>>> rad11 = viirs.tb2radiance(tb11, normalized=False)
>>> sflux = 2.242817881698326
>>> nomin = rad37['radiance'] - rad11['radiance']
>>> print(np.isnan(nomin))
[False False False False False]
>>> print([np.round(val, 8) for val in nomin])
[0.05083677, 0.0480562, 0.04041571, 0.01279277, 0.00204485]
>>> denom = np.cos(np.deg2rad(sunz))/np.pi * sflux - rad11['radiance']
>>> print(np.isnan(denom))
[False False False False False]
>>> print([np.round(val, 8) for val in denom])
[0.23646312, 0.23645681, 0.23650559, 0.23582014, 0.23586609]
>>> res = nomin/denom
>>> print(np.isnan(res))
[False False False False False]
>>> print([np.round(val, 8) for val in res])
[0.21498817, 0.20323458, 0.17088693, 0.05424801, 0.00866952]
```

8.4 Derive the emissive part of the 3.7 micron band

Now that we have the reflective part of the $3.x$ signal, it is easy to derive the emissive part, under the same assumptions of completely opaque (zero transmissivity) objects.

$$L_{3.7,thermal} = (1 - \rho_{3.7}) \int_0^\infty \Phi_{3.7}(\lambda) B_\lambda(T_{11}) \mathrm{d}\lambda$$

Using the example of the VIIRS M12 band from above this gives the following spectral radiance:

```
>>> from pyspectral.near_infrared_reflectance import Calculator
>>> import numpy as np
```

(continues on next page)

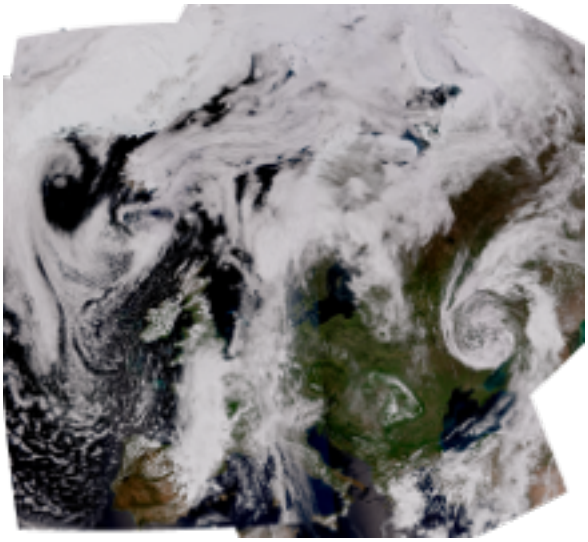
(continued from previous page)

```
>>> refl_m12 = Calculator('Suomi-NPP', 'viirs', 'M12')
>>> sunz = np.array([68.98597217,  68.9865146 ,  68.98705756,  68.98760105, 68.98814508])
>>> tb37 = np.array([298.07385254, 297.15478516, 294.43276978, 281.67633057, 273.
↪7923584])
>>> tb11 = np.array([271.38806152, 271.38806152, 271.33453369, 271.98553467, 271.
↪93609619])
>>> m12r = refl_m12.reflectance_from_tbs(sunz, tb37, tb11)
>>> tb = refl_m12.emissive_part_3x()
>>> ['{tb:6.3f}'.format(tb=np.round(t, 4)) for t in tb]
['266.996', '267.262', '267.991', '271.033', '271.927']
>>> rad = refl_m12.emissive_part_3x(tb=False)
>>> ['{rad:6.1f}'.format(rad=np.round(r, 1)) for r in rad.compute()]
['80285.2', '81458.0', '84749.7', '99761.4', '104582.0']
```

CHAPTER

NINE

ATMOSPHERIC CORRECTION IN THE VISIBLE SPECTRUM



In particular at the shorter wavelengths around $400-600nm$, which include the region used e.g. for ocean color products or true color imagery, in particular the Rayleigh scattering due to atmospheric molecules or atoms, and Mie scattering and absorption of aerosols, becomes significant. As this atmospheric scattering and absorption is obscuring the retrieval of surface parameters and since it is strongly dependent on observation geometry, it is custom to try to correct or subtract this unwanted signal from the data before performing the geophysical retrieval or generating useful and nice looking imagery.

In order to correct for this atmospheric effect we have simulated the solar reflectance under various sun-satellite viewing conditions for a set of different standard atmospheres, assuming a black surface, using a radiative transfer model. For a given atmosphere the reflectance is dependent on wavelength, solar-zenith angle, satellite zenith angle, and the relative sun-satellite azimuth difference angle:

$$\sigma = \sigma(\theta_0, \theta, \phi, \lambda)$$

The relative sun-satellite azimuth difference angle is defined as illustrated in the figure below:

The method is described in detail in a scientific paper.

To apply the atmospheric correction for a given satellite sensor band, the procedure involves the following three steps:

- Find the effective wavelength of the band
- Derive the atmospheric absorption and scattering contribution
- Subtract that from the observations

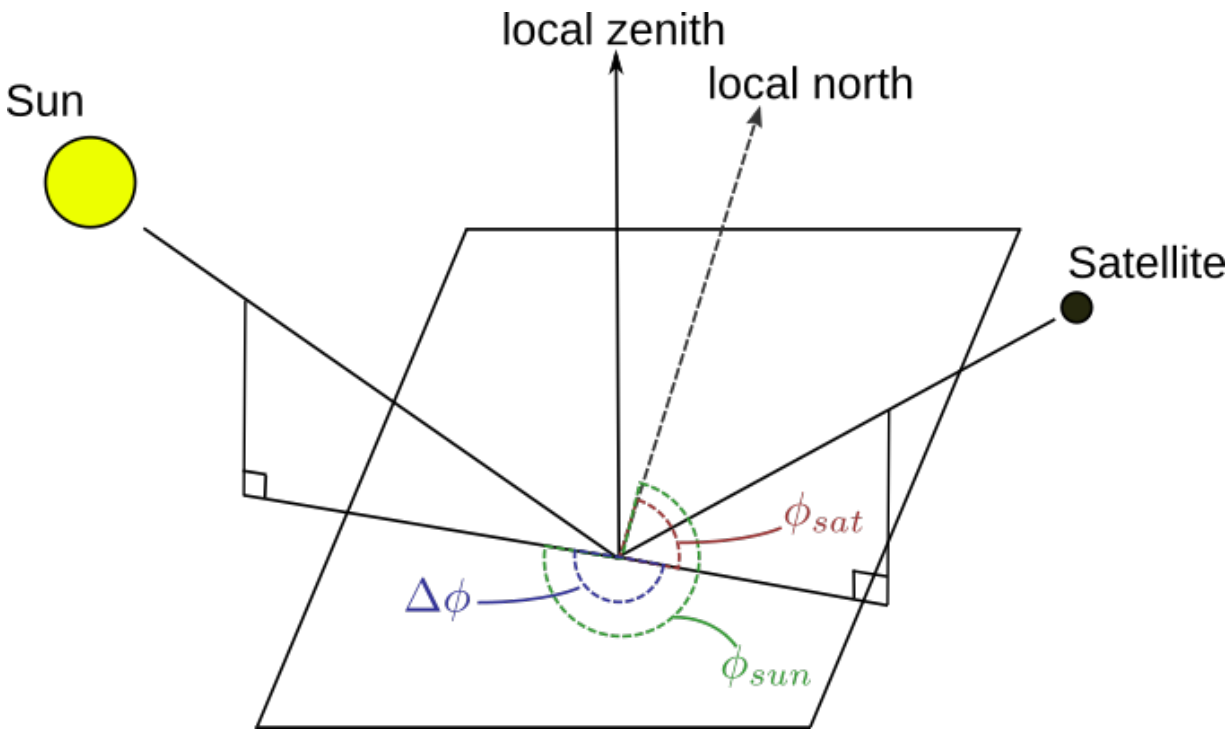
As the Rayleigh scattering, which is the dominating part we are correcting for under normal situations (when there is no excessive pollution or aerosols in the line of sight) is proportional to $\frac{1}{\lambda^4}$ the effective wavelength is derived by convolution of the spectral response with $\frac{1}{\lambda^4}$.

To get the atmospheric contribution for an arbitrary band, first the solar zenith, satellite zenith and the sun-satellite azimuth difference angles should be loaded.

For the VIIRS M2 band the interface looks like this:

```
>>> from pyspectral.rayleigh import Rayleigh
>>> viirs = Rayleigh('Suomi-NPP', 'viirs')
```

(continues on next page)



(continued from previous page)

```
>>> import numpy as np
>>> sunz = np.array([[32., 40.], [31., 41.]])
>>> satz = np.array([[45., 20.], [46., 21.]])
>>> ssadiff = np.array([[110, 170], [120, 180]])
>>> refl_cor_m2 = viirs.get_reflectance(sunz, satz, ssadiff, 'M2')
>>> print(refl_cor_m2)
[[ 10.45746088   9.69434733]
 [ 10.35336108   9.74561515]]
```

This Rayleigh (including Mie scattering and aborption by aerosols) contribution should then of course be subtracted from the data. Optionally the red band can be provided as the fifth argument, which will provide a more gentle scaling in cases of high reflectances (above 20%):

```
>>> redband =  np.array([[23., 19.], [24., 18.]])
>>> refl_cor_m2 = viirs.get_reflectance(sunz, satz, ssadiff, 'M2', redband)
>>> print(refl_cor_m2)
[[ 10.06530609   9.69434733]
 [  9.83569303   9.74561515]]
```

In case you want to bypass the reading of the sensor response functions or you have a sensor for which there are no RSR data available in PySpectral it is still possible to derive an atmospheric correction for that band. All that is needed is the effective wavelength of the band, given in micrometers (μm). This wavelength is normally calculated by PySpectral from the RSR data when passing the name of the band as above.

```
>>> from pyspectral.rayleigh import Rayleigh
>>> viirs = Rayleigh('UFO', 'ufo')
>>> import numpy as np
>>> sunz = np.array([[32., 40.], [31., 41.]])
```

(continues on next page)

(continued from previous page)

```
>>> satz = np.array([[45., 20.], [46., 21.]])
>>> ssadiff = np.array([[110, 170], [120, 180]])
>>> refl_cor_m2 = viirs.get_reflectance(sunz, satz, ssadiff, 0.45, redband)
[[ 9.55453743  9.20336915]
 [ 9.33705413  9.25222498]]
```

You may choose any name for the platform and sensor as you like, as long as it does not match a platform/sensor pair for which RSR data exist in PySpectral.

At the moment we have done simulations for a set of standard atmospheres in two different configurations, one only considering Rayleigh scattering, and one also accounting for Aerosols. On default we use the simulations with the marine-clean aerosol distribution and the US-standard atmosphere, but it is possible to specify if you want another setup, e.g.:

```
>>> from pyspectral.rayleigh import Rayleigh
>>> viirs = Rayleigh('Suomi-NPP', 'viirs', atmosphere='midlatitude summer', rural_
↪aerosol=True)
>>> refl_cor_m2 = viirs.get_reflectance(sunz, satz, ssadiff, 'M2', redband)
[[ 10.01281363   9.65488615]
 [  9.78070046   9.70335278]]
```

At high solar zenith angles the assumptions used in the simulations begin to break down, which can lead to unrealistic correction values. In particular, for true color imagery this often results in the red channel being too bright compared to the green and / or blue channels. We have implemented an optional scaling method to minimise this overcorrection at high solar zeniths, *reduce_rayleigh*:

```
>>> from pyspectral.rayleigh import Rayleigh
>>> import numpy as np
>>> viirs = Rayleigh('Suomi-NPP', 'viirs', atmosphere='midlatitude summer', rural_
↪aerosol=True)
>>> sunz = np.array([[32., 40.], [80., 88.]])
>>> satz = np.array([[45., 20.], [46., 21.]])
>>> ssadiff = np.array([[110, 170], [120, 180]])
>>> refl_cor_m2 = viirs.get_reflectance(sunz, satz, ssadiff, 0.45, redband)
[[ 10.40291763 9.654881]
 [ 30.9275331  39.41288558]]
>>> reduced_refl_cor_m2 = viirs.reduce_rayleigh_highzenith(sunz, refl_cor_m2, 70., 90.,␣
↪1.)
[[ 10.40291763 9.654881],
 [ 15.46376655 3.94128856]]
```

These reduced atmospheric correction (primarily due to increased Rayleigh scattering in the clear atmosphere) values can then be used to correct the original satellite reflectance data to produce more visually pleasing imagery, also at low sun elevations. Due to the nature of this reduction method they should not be used for any scientific analysis.

CHAPTER

TEN

FORMATTING THE ORIGINAL SPECTRAL RESPONSES

Pyspectral defines a unified hdf5 format to hold the various sensor specific relative spectral response functions.

10.1 Convert from original RSR data to pyspectral hdf5 format

The python modules in the directory `rsr_convert_scripts` contain code to convert from the original agency specific relative spectral responses to the internal unified pyspectral format in HDF5.

This conversion should normally never be done by the user. It will only be relevant if the original responses are updated. It that case we will need to redo the conversion and include the updated hdf5 file in the package data on zenodo.org.

Running the conversion scripts requires the pyspectral.yaml file to point to the directory of the original spectral response data. For AVHRR/3 onboard Metop-C this may look like this:

```
Metop-C-avhrr/3:
  path: /home/a000680/data/SpectralResponses/avhrr
  ch1: Metop_C_A309C001.txt
  ch2: Metop_C_A309C002.txt
  ch3a: Metop_C_A309C03A.txt
  ch3b: Metop_C_A309C03B.txt
  ch4: Metop_C_A309C004.txt
  ch5: Metop_C_A309C005.txt
```

Here <path> points to the place with the response functions for each channel. For Metop-C, the case was a bit special, as we got one file with all bands and using wavenumber and not wavelength. Therefore we first converted the file using script "split_metop_avhrr_rsrfile.py".

For all the other sensors supported in Pyspectral we run the conversion script directly on the files downloaded from internet (or acquired via mail-contact).

10.2 Conversion scripts

```
%> aatsr_reader.py
```

Converting the original ENVISAT AATSR responses. The data are stored i MS Excel format and the file name look like this: `consolidatedsrfs.xls`

```
%> python abi_rsr.py
```

Converting from original GOES-16&17 ABI responses. The file names look like this: `GOES-R_ABI_PFM_SRF_CWG_ch1.txt`

```
%> python ahi_rsr.py
```

Converting the original Himawari-8&9 AHI spectral responses. The data are stored i MS Excel format and the file names look like this: `AHI-9_SpectralResponsivity_Data.xlsx`

```
%> python avhrr_rsr.py
```

Converting from original Metop/NOAA AVHRR responses. The file names look like this: `NOAA_10_A101C004.txt`

As mentioned above for Metop-C we chopped up the single original file into band specific files to be consistent with the other AVHRRs. The original file we got from EUMETSAT: `AVHRR_A309_METOPC_SRF_PRELIMINARY.TXT`

```
%> python convert_avhrr_old2star.py
```

Convert the NOAA 15 Spectral responses to new NOAA STAR format.

```
%> python mersi2_rsr.py
```

Converts the FY-3D MERSI-2 spectral responses. Original files acquired via personal contact has names like this: `FY3D_MERSI_SRF_CH01_Pub.txt`

```
%> python modis_rsr.py
```

Converting the Terra/Aqua MODIS spectral responses to hdf5. Original Aqua MODIS files have names like this: `01.amb.1pct.det` Terra files have names like this: `rsr.1.oobd.det`

```
%> python msi_reader.py
```

The original Sentinel-2 A&B MSI spectral responses. Filenames look like this `S2-SRF_COPE-GSEG-EOPG-TN-15-0007_3.0.xlsx`

```
%> python olci_rsr.py
```

Converting the Sentinel 3A OLCI RSR data to hdf5. The original OLCI responses comes in a single netCDF4 file: `OLCISRFNetCDF.nc4`

```
%> python oli_reader.py
```

Conversion of the original Landsat-8 OLI data. File names look like this: `Ball_BA_RSR.v1.1-1.xlsx`

```
%> python seviri_rsr.py
```

Converting the Meteosat (second generation) SEVIRI responses to hdf5. Original filename: `MSG_SEVIRI_Spectral_Response_Characterisation.XLS`

```
%> python slstr_rsr.py
```

Converting the Sentinel-3 SLSTR spectral responses to hdf5. Original responses from ESA comes as a set of netCDF files. One file per band. Band 1: `SLSTR_FM02_S1_20150122.nc`

```
%> python viirs_rsr.py
```

Converting the NOAA-20 and Suomi-NPP VIIRS original responses to hdf5. File names follow 9 different naming conventions depending on the band, here as given in the pyspectral.yaml file:

```
section1:
  filename: J1_VIIRS_Detector_RSR_V2/J1_VIIRS_RSR_{bandname}_Detector_Fused_V2.txt
  bands: [M1, M2, M3, M4, M5, M6, M7]

section2:
  filename: J1_VIIRS_Detector_RSR_V2/J1_VIIRS_RSR_{bandname}_Detector_Fused_V2.txt
  bands: [I1, I2]

section3:
  filename: J1_VIIRS_V1_RSR_used_in_V2/J1_VIIRS_RSR_M8_Det_V1.txt
  bands: [M8]

section4:
  filename: J1_VIIRS_Detector_RSR_V2.1/J1_VIIRS_RSR_M9_Det_V2.1.txt
  bands: [M9]

section5:
  filename: J1_VIIRS_V1_RSR_used_in_V2/J1_VIIRS_RSR_{bandname}_Det_V1.txt
  bands: [M10, M11, M12, M14, M15]

section6:
  filename: J1_VIIRS_Detector_RSR_V2/J1_VIIRS_RSR_M13_Det_V2.txt
  bands: [M13]

section7:
  filename: J1_VIIRS_V1_RSR_used_in_V2/J1_VIIRS_RSR_M16A_Det_V1.txt
  bands: [M16]

section8:
  filename: J1_VIIRS_V1_RSR_used_in_V2/J1_VIIRS_RSR_{bandname}_Det_V1.txt
  bands: [I3, I4, I5]

section9:
  filename: J1_VIIRS_Detector_RSR_V2/J1_VIIRS_RSR_DNBLGS_Detector_Fused_V2S.txt
  bands: [DNB]
```

```
%> python msu_gsa_reader.py
```

Converts RSRs for the MSU-GS/A sensors aboard Arctica-M N1 satellite. RSRs were retrieved from Roshydromet. Filenames look like:

`rtcoef_electro-l_2_msugs_srf_ch01.txt`

Converts RSRs for the MSU-GS sensor aboard Electro-L N2. RSRs were retrieved from the NWP-SAF. Filenames look like: `rtcoef_electro-l_2_msugs_srf_ch01.txt`

```
%> python virr_rsr.py
```

Converting the FY-3B or FY-3C VIRR spectral responses to HDF5. Original files for FY-3B come as `.prn` text files for each channel (ex. `ch1.prn`). For FY-3C they come as `.txt` text files for channels 1, 2, 6, 7, 8, 9, and 10 only with names like `FY3C_VIRR_CH01.txt`.

PyTroll developers

CHAPTER

ELEVEN

THE PYSPECTRAL API

11.1 Blackbody radiation

Planck radiation equation.

pyspectral.blackbody.**blackbody**(*wavel*, *temp*)

Derive the Planck radiation as a function of wavelength.

SI units. blackbody(wavelength, temperature) wavel = Wavelength or a sequence of wavelengths (m) temp = Temperature (scalar) or a sequence of temperatures (K)

Output: The spectral radiance per meter (not micron!)
Unit = W/m^2 sr^-1 m^-1

pyspectral.blackbody.**blackbody_rad2temp**(*wavelength*, *radiance*)

Derive brightness temperatures from radiance using the inverse Planck function.

Parameters

- **wavelength** – The wavelength to use, in SI units (metre).
- **radiance** – The radiance to derive temperatures from, in SI units (W/m^2 sr^-1). Scalar or arrays are accepted.

Returns
The derived temperature in Kelvin.

pyspectral.blackbody.**blackbody_wn**(*wavenumber*, *temp*)

Derive the Planck radiation as a function of wavenumber.

SI units. blackbody_wn(wavnum, temperature) wavenumber = A wavenumber (scalar) or a sequence of wave numbers (m-1) temp = A temperatfure (scalar) or a sequence of temperatures (K)

Output: The spectral radiance in Watts per square meter per steradian
per m-1: Unit = W/m^2 sr^-1 (m^-1)^-1 = W/m sr^-1

Converting from SI units to mW/m^2 sr^-1 (cm^-1)^-1: 1.0 W/m^2 sr^-1 (m^-1)^-1 = 1.0e5 mW/m^2 sr^-1 (cm^-1)^-1

pyspectral.blackbody.**blackbody_wn_rad2temp**(*wavenumber*, *radiance*)

Derive brightness temperatures from radiance using the inverse Planck function.

Parameters

- **wavenumber** – The wavenumber to use, in SI units (1/metre).
- **radiance** – The radiance to derive temperatures from, in SI units (W/m^2 sr^-1). Scalar or arrays are accepted.

Returns
The derived temperature in Kelvin.

pyspectral.blackbody.**planck**(*wave*, *temperature*, *wavelength=True*)

Derive the Planck radiation as a function of wavelength or wavenumber.

SI units. _planck(wave, temperature, wavelength=True) wave = Wavelength/wavenumber or a sequence of wavelengths/wavenumbers (m or m^-1) temp = Temperature (scalar) or a sequence of temperatures (K)

Output: Wavelength space: The spectral radiance per meter (not micron!)
Unit = W/m^2 sr^-1 m^-1

Wavenumber space: The spectral radiance in Watts per square meter per steradian per m-1: Unit = W/m^2 sr^-1 (m^-1)^-1 = W/m sr^-1

Converting from SI units to mW/m^2 sr^-1 (cm^-1)^-1: 1.0 W/m^2 sr^-1 (m^-1)^-1 = 1.0e5 mW/m^2 sr^-1 (cm^-1)^-1

11.2 Spectral responses

Reading the spectral responses in the internal pyspectral hdf5 format.

class pyspectral.rsr_reader.**RSRDataBaseClass**

Bases: object

Data container for the Relative Spectral Responses for all (supported) satellite sensors.

property rsr_data_version_uptodate

Check whether RSR data are up to date.

class pyspectral.rsr_reader.**RSRDict**(*instrument=None*)

Bases: dict

Helper dict-like class to handle multiple names for band keys.

class pyspectral.rsr_reader.**RelativeSpectralResponse**(*platform_name=None*, *instrument=None*, *filename=None*)

Bases: *RSRDataBaseClass*

Container for the relative spectral response functions for various satellite imagers.

convert()

Convert spectral response functions from wavelength to wavenumber.

get_bandname_from_wavelength(*wavelength*, *epsilon=0.1*, *multiple_bands=False*)

Get the band name from the wavelength.

get_number_of_detectors4bandname(*h5f*, *bandname*)

For a band name get the number of detectors, if any.

get_relative_spectral_responses(*h5f*)

Read the rsr data and add to the object.

integral(*bandname*)

Calculate the integral of the spectral response function for each detector.

load()

Read the internally formatet hdf5 relative spectral response data.

set_band_central_wavelength_per_detector(*h5f*, *bandname*, *detector_name*)

Set the central wavelength for the band and detector.

set_band_names(*h5f*)

Set the band names.

set_band_responses_per_detector(*h5f*, *bandname*, *detector_name*)

Set the RSR responses for the band and detector.

set_band_wavelengths_per_detector(*h5f*, *bandname*, *detector_name*)

Set the RSR wavelengths for the band and detector.

set_description(*h5f*)

Set the description.

set_instrument(*h5f*)

Set the instrument name.

set_platform_name(*h5f*)

Set the platform name.

pyspectral.rsr_reader.**check_and_download**(*dest_dir=None*, *dry_run=False*)

Do a check for the version and attempt downloading only if needed.

Base class for reading raw instrument spectral responses.

class pyspectral.raw_reader.**InstrumentRSR**(*bandname*, *platform_name*, *bandnames=None*)

Bases: object

Base class for the raw (agency dependent) instrument response functions.

11.3 Solar irradiance

Read solar irradiances and calculate solar flux.

Module to read solar irradiance spectra and calculate the solar flux over various instrument bands given their relative spectral response functions

class pyspectral.solar.**SolarIrradianceSpectrum**(*filename=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/p*
***options*)

Bases: object

Total Top of Atmosphere (TOA) Solar Irradiance Spectrum.

Wavelength is in units of microns (10^-6 m). The spectral Irradiance in the file TOTAL_IRRADIANCE_SPECTRUM_2000ASTM is in units of W/m^2/micron

convert2wavenumber()

Convert from wavelengths to wavenumber.

Units:
Wavelength: micro meters (1e-6 m) Wavenumber: cm-1

inband_solarflux(*rsr*, *scale=1.0*, ***options*)

Get the in band solar flux.

Derive the inband solar flux for a given instrument relative spectral response valid for an earth-sun distance of one AU.

inband_solarirradiance(*rsr*, *scale=1.0*, ***options*)

Get the in band solar irradiance.

Derive the inband solar irradiance for a given instrument relative spectral response valid for an earth-sun distance of one AU.

(Same as the in band solar flux).

interpolate(***options*)

Interpolate Irradiance to a specified evenly spaced resolution/grid.

This is necessary to make integration and folding (with a channel relative spectral response) straightforward.

dlambda = wavelength interval in microns start = Start of the wavelength interval (left/lower) end = End of the wavelength interval (right/upper end) options: dlambda: Delta wavelength used when interpolating/resampling ival_wavelength: Tuple. The start and end interval in wavelength space, defining where to integrate/convolute the spectral response curve on the spectral irradiance data.

plot(*plotname=None*, ***options*)

Plot the data.

solar_constant()

Calculate the solar constant.

11.4 Near-Infrared reflectance

Derive NIR reflectances of a a band in the 3-4 micron window region.

Derive the Near-Infrared reflectance of a given band in the solar and thermal range (usually the 3.7-3.9 micron band) using a thermal atmospheric window channel (usually around 11-12 microns).

class pyspectral.near_infrared_reflectance.**Calculator**(*platform_name*, *instrument*, *band*, *detector='det-1'*, *wavespace='wavelength'*, *solar_flux=None*, *sunz_threshold=85.0*, *masking_limit=85.0*)

Bases: RadTbConverter

A thermal near-infrared (~3.7 micron) band reflectance calculator.

Given the relative spectral response of the NIR band, the solar zenith angle, and the brightness temperatures of the NIR and the Thermal bands, derive the solar reflectance for the NIR band removing the thermal (terrestrial) part. The in-band solar flux over the NIR band is optional. If not provided, it will be calculated here!

The relfectance calculated is without units and should be between 0 and 1.

derive_rad39_corr(*bt11*, *bt13*, *method='rosenfeld'*)

Derive the CO2 correction to be applied to the 3.9 channel.

Derive the 3.9 radiance correction factor to account for the attenuation of the emitted 3.9 radiance by CO2 absorption. Requires the 11 micron window band and the 13.4 CO2 absorption band, as e.g. available on SEVIRI. Currently only supports the Rosenfeld method

emissive_part_3x(*tb=True*)

Get the emissive part of the 3.x band.

reflectance_from_tbs(*sun_zenith*, *tb_near_ir*, *tb_thermal*, ***kwargs*)

Derive reflectances from Tb's in the 3.x band.

The relfectance calculated is without units and should be between 0 and 1.

Inputs:

sun_zenith: Sun zenith angle for every pixel - in degrees

tb_near_ir: The 3.7 (or 3.9 or equivalent) IR Tb's at every pixel
(Kelvin)

tb_thermal: The 10.8 (or 11 or 12 or equivalent) IR Tb's at every
pixel (Kelvin)

tb_ir_co2: The 13.4 micron channel (or similar - co2 absorption band)
brightness temperatures at every pixel. If None, no CO2 absorption correction will be applied.

pyspectral.near_infrared_reflectance.**get_as_array**(*variable*)

Return variable as a Dask or Numpy array.

Variable may be a scalar, a list or a Numpy/Dask array.

11.5 Rayleigh scattering

Atmospheric correction of shortwave imager bands in the wavelength range 400 to 800 nm.

class pyspectral.rayleigh.**Rayleigh**(*platform_name*, *sensor*, ***kwargs*)

Bases: *RayleighConfigBaseClass*

Container for the atmospheric correction of satellite imager bands.

This class removes background contributions of Rayleigh scattering of molecules and Mie scattering and absorption by aerosols.

get_reflectance(*sun_zenith*, *sat_zenith*, *azidiff*, *band_name_or_wavelength*, *redband=None*)

Get the reflectance from the three sun-sat angles.

static reduce_rayleigh_highzenith(*zenith*, *rayref*, *thresh_zen*, *maxzen*, *strength*)

Reduce the Rayleigh correction amount at high zenith angles.

This linearly scales the Rayleigh reflectance, *rayref*, for solar or satellite zenith angles, *zenith*, above a threshold angle, *thresh_zen*. Between *thresh_zen* and *maxzen* the Rayleigh reflectance will be linearly scaled, from one at *thresh_zen* to zero at *maxzen*.

class pyspectral.rayleigh.**RayleighConfigBaseClass**(*aerosol_type*, *atm_type='us-standard'*)

Bases: `object`

A base class for the Atmospheric correction, handling the configuration and LUT download.

property lutfiles_version_uptodate

Tell whether LUT file is up to date or not.

pyspectral.rayleigh.**check_and_download**(*dry_run=False*, *aerosol_types=None*)

Download atm correction LUT tables if they are not up-to-date already.

Do a check for the version of the atmospheric correction LUTs and attempt downloading only if needed.

pyspectral.rayleigh.**get_reflectance_lut_from_file**(*lut_filename*)

Get reflectance LUT.

Read the Look-Up Tables from file with reflectances as a function of wavelength, satellite zenith secant, azimuth difference angle, and sun zenith secant.

11.6 Utils

Utility functions.

pyspectral.utils.**AEROSOL_TYPES = ['antarctic_aerosol', 'continental_average_aerosol', 'continental_clean_aerosol', 'continental_polluted_aerosol', 'desert_aerosol', 'marine_clean_aerosol', 'marine_polluted_aerosol', 'marine_tropical_aerosol', 'rayleigh_only', 'rural_aerosol', 'urban_aerosol']**

Aerosol types available as downloadable LUTs for rayleigh correction

class pyspectral.utils.**NullHandler**(*level=0*)

Bases: Handler

Empty handler.

emit(*record*)

Record a message.

pyspectral.utils.**are_instruments_identical**(*name1*, *name2*)

Given two instrument names check if they are both describing the same instrument.

Takes care of the case of AVHRR where the internal pyspectral naming (following WMO Oscar) is is with a slash as in 'avhrr/1', but where a naming using a dash instead is equally accepted, as in 'avhrr-1'.

pyspectral.utils.**bytes2string**(*var*)

Decode a bytes variable and return a string.

pyspectral.utils.**check_and_adjust_instrument_name**(*platform_name*, *instrument*)

Check instrument name and try fix if inconsistent.

It checks against the possible listed instrument names for each platform. It also makes an adjustment replacing names like avhrr/1 with avhrr1, removing the '/'.

pyspectral.utils.**convert2hdf5**(*ClassIn*, *platform_name*, *bandnames*, *scale=1e-06*, *detectors=None*)

Retrieve original RSR data and convert to internal hdf5 format.

scale is the number which has to be multiplied to the wavelength data in order to get it in the SI unit meter

pyspectral.utils.**convert2str**(*value*)

Convert a value to string.

Parameters

value – Either a str, bytes or 1-element numpy array

pyspectral.utils.**convert2wavenumber**(*rsr*)

Convert Spectral Responses from wavelength to wavenumber space.

Take rsr data set with all channels and detectors for an instrument each with a set of wavelengths and normalised responses and convert to wavenumbers and responses

Rsr

Relative Spectral Response function (all bands)

Returns

retv: Relative Spectral Responses in wave number space :info: Dictionary with scale (to go convert to SI units) and unit

pyspectral.utils.**debug_on**()

Turn debugging logging on.

pyspectral.utils.**download_luts**(*aerosol_types=None*, *dry_run=False*, *aerosol_type=None*)

Download the luts from internet.

See *pyspectral.rayleigh.check_and_download()* for a "smart" version of this process that only downloads the necessary files.

Parameters

- **aerosol_types** (*Iterable*) – Aerosol types to download the LUTs for. Defaults to all aerosol types. See *AEROSOL_TYPES* for the full list.
- **dry_run** (*bool*) – If True, don't actually download files, only log what URLs would be downloaded. Defaults to False.
- **aerosol_type** (*str*) – Deprecated.

pyspectral.utils.**download_rsr**(*dest_dir=None*, *dry_run=False*)

Download the relative spectral response functions.

Download the pre-compiled HDF5 formatted relative spectral response functions from the internet as tarballs, extracts them, then deletes the tarball.

See *pyspectral.rsr_reader.check_and_download()* for a "smart" version of this process that only downloads the necessary files.

Parameters

- **dest_dir** (*str*) – Path to put the temporary tarball and extracted RSR files.
- **dry_run** (*bool*) – If True, don't actually download files, only log what URLs would be downloaded. Defaults to False.

pyspectral.utils.**get_bandname_from_wavelength**(*sensor*, *wavelength*, *rsr*, *epsilon=0.1*, *multiple_bands=False*)

Get the bandname from h5 rsr provided the approximate wavelength.

pyspectral.utils.**get_central_wave**(*wav*, *resp*, *weight=1.0*)

Calculate the central wavelength or the central wavenumber.

Calculate the central wavelength or the central wavenumber, depending on which parameters is input. On default the weighting funcion is f(lambda)=1.0, but it is possible to add a custom weight, e.g. f(lambda) = 1./lambda**4 for Rayleigh scattering calculations

pyspectral.utils.**get_logger**(*name*)

Return logger with null handle.

pyspectral.utils.**get_rayleigh_lut_dir**(*aerosol_type*)

Get the rayleigh LUT directory for the specified aerosol type.

pyspectral.utils.**get_wave_range**(*in_chan*, *threshold=0.15*)

Return central, min and max wavelength in an RSR greater than threshold.

An RSR function will generally start near zero, increase to a maximum and then drop back to near zero. This function takes advantage of this to find the first and last points where the RSR is greater than a threshold. These

points are then defined as the minimum and maximum wavelengths for a given channel, and can be used, for example, in Satpy reader YAML files.

pyspectral.utils.**logging_off**()

Turn logging off.

pyspectral.utils.**logging_on**(*level=30*)

Turn logging on.

pyspectral.utils.**np2str**(*value*)

Convert an *numpy.string_* to str.

Parameters
value (*ndarray*) – scalar or 1-element numpy array to convert

Raises
ValueError – if value is array larger than 1-element or it is not of type *numpy.string_* or it is not a numpy array

pyspectral.utils.**sort_data**(*x_vals*, *y_vals*)

Sort the data so that x is monotonically increasing and contains no duplicates.

pyspectral.utils.**use_map_blocks_on**(*argument_to_run_map_blocks_on*)

Use map blocks on a given argument.

This decorator assumes only one of the arguments of the decorated function is chunked.

CHAPTER

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

L

M

N

P

R

S

U