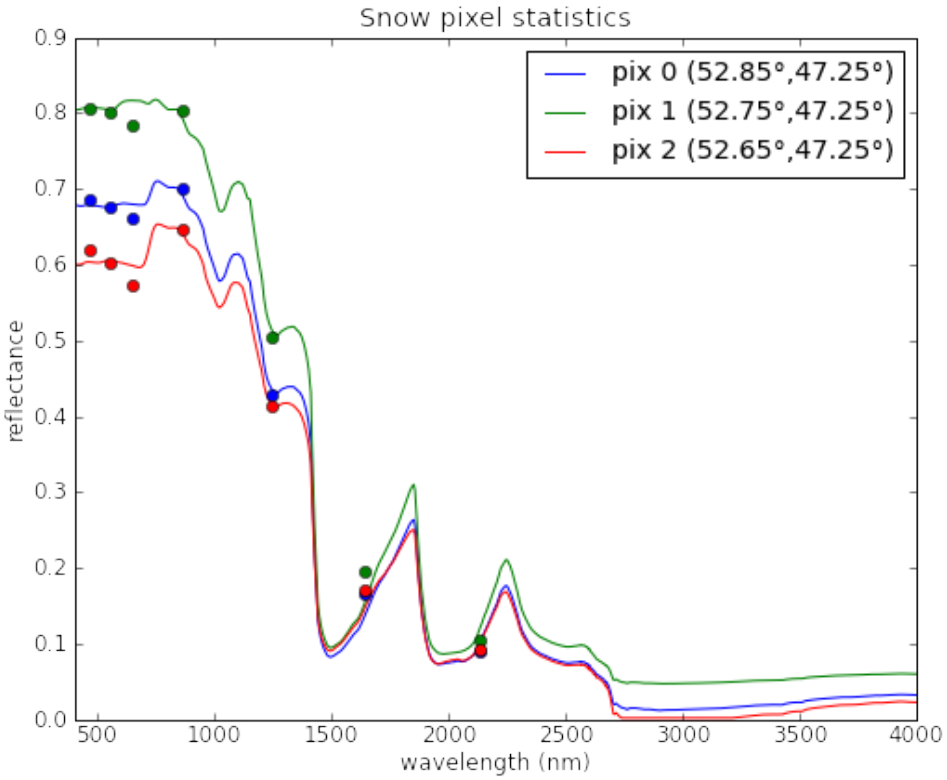
ADAM Documentation

Release 3.0

C. Bacour, I. Price, C. Boudesocque

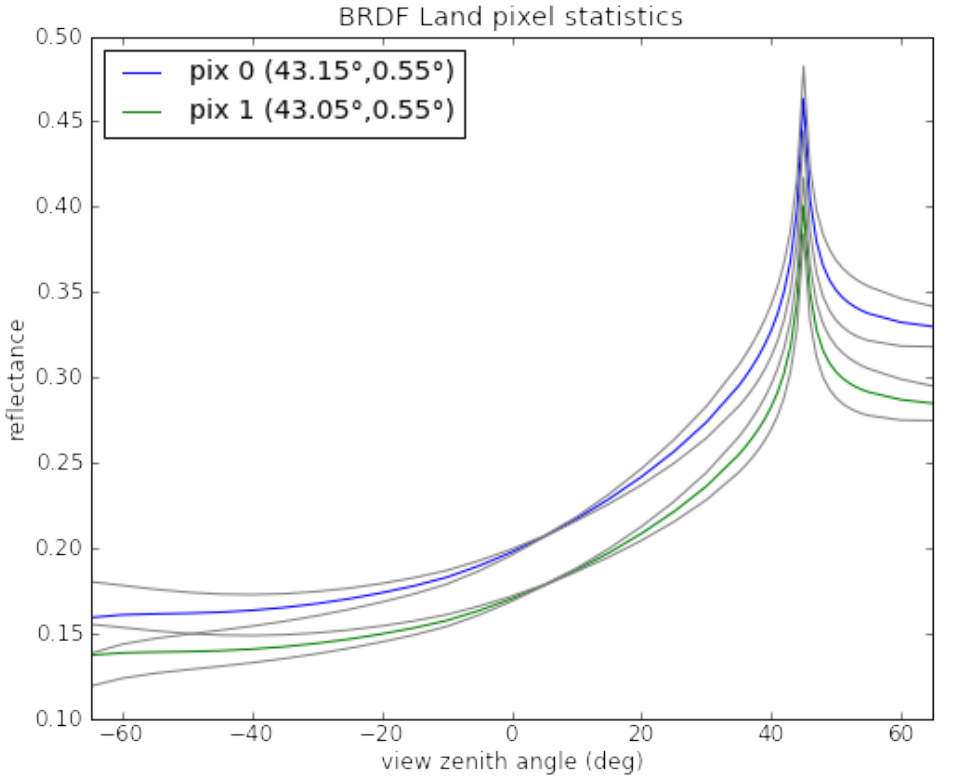
February 27, 2017



ADAM

Reflectance Database

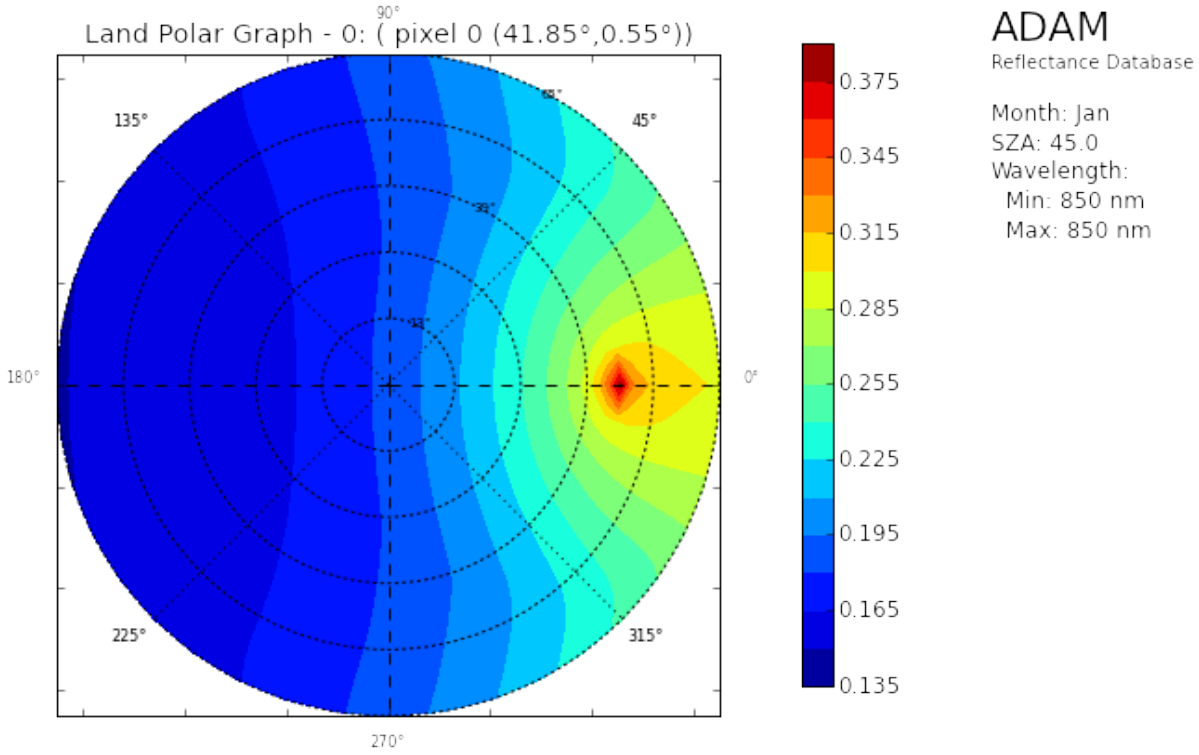
Month: Jan
Lats:
52.650:52.850
Lons:
47.250:47.250
SZA: 40.0
VZA: [30.]
phi: [0.]

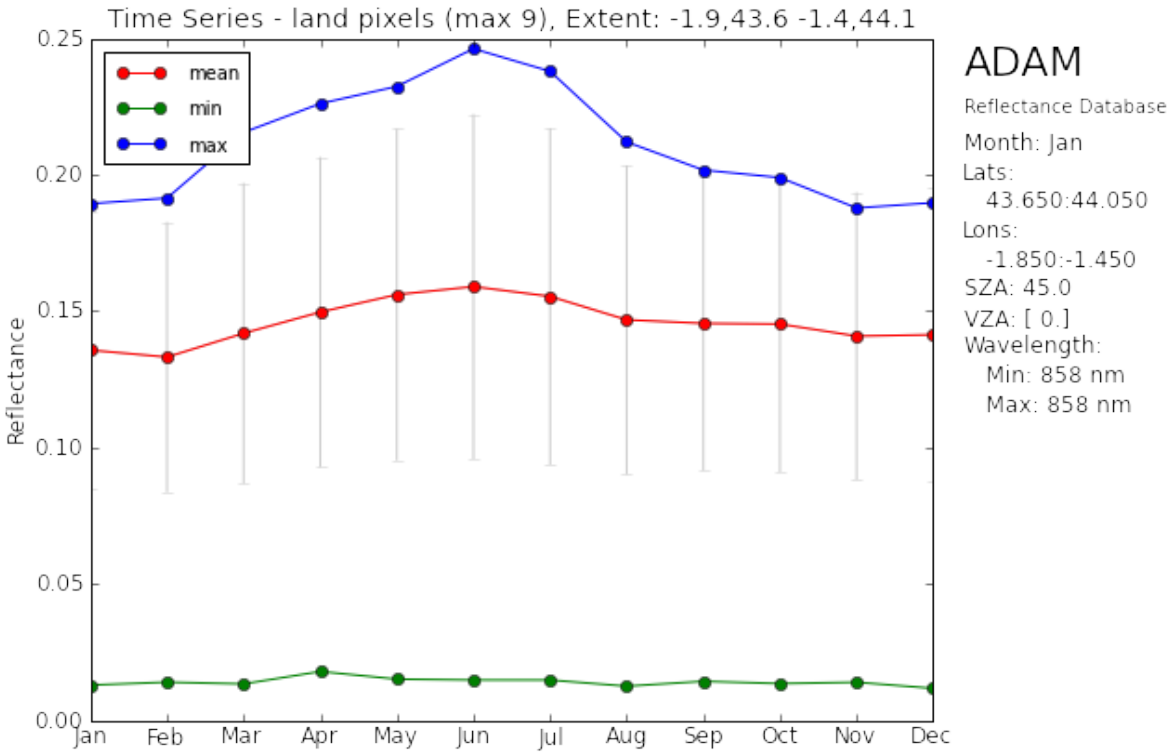


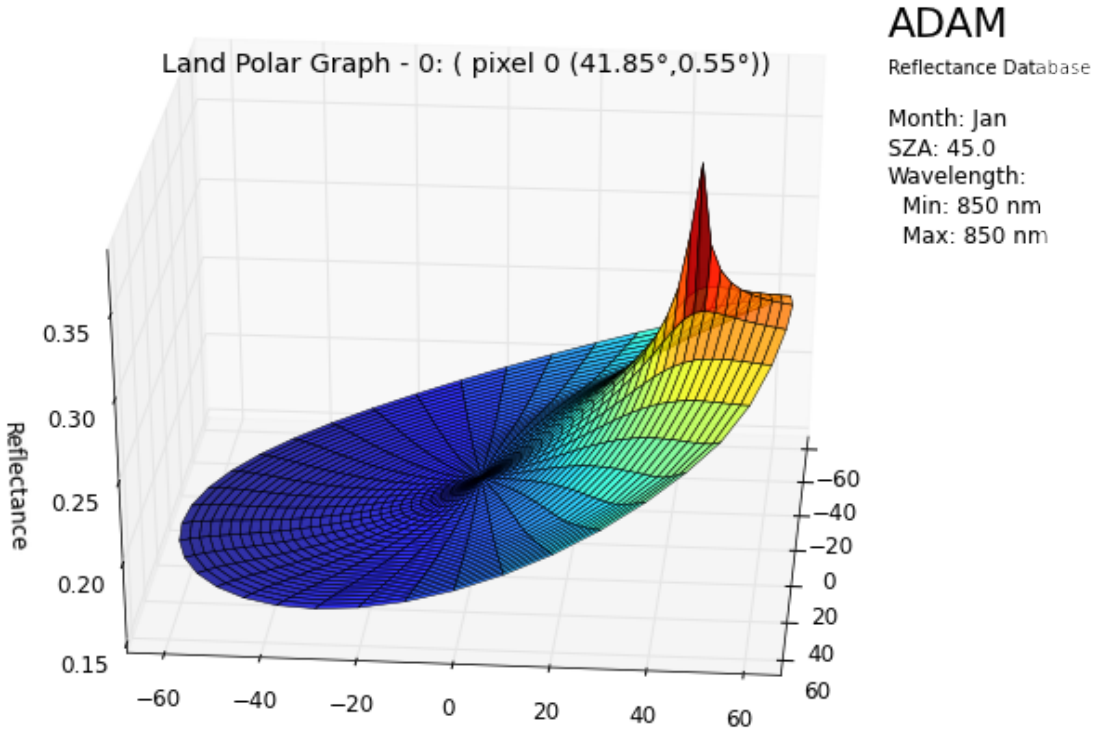
ADAM

Reflectance Database

Month: Jan
Lats:
43.05000:43.15000
Lons:
0.55000:0.55000
SZA: 45.0
Wavelength:
Min: 850 nm
Max: 860 nm


Land Polar Graph - 0: (pixel 0 (41.85°,0.55°))
90°
135°
45°
180°
0°
225°
315°
270°
0.375
0.345
0.315
0.285
0.255
0.225
0.195
0.165
0.135
ADAM
Reflectance Database
Month: Jan
SZA: 45.0
Wavelength:
Min: 850 nm
Max: 850 nm


Time Series - land pixels (max 9), Extent: -1.9,43.6 -1.4,44.1
mean
min
max
Reflectance
0.25
0.20
0.15
0.10
0.05
0.00
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
ADAM
Reflectance Database
Month: Jan
Lats:
43.650:44.050
Lons:
-1.850:-1.450
SZA: 45.0
VZA: [0.]
Wavelength:
Min: 858 nm
Max: 858 nm


Land Polar Graph - 0: (pixel 0 (41.85°,0.55°))
ADAM
Reflectance Database
Month: Jan
SZA: 45.0
Wavelength:
Min: 850 nm
Max: 850 nm
Reflectance
0.35
0.30
0.25
0.20
0.15
−60
−40
−20
0
20
40
60

CHAPTER

ONE

CONTENTS:

1.1 Introduction

The ADAM API was originally intended to be used by the ADAM Website as the 'server side code', responding to requests for graphs and calculations.

During the course of the ADAM project it became apparent that there was a need for expert users to have a greater degree of control over the inputs, outputs and parameters of the ADAM functions than would be practical via the website. For this reason the API has been made available for such users, allowing them to download the entire codebase behind the website in order to execute calculations locally on their own hardware according to their specific requirements.

Note: There exists the ability for users to use the ADAM website via the command line using HTTP clients such as wget or cURL. This may meet the requirements of certain users who need to frequently repeat tasks which would be possible but time consuming using the website graphical user interface. The help documentation on the ADAM website provides examples of how this can be achieved, it does not involve the use of the ADAM API at all and is therefore not discussed in this documentation.

1.2 Installing the API

To use the API the user must first:

- install the Python code from PyPi
- download and unzip the complete ADAM database

Note: The ADAM database is around 2.4 Gigabytes uncompressed, so you will need to have about twice that space available on your disk for the download and decompression.

Installing the ADAM Python code is performed in the same way as any other Python module. The easiest method is by using easy_install:

```
# easy_install -U ADAM-Tools
```

The PyPi page has further installation instructions for the Python code.

After the installation of the code the user should uncompress the entire ADAM database archive to a folder, keeping the directry structure intact.

When writing scripts that use the API you will initialise the ADAM API with a reference to the directory containing the unzipped database like this:

```
import adam
import adam_config

# define config:
```

```
my_config = adam_config.adam_config(path_data='/tmp/data', output_root_dir='/tmp/output/')
# path syntax for windows:
# my_config = adam_config.adam_config(path_data='c://adam-data/input/', output_root_dir='c://adam

# initialise the job
job = adam.adam_job( cfg=my_config )
```

In the example above the ADAM database was unzipped to the directory **/tmp/data/** and output data will be written to **/tmp/output/**

That's it ! If you can run the above example without an error message then the API is installed and working on your computer.

1.3 Getting Started

After installing the API, database and verifying that the installation is happy with the above code, we recommend that you try out some of the scripts found in the test/ directory, particularly *spectrum_graph.py* and *brdf_graph_transectplane.py* .

These scripts provide a good starting point for further customisation to provide the results you require from the API.

Looking at the examples you can see the general structure of an ADAM calculation:

- initialise config, suitable for the local machine
- initialise an instance of an 'ADAM Job' with that config
- verify & pass the calculation parameters (region / illumination / viewing angles) to the job
- load input data into the job
- calculate reflectance
- optionally: calculate BRDF
- optionally: output graphs
- optionally: output netCDF
- optionally: access the raw calculated reflectance/BRDF data and do something different

1.4 The ADAM Job Class

Central to the use of the ADAM API is the concept of the ADAM Job. A single calculation performed by the API is termed a

- `load_data()`: to load raw data into the job
- `validate_input()`: to load and validate the input parameters
- `process_reflectance()`: to process the reflectance for the 240-4000nm range
- `graph_main_ref_spectra()`: output graph(s) of the reflectance spectra
- ... and so on. For a full list of methods refer to the *ADAM Job*.

1.4.1 Initialisation

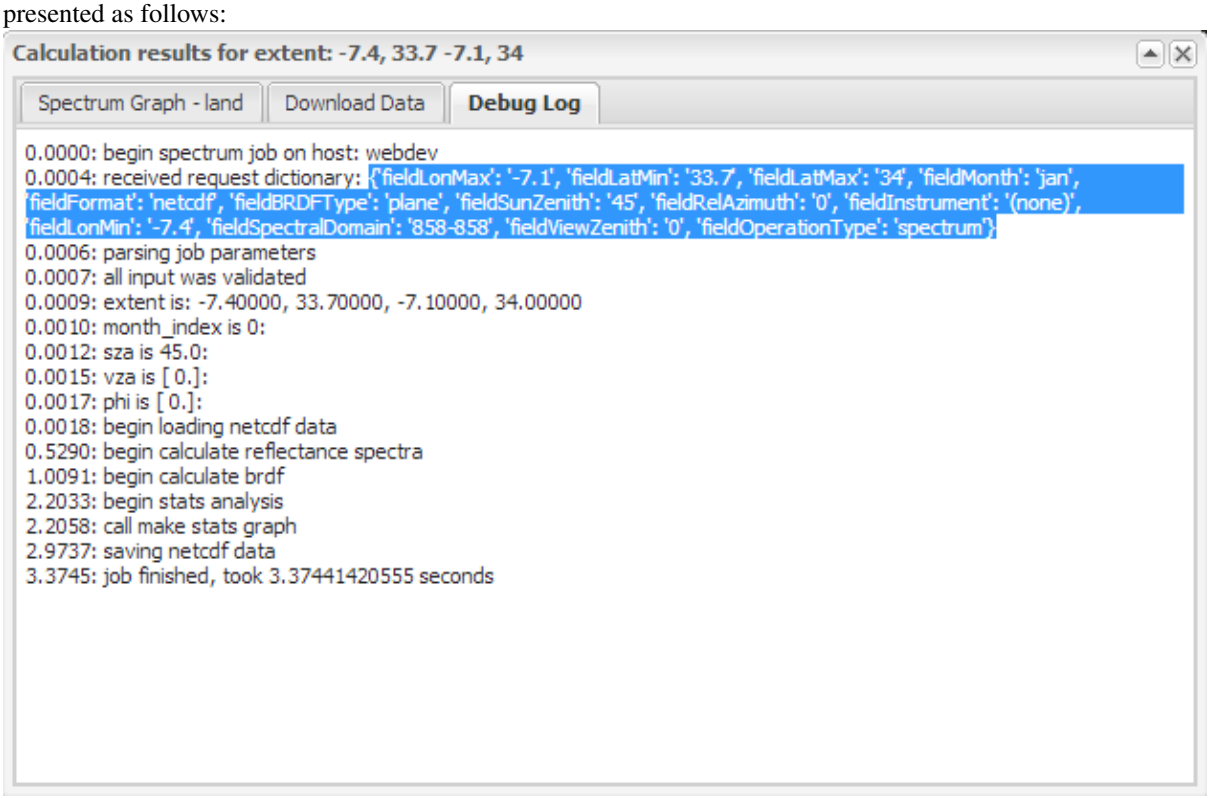
The ADAM Job class can be initialised in the normal pythonic way:

```
job = adam.adam_job( job_output_dir='/tmp/', sza=45, vza=np.arange(0,100) )
```

However the preferred method is to use the `adam.adam_job.validate_input()` method which performs a verification of the input fields and performs some automatic preparation depending on the type of job you would like to perform.

`adam.adam_job.validate_input()` takes a dictionary as an argument, containing the names and values of the various keywords for the new job. This dictionary is the same format as that used by the website, which allows you to use the ADAM website to generate a typical input dictionary for use and later customisation in the API.

For example, after performing a Spectrum Graph via the website, the 'Debug Log' in the results window is presented as follows:



Highlighted in the above example is the 'request dictionary'. This text can be directly copy and pasted as an argument for the `adam.adam_job.validate_input()` method as follows:

```
request_dict = {
    'fieldOperationType': 'spectrum',
    'fieldMonth'        : 'jan',
    'fieldSunZenith'    : 45,
    'fieldViewZenith'   : 0,
    'fieldRelAzimuth'   : 0,
    'fieldSpectralDomain': '858-858 nm',

    'fieldLonMin': '-6',
    'fieldLonMax': '-5.3',
    'fieldLatMax': '35.6',
    'fieldLatMin': '36.3'
}

if not job.validate_input(request_dict):
    raise Exception('request input invalid !')
```

The `adam.adam_job.validate_input()` method returns a True or False depending on the success of the validation.

1.4.2 Accessing Job Data

The ADAM Job object has a property called 'data' which itself is a dictionary. The dictionary contains all the various arrays of data that have been loaded and calculated for that job, for example, after running job.process_brdf() we can perform the following:

```
>>> for key in sorted(job.data.keys()):
...     print key
...
NDVI
SDR
SDR_err_land
chloro_conc
idx_land
idx_ocean
idx_snow
latitude
longitude
mask_land
mask_ocean
mask_snow
month_index
qf
ref_land
ref_land_covar
reflectance
reflectance_err_land
reflectance_full
reflectance_full_err_land
wind_speed
>>>
```

The contents of each item in the dictionary is access like this:

```
>>> job.data['SDR'].shape
(1, 2, 1, 47, 1)
>>> job.data['SDR'].mean()
0.26286457954569065
>>> job.data['SDR']
array([[[[[ 0.1592543 ],
          [ 0.1608945 ],
          [ 0.16142392],
          [ 0.16178204],
          [ 0.16240701],
          [ 0.16350621],
          [ 0.16517392],
          [ 0.16744705],
          [ 0.17033319],
.......
          [ 0.28698364],
          [ 0.28480038]]]]], dtype=float32)
>>>
```

Many of the arrays in the ADAM Job data dictionary are multidimensional. For instance in the above example the dimensions of the SDR variable are, after BRDF Transect Plane calculation: pixels_x, pixels_y, wavelength, vza, phi (corresponding to the output of job.data['SDR'].shape: (1, 2, 1, 47, 1))

For details about the dimensions of the various arrays you can refer to the documentation of the functions that generate them in the *ADAM Job*.

1.5 Default configuration and behaviour

1.5.1 Calculation configurations

- By default, the directional parameters `R` and `V` of the BRDF model are parameterized as a function of the normalized reflectance and the values of NDVI.

  There is a possibility for the user to change that configuration and allow the parameterization for `R` and `V` not accounting for the NDVI value. To do so, the user must change in the *adam_config.py* file the value of the "*BRDF_method*" of the *config_land* dictionary from "*ndvi_classes*" to "*global*".

- The ART model used to calculate the spectro-directional variation of snow reflectance do not account for the impact of pollutants by default. The user may change that configuration by changing the value of the "*do_estimate_pollutant*" keyword of the *config_land* dictionary from *False* to *True* (*adam_config.py* file).

Note: Note however that the ADAM API has not been validated using this configuration and that possible numerical issues may occur.

1.5.2 Display

- For "BRDF" calculations (either transect plane or polar or 3D representation), when several spectral bands are requested, only results for the first one are displayed (unless prescribed with the *waveband_index* keyword).
- For "spectrum" calculations, if several viewing geometries are requested, the SDR will be displayed only for the first one.

1.6 Examples

The following examples are lifted directly from the scripts found in the test/ directory in the distribution.

To make them work on your system you will need to ensure the paths for the input and output data are correct at the top of each script.

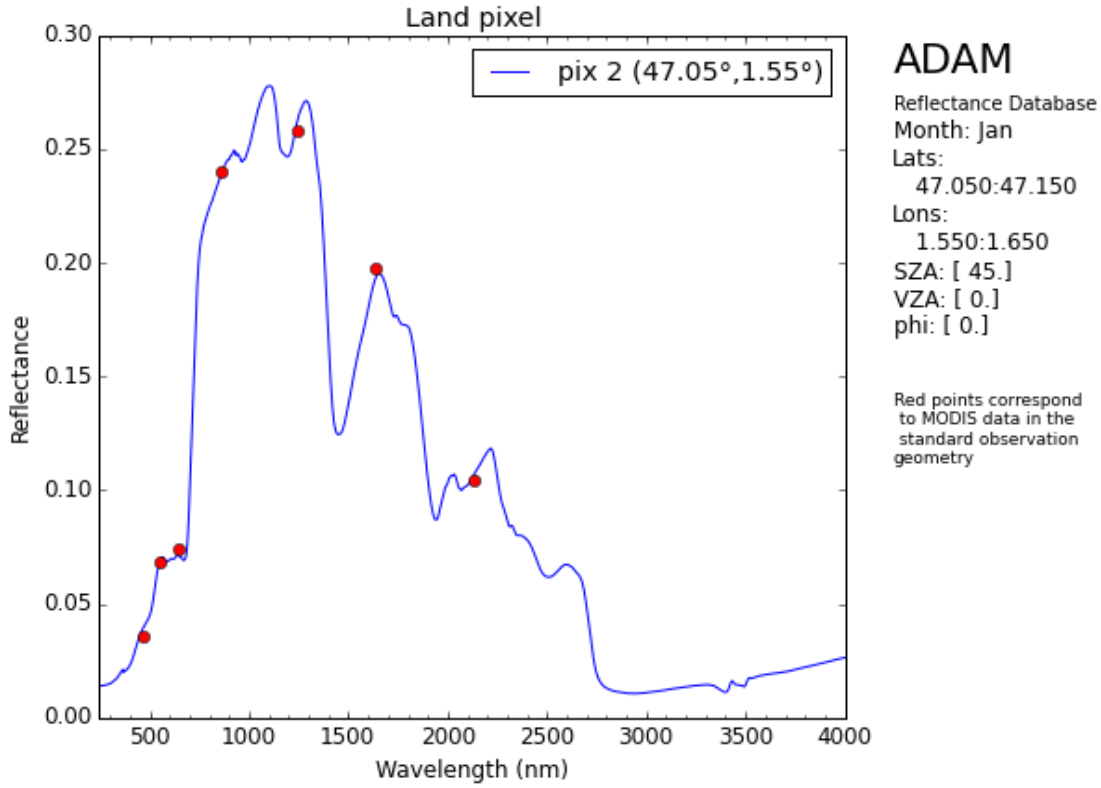
1.6.1 spectrum_graph.py

Script using the ADAM Toolkit demonstrating making a spectrum graph. The scripts calculate the SDR over the 240-4000 spectral domain for an ensemble of pixels (contiguous area) for one observation geometry.

This example demonstrates:

- initialisation of a new ADAM job
- entry / validation of calculation parameters
- calculation of the reflectance
- calculation of the BRDF
- generation of some graph outputs

Example graph output should look similar to this:



The contents of spectrum_graph.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : spectrum_graph.py
description     : Script using the ADAM Toolkit demonstrating making a spectrum graph
                  The scripts calculate the SDR over the 240-4000 spectral domain for an ensemble
                  for one observation geometry
version         : 3.0
usage           : python spectrum_graph.py
notes           :
"""


import sys
import numpy as np

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')


# -
# - Configuration
# -
# import the main adam library
import adam
import adam_config


# input directory containing
#  - the ADAM-V3 netCDF files
#  - the ascii/ directory containing additional text files required by the API
# my_path_data       = '/data_no_saved/adam/adam-data/'
```

```
# # output directory
# my_output_root_dir = '/NOVELTIS/boudesocque/dev/adam/output/'

my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'

# output directory
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'


my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_SPECTRUM'

# -
# - API parameters
# -

# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = {'fieldOperationType': 'spectrum',
                'fieldMonth'        : 'jan',
                # 'fieldSunZenith'    : np.array([40,45]),
                # 'fieldViewZenith'   : np.array([30,35]),
                # 'fieldRelAzimuth'   : np.array([0,85]),
                'fieldSunZenith'    : 45,
                'fieldViewZenith'   : 0,
                'fieldRelAzimuth'   : 0,
                'fieldComputeError': False,
#                'fieldSpectralDomains': '400-1200 nm; 1400-1900',
#                'fieldInstrument': 'modis',

# land
                 'fieldLatMin'   : 47.2,
                 'fieldLatMax'   : 47.,
                 'fieldLonMin'   : 1.5,
                 'fieldLonMax'   : 1.7


                # 'fieldLatMin'   : -6.,
                # 'fieldLatMax'   : -4,
                # 'fieldLonMin'   : 17.9,
                # 'fieldLonMax'   : 18.1

                # 'fieldLatMin'    : -2.8,
                #  'fieldLatMax'   : -2.5,
                #  'fieldLonMin'   : -65.5,
                #  'fieldLonMax'   : -65.4

                # soil
                # 'fieldLatMin'    : -25.1,
                # 'fieldLatMax'   : -25,
                # 'fieldLonMin'   : 136.1,
                # 'fieldLonMax'   : 136.2

                # 'fieldLatMin'    : 20.3,
                # 'fieldLatMax'   : 20.4,
                # 'fieldLonMin'   : 50.6,
                # 'fieldLonMax'   : 50.7
```

```
    # # ocean
    #               'fieldLatMin'   : -15.6,
    #               'fieldLonMin'   : -15.6,
    #               'fieldLatMax'   : -15.,
    #               'fieldLonMax'   : -15.1

                  # 'fieldLatMin'   : 43.6,
                  # 'fieldLonMin'   : -1.9,
                  # 'fieldLatMax'   : 44.1,
                  # 'fieldLonMax'   : -1.4

                  # 'ieldCorner1Lon': '-1.9',
                  # 'fieldLonMax': '-1.4',
                  # 'fieldSpectralDomain': '858-858',
                  # 'fieldLatMax': '44.1',
                  # 'fieldLatMin': '43.6',

                  # snow with low reflectance values
                  # 'fieldLonMin': '47.2',
                  # 'fieldLonMax': '47.3',
                  # 'fieldLatMax': '52.9',
                  # 'fieldLatMin': '52.6'

                  # land / ocean pixels
                  # 'fieldLatMin'   : 37.2,
                  # 'fieldLonMin'   : -3.7,
                  # 'fieldLatMax'   : 36.4,
                  # 'fieldLonMax'   : -2.9


                  # ocean only
                  # 'fieldLonMin': '-35',
                  # 'fieldLonMax': '-35.2',
                  # 'fieldLatMax': '28',
                  # 'fieldLatMin': '27.8'
                  # 'fieldLonMin': '-3.1',
                  # 'fieldLonMax': '-3.2',
                  # 'fieldLatMax': '44.2',
                  # 'fieldLatMin': '44.1'


                  # snow with low reflectance values
                  # 'fieldLonMin': '47.2',
                  # 'fieldLonMax': '47.3',
                  # 'fieldLatMax': '52.9',
                  # 'fieldLatMin': '52.6'


                  # snow with high reflectance values
                  # 'fieldLonMin': '-47.3',
                  # 'fieldLonMax': '-47.2',
                  # 'fieldLatMax': '66.9',
                  # 'fieldLatMin': '66.6'

                  # Pyrenees
                  # 'fieldLonMin': '-0.4',
                  # 'fieldLonMax': '-0.3',
                  # 'fieldLatMax': '43.0',
                  # 'fieldLatMin': '42.7'

                  }
```

```
# -
# - Processing
# -
# initialise the job, providing us with a blank object ('job') to work with
print 'initialise job..'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir ) # here we provide the job_ou

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')

# now we are ready to load the data from our source database into our job object
print 'loading data..'
job.load_data()


# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))


# call the convenience functions to populate the various variables

# - spectral extrapolation of the normalized MODIS reflectance
print 'calculate reflectance..'
# populates job.data['reflectance']
job.process_reflectance()

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'calculate brdf..'
job.process_brdf()


# - now we can make our graphs
print 'make graphs..'

# graphs are created differently depending on how many pixels are to be rendered
# if there are only a few pixels we generate graphs depicting individual pixels
if job.is_pixel_request():
    print 'generating pixel graphs'
    # unlike ocean and snow, land spectrum graphs are treated one graph per pixel
    for i, pixel_index in enumerate(job.data['idx_land']):
        idx = [pixel_index]
        # the land error matrix is only populated for land pixels, so it uses a different counter
        job.graph_main_ref_spectra(case='pixels', indices=idx, title='Land pixel')

    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_snow'],  title='Snow pixel st
    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_ocean'], title='Ocean pixel s
else:
    # or if there are many pixels we take the mean/max/min and graph only those stats
    print 'generating stats graphs'
    job.calculate_stats(job.data['SDR'])
    job.graph_main_ref_spectra(case='stats')


# - Output netCDF file
print 'saving netcdf..'
job.save_netcdf()
```

```
print 'done.'
```

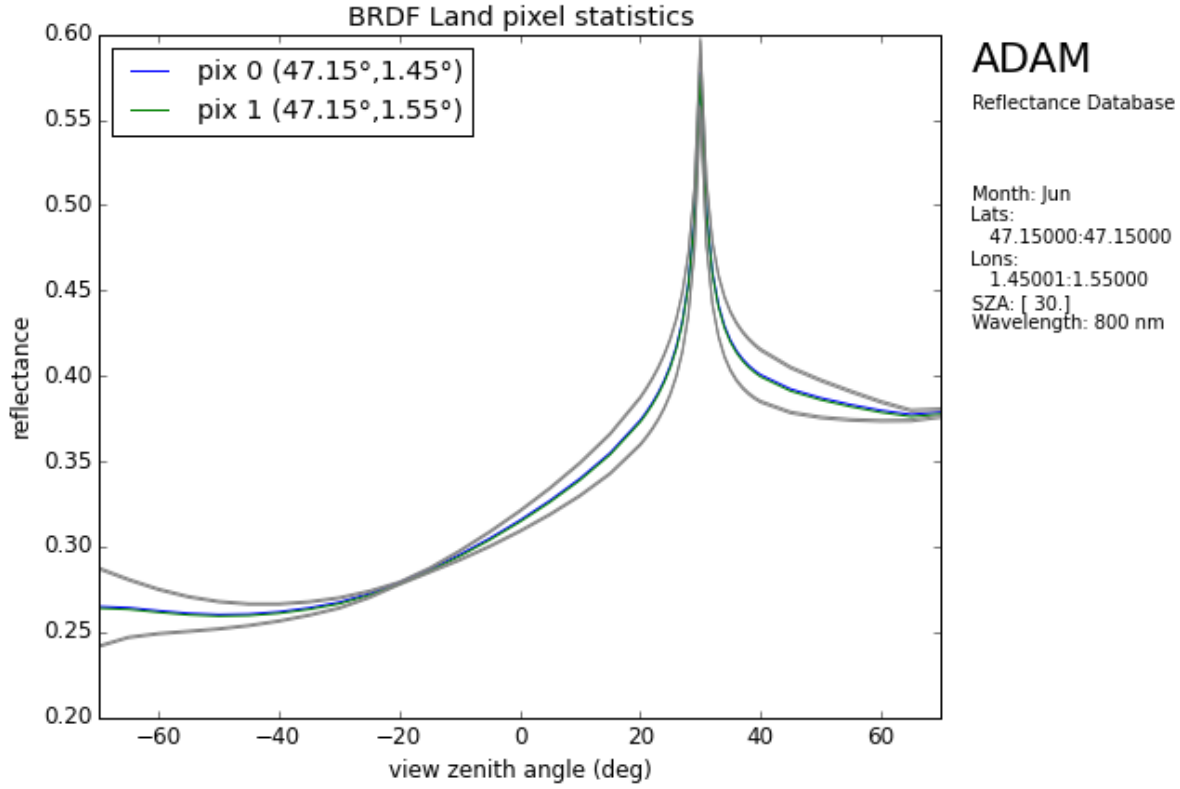
1.6.2 brdf_graph_transectplane.py

Script using the ADAM Toolkit demonstrating making BRDF graphs in a particular plane. The calculation is performed over 2 land pixels with activating the calculation of the model uncertainty in order to display the directional modelling error. The output SDR are calculated at 800 nm.

This example is very similar, except that we use the define_vza_hotspot() method to create a 'hot spot' of increased vza values around the sza and call the BRDF graphing functions.

Several other parameters were specified in the request_dict input, including that the spectral domain for the job is between 800 and 800 nm.

Example graph output should look similar to this:



The contents of brdf_graph_transectplane.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : brdf_graph_transectplane.py
description     : Script using the ADAM Toolkit demonstrating making BRDF graphs in a particular
                  The calculation is performed over 2 land pixels with activating the calculation
                  the model uncertainty in order to display the directional modelling error. The
                  are calculated at 800 nm.
version         : 3.0
usage           : python brdf_graph_transectplane.py
notes           :
"""


import numpy as np
import sys
```

```
# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')



# -
# - Configuration
# -
# import the adam libraries
import adam
import adam_config

# input directory containing
#  +  the ADAM-V3 netCDF files
#  + the ascii/ directory containing additional text files required by the API

my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'
my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_BRDF'

# -
# - API parameters
# -

# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = { 'fieldOperationType': 'brdf',
                 'fieldBRDFType': 'plane',             # this define the display (2D BRDF)
                 'fieldMonth': 'jun',
                 'fieldSunZenith':  30,
                 'fieldViewZenith': 0,                 # the VZA angles are overwritten later (jo
                 'fieldRelAzimuth': 0,                 # with 0 (90), the calculations are perfor
                 'fieldSpectralDomain': '800-800',
                  #'fieldInstrument': 'modis',
                 'fieldComputeError': True,

# land
                 'fieldLatMin'   : 47.2,
                 'fieldLatMax'   : 47.1,
                 'fieldLonMin'   : 1.5,
                 'fieldLonMax'   : 1.505

               }

# -
# - Processing
# -

# initialise the job, providing us with a blank object ('job') to work with
print 'init job/config'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir )

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')
```

```
# now we are ready to load the data from our source database into our job object
print 'load data...'
job.load_data()

# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))

# - spectral extrapolation of the normalized MODIS reflectance
print 'perform reflectance calc...'
job.process_reflectance()

# - increase vza sampling depending on sza (increase resolution near sza)
print 'define_vza_hotspot...'
job.define_vza_hotspot()

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'Perform BRDF calculation...'
job.process_brdf()


# - now we can make our graphs
print 'make graphs..'

# graphs are created differently depending on how many pixels are to be rendered
# if there are only a few pixels we generate graphs depicting individual pixels
if job.is_pixel_request():
    print 'generating pixel graphs'
    job.graph_transect_plane(case='pixels', indices=job.data['idx_land'],  title='BRDF Land pixel
    job.graph_transect_plane(case='pixels', indices=job.data['idx_ocean'], title='BRDF Ocean pixe
    job.graph_transect_plane(case='pixels', indices=job.data['idx_snow'],  title='BRDF Snow pixel


else:
    # or if there are many pixels we take the mean/max/min and graph only those stats
    print 'generating stats graphs'
    job.calculate_stats(job.data['SDR'])
    job.graph_transect_plane(case = 'stats')


# - Save the output netCDF file
job.save_netcdf()
print 'job output was written to: %s' % job.job_output_dir
```

1.6.3 brdf_graph_polar_3D.py

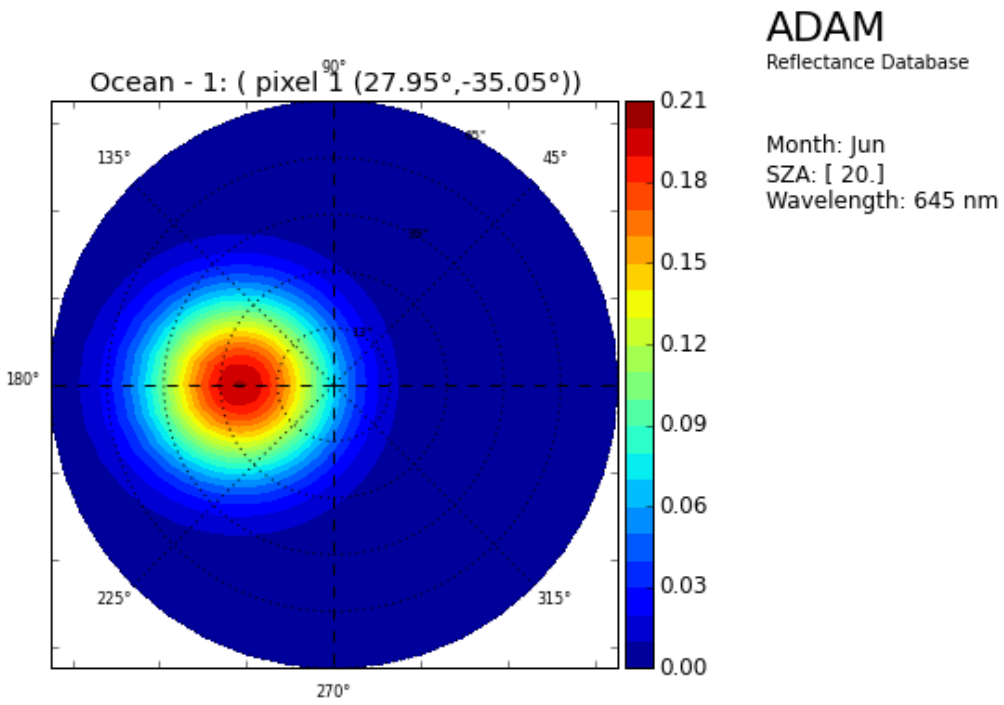
Script using the ADAM Toolkit demonstrating making polar and 3D BRDF graphs. The SDR are calculated in MODIS bands. By default, the SDR is displayed in the 1st waveband (*waveband_index* = 0):

- for the 3D plots, we use that default behavior
- for the polar plots, we force the display in the 3rd MODIS band

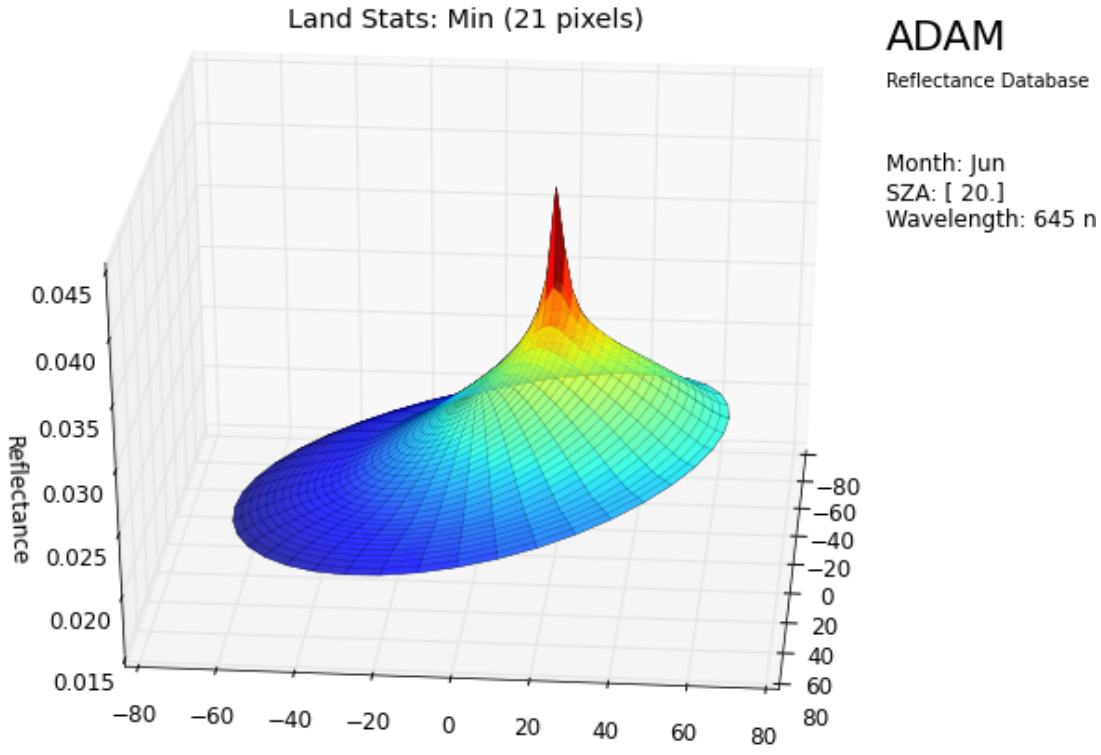
Once it is generated, the users is able to rotate the graph using the mouse in an interactive fashion.

It demonstrates calculating the BRDF for a multi-dimensional array, where both vza and phi are multi-valued.

Example graph output should look similar to this for polar plot (example for 1 Ocean pixel):



Example graph output should look similar to this for 3 plot (minimum reflectance values of several land pixels)



The contents of brdf_graph_polar_3D.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : brdf_graph_polar_3D.py
description     : Script using the ADAM Toolkit demonstrating making polar and 3D BRDF graphs
```

```
                    The SDR are calculated in MODIS bands
                    By default, the SDR is displayed in the 1st waveband (waveband_index = 0):
                     - for the 3D plots, we use that default behavior
                     - for the polar plots, we force the display in the 3rd MODIS band
version          : 3.0
usage            : python brdf_graph_polar_3D.py
notes            :
"""



import numpy as np
import sys


# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')

# -
# - Configuration
# -
# import the main adam library
import adam
import adam_config


# input directory containing
#  - the ADAM-V3 netCDF files
#  - the ascii/ directory containing additional text files required by the API
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'

# output directory
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'


my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_API'

# -
# - API parameters
# -

# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = { 'fieldOperationType': 'brdf',
                 'fieldBRDFType': 'polar',
                 'fieldSunZenith': 20,
                 'fieldViewZenith': 0,
                 'fieldRelAzimuth': 0,
                 'fieldMonth': 'jun',
#                 'fieldSpectralDomains': '800-850 nm',
                 'fieldInstrument': 'modis',

# land
                 # 'fieldLatMin'   : 43.9,
                 # 'fieldLonMin'   : 0.5,
                 # 'fieldLatMax'   : 41.8,
                 # 'fieldLonMax'   : 0.6
```

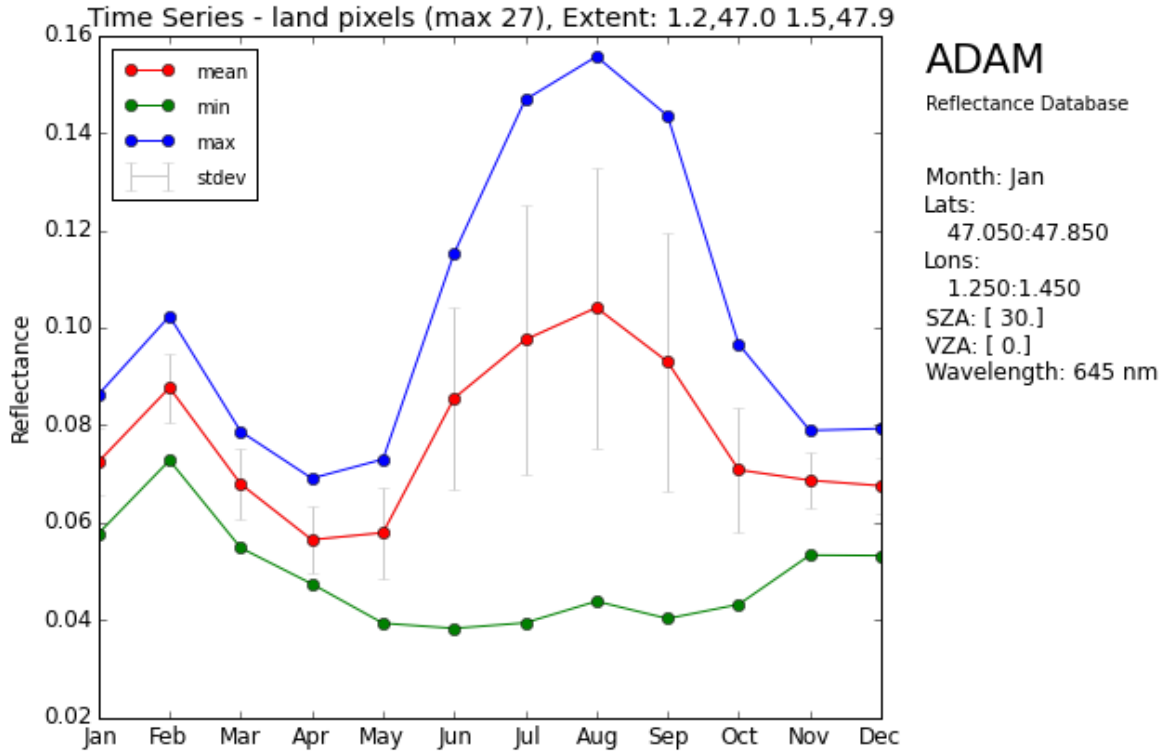
```
#  ocean only
                'fieldLonMin': '-35',
                'fieldLonMax': '-35.2',
                'fieldLatMax': '28',
                'fieldLatMin': '27.8'

#                 snow with high reflectance values
#                 'fieldLonMin': '-47.3',
                # 'fieldLonMax': '-47.2',
                # 'fieldLatMax': '66.9',
                # 'fieldLatMin': '66.6'
                }


# -
# - Processing
# -
# initialise the job, providing us with a blank object ('job') to work with
print 'init job/config'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir )

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')


# now we are ready to load the data from our source database into our job object
print 'load data...'
job.load_data()

# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))

# - spectral extrapolation of the normalized MODIS reflectance
print 'perform reflectance calc...'
job.process_reflectance()

# - now redefined vza and phi suitable for a polar/3D plots
job.phi = np.linspace(0,360, 41)
job.vza = np.linspace(0,65, 31)

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'Perform BRDF calculation...'
job.process_brdf()

if job.is_pixel_request():
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_land'], title='Land', three_d=True
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_snow'], title='Snow', three_d=True
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_ocean'], title='Ocean', three_d=T

    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_land'], title='Land', three_d=Fals
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_snow'], title='Snow', three_d=Fals
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_ocean'], title='Ocean', three_d=Fa
else:
    job.calculate_stats(job.data['SDR'])
    job.graph_3Dpolar_plot(case='stats',  title='Stats', three_d=True, waveband_index = 2) # displ
    job.graph_3Dpolar_plot(case='stats',  title='Stats', three_d=False, waveband_index = 2) # disp

# - Save the output netCDF file
job.save_netcdf()
```

```
print 'job output was written to: %s' % job.job_output_dir

# - Interactive graph
if job.is_pixel_request():
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_land'], title='Land', three_d=True
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_snow'], title='Snow', three_d=True
    job.graph_3Dpolar_plot(case='pixels', indices=job.data['idx_ocean'], title='Ocean', three_d=T
```

1.6.4 time_series_graph.py

Script using the ADAM Toolkit demonstrating making SDR time series graphs. The scripts calculate the SDR in the 7 MODIS bands for an ensemble of pixels (contiguous area) for one observation geometry over 1 year.

Example graph output should look similar to this:



The contents of time_series_graph.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : time_series_graph.py
description     : Script using the ADAM Toolkit demonstrating making SDR time series graphs
                  The scripts calculate the SDR in MODIS bands for an ensemble of pixels (contigu
                  for one observation geometry over 1 year
version         : 3.0
usage           : python time_series_graph.py
notes           :
"""


import sys

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
```

```
if not '../' in sys.path:
    sys.path.insert(0, '../')

# import the main adam library
import adam
import adam_config as adam_config

# -
# - Configuration
# -

# input directory containing
#  - the ADAM-V3 netCDF files
#  - the ascii/ directory containing additional text files required by the API
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'

# output directory
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'


my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_API'

# -
# - API parameters
# -
# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = { 'fieldOperationType': 'time-series',
                 'fieldSunZenith':  30,
                 'fieldViewZenith': 0,
                 'fieldRelAzimuth': 0,
                 'fieldInstrument': 'modis',
                  #'fieldSpectralDomains': '800-865',

                 # ocean + land
                 # 'fieldLatMin'   : 44.1,
                 # 'fieldLonMin'   : -1.9,
                 # 'fieldLatMax'   : 43.6,
                 # 'fieldLonMax'   : -1.4

                 # land
                 'fieldLatMin'   : 47.9,
                 'fieldLatMax'   : 47.,
                 'fieldLonMin'   : 1.5,
                 'fieldLonMax'   : 1.2
                 }



# -
# - Processing
# -
# initialise the job, providing us with a blank object ('job') to work with
print 'initialise job..'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir  )

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
```

```
    raise Exception('request input invalid !')


# - calls the spectral extrapolation and then the calculation in the requested geometry
print 'make time series'
job.graph_time_series(waveband_index = 2)    # display of the data for the waveband index 2 (start

# data is available in: job.data['stats']['land']['mean']
print 'job output was written to: %s' % job.job_output_dir
```

1.6.5 brdf_commandline.py

Script using the ADAM Toolkit demonstrating how taking user input from the command line.

This maybe useful if the code is to be called in a loop or similar from a non-python source.

Example usage from the command line:

> python -i brdf_commandline.py –sza 30.0 –vza 40.0 –phi 0.0 –latMin 36.0 –lonMin 5.2 –latMax 36.2 –lonMax 5.5 –month jan –beyond_za_limit nan –sza_max_limit 30 –netcdf-output ./output.nc –operation-type spectrum –wave-min 400 –wave-max 2500

The contents of brdf_commandline.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : brdf_commandline.py
description     : Script using the ADAM Toolkit demonstating taking user input from the commandli
version         : 3.0
notes           :

EXAMPLE USAGE:
python brdf_commandline.py --sza 30.0 --vza 40.0 --phi 0.0 --latMin 36.0 --lonMin 5.2 --latMax 36

"""


import numpy as np
import sys
#import argparse
from optparse import OptionParser

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')


# import the adam libraries
import adam
import process_brdf
import adam_config as adam_config


# -
# - Configuration
# -
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'
my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# -
# - Parameters
```

```
# -
# command line input

parser = OptionParser(description='ADAM commandline options')
parser.add_option('--sza', help='Sun Zenith Angle (default 45)', default=45)
parser.add_option('--vza', help='View Zenith Angle (default 0)', default=0)
parser.add_option('--phi', help='Phi (default 0)', default=0)
parser.add_option('--sza_max_limit', help='Maximum value of sza authorized', default=70)
parser.add_option('--vza_max_limit', help='Maximum value of vza authorized', default=70)
parser.add_option('--beyond_za_limit', help='Behaviour of the BRDF models when sza of vza are beyo
parser.add_option('--latMin', help='Latitude Min')
parser.add_option('--lonMin', help='Longitude Min')
parser.add_option('--latMax', help='Latitude Max')
parser.add_option('--lonMax', help='Longitude Max')
parser.add_option('--wave-min', type=int, help='Wavelength min', default = 240)
parser.add_option('--wave-max', type=int, help='Wavelength max', default = 4000)
parser.add_option('--month', help='Month', default='jan', choices=['jan', 'feb', 'mar', 'apr', 'ma
parser.add_option('--operation-type', help='Type of operation to perform', choices=["brdf","specti
parser.add_option('--netcdf-output', help='Name of netcdf output filename')


options, args = parser.parse_args()
def usage_and_exit():
    print '** Error parsing input **'
    parser.print_help()
    print '''Example usage:\npython -i brdf_commandline.py --sza 30.0 --vza 40.0 --phi 0.0 --latMi
\n\n'''
    sys.exit()

if None in [options.latMin, options.lonMin, options.latMax, options.latMax, options.operation_type
    usage_and_exit()




# -
# - API parameters
# -
# Determine how the spectral domain must be interpreted, depending on the operation type
#  - brdf => averaging over the spectral domain defined between wave_min and wave_max
#  - spectrum => spectral sampling over contiguous bands between wave_min and wave_max
spectral_domain = '%s-%s' % (options.wave_min, options.wave_max)
# if spectrum selected, one wants to get the SDR in all wavebands between wave_min and wave_max
if options.operation_type == 'spectrum':
    spectral_domain = np.arange(options.wave_max-options.wave_min+1)+options.wave_min

# define the type of adam calculation we wish to perform..
request_dict = { 'fieldOperationType'  : options.operation_type,
                 'fieldSpectralDomain' : spectral_domain,
                 'fieldSunZenith'      : options.sza,
                 'fieldViewZenith'     : options.vza,
                 'fieldRelAzimuth'     : options.phi,

                 'fieldSunZenithMaxLimit'     : options.sza_max_limit,
                 'fieldViewZenithMaxLimit'    : options.vza_max_limit,
                 'fieldBeyondZALimit'         : options.beyond_za_limit,

                 'fieldLatMin'     : options.latMin,
                 'fieldLonMin'     : options.lonMin,
                 'fieldLatMax'     : options.latMax,
                 'fieldLonMax'     : options.lonMax,
```

```
                }

# initialise the job, providing us with a blank object ('job') to work with
print 'init job/config'
job = adam.adam_job( cfg=my_config )

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')


# now we are ready to load the data from our source database into our job object
print 'load data...'
job.load_data()

# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))


# - spectral extrapolation of the normalized MODIS reflectance
print 'perform reflectance calc...'
job.process_reflectance()

# - increase vza sampling depending on sza (increase resolution near sza) (if necesary)
if "brdf" in job.job_type:
    print 'define_vza_hotspot...'
    job.define_vza_hotspot()

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'Perform BRDF calculation...'
job.process_brdf()

# - save output file
print 'save netcdf'
job.save_netcdf(output_filename=options.netcdf_output)


if options.netcdf_output is not None:
    print 'finished, netcdf written to: %s' % options.netcdf_output
else:
    print 'finished, netcdf written to: %s' % job.job_output_dir
```

1.6.6 calc_brdf_PARASOL.py

Script using the ADAM Toolkit to calculate the spectral BRDF for an ensemble of pixels in PARASOL observations geometries. The ouputs are saved here in a cPickle file.

The contents of calc_brdf_PARASOL.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : calc_brdf_PARASOL.py
description     : Script using the ADAM Toolkit to calculate the spectral BRDF
                  for an ensemble of pixels in PARASOL observations geometries
                  The ouputs are saved here in a cPickle file

usage           : python calc_brdf_PARASOL.py

"""
```

```
import numpy as np
import sys
import time
import multiprocessing
import traceback
from optparse import OptionParser
import os,sys
import cPickle
import matplotlib.pyplot as plt

plt.ion()


# import the adam libraries

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')

import adam
import adam_config as adam_config

# -
# - Configuration
# -
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3'
my_output_root_dir = '/home/scratch01/cbacour/ADAM_V3_vs_PARASOL/'
my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di


# -
# - Parameters
# -
months = [['M01','M02','M03','M04','M05','M06','M07','M08','M09','M10','M11','M12'],
          ['jan','feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']]


delta_deg = 0.001

lmbds = [490,565,670,765,865,1020]
parasol_inst = ['parasol', [490,565,670,765,865,1020]]
lmbds_MODIS = [469, 555, 645, 858, 1240, 1640, 2130]

# - definition of the site dictionnary containing informations on the pixels to
#   process (it can be read from a file)

# Here we define data for 3 pixels
pixels = {}
for i in range(3): pixels[i] = {}
# 0
pixels[0]['month'] = 'M02'
pixels[0]['sza'] = np.array([20,21,23])
pixels[0]['vza'] = np.array([10,10,10])
pixels[0]['phi'] = np.array([40,50,60])
pixels[0]['lat'] = 45.
pixels[0]['lon'] = 10.
# 1
pixels[1]['month'] = 'M06'
pixels[1]['sza'] = np.array([40,41,43,50])
pixels[1]['vza'] = np.array([20,20,20,20])
pixels[1]['phi'] = np.array([40,50,60,70])
```

```
pixels[1]['lat'] = 35.
pixels[1]['lon'] = 28.
# 2
pixels[2]['month'] = 'M01'
pixels[2]['sza'] = np.array([30,31,33])
pixels[2]['vza'] = np.array([30,30,30])
pixels[2]['phi'] = np.array([74,75,76])
pixels[2]['lat'] = 10.
pixels[2]['lon'] = 5.

# -
# - Processing for each pixel
# -
for isite in range(len(pixels)):
    data = pixels[isite]
    month = months[1][months[0].index(data['month'])]
    szas, vzas, phis = data['sza'], data['vza'], data['phi']
    lat, lon = data['lat'],data['lon']
    LatMin,LatMax = lat-delta_deg, lat+delta_deg
    LonMin,LonMax = lon-delta_deg, lon+delta_deg

    request_dict = { 'fieldOperationType': 'spectrum',
                     'fieldSunZenith': 0,
                     'fieldViewZenith': 0,
                     'fieldRelAzimuth': 0,
                     'fieldMonth': month,
                     'fieldComputeError': False,
                     'fieldLatMin'    : LatMin,
                     'fieldLonMin'    : LonMin,
                     'fieldLatMax'    : LatMax,
                     'fieldLonMax'    : LonMax,
                     'fieldDefineInstrument': parasol_inst
                     }

    # initialize the job
    print 'init job/config'
    job = adam.adam_job( cfg=my_config) #, job_output_dir = 'TOTO' )
    job.validate_input(request_dict)
    job.load_data()

    # - spectral calculations
    job.process_reflectance()


    # initialize outputs
    for ilmbd in range(len(lmbds_MODIS)):
        data['r%i_MODIS'%(lmbds_MODIS[ilmbd])] = job.data['ref_land'][0,0,ilmbd]
    for ilmbd in range(len(lmbds)):

        data['r%i_ADAM_ref'%(lmbds[ilmbd])] = job.data['reflectance'][0,0,ilmbd]


    # - SDR calculation
    request_dict['fieldSunZenith'] = szas  # array
    request_dict['fieldViewZenith'] = vzas # array
    request_dict['fieldRelAzimuth'] = phis # array
    job.validate_input(request_dict)
    job.process_brdf()

    for ilmbd in range(len(lmbds)):
        data['r%i_ADAM_SDR'%(lmbds[ilmbd])] = job.data['SDR'][0,0,ilmbd,:]
```

```
# -
# - Sauvegarde
# -

cPickle.dump(pixels,file(os.path.join(my_output_root_dir,'Pixels_ADAM_V3_PARASOL.pickle'),'w'))
```

1.6.7 calc_SDR_in_CALIPSO.py

Script using the ADAM Toolkit demonstrating how calculating the surface reflectance in the CALIPSO band at 532 nm for an illumination at zenith (for 12 months).

The calculation is performed for all land points: this necessitate to modify in adam_config.py self.limit_square_degrees = 410 to self.limit_square_degrees = 3600*1800. The calculations are restricted to land pixels only.

The contents of calc_SDR_in_CALIPSO.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : calc_SDR_in_CALIPSO.py
description     : Script using the ADAM Toolkit demonstrating how calculating the surface reflecta
                  in the CALIPSO band at 532 nm for an illumination at zenith (for 12 months)

                  The calculation is performed for all land points: this necessitate to modify in
                  self.limit_square_degrees =  410 to self.limit_square_degrees =  3600*1800
                  The calculations are restricted to land pixels only

version         : 3.0
usage           : python calc_SDR_in_CALIPSO.py
notes           :
"""


import sys,os
import numpy as np
from optparse import OptionParser
import netCDF4

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')


# import the main adam library
import adam
import adam_config


# -
# - Configuration
# -

my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3'
my_output_root_dir = '/home/scratch01/cbacour/CALIPSO_SDR/'
my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di
months = ['jan', 'feb', 'mar', 'apr', 'may', 'jun', 'jul', 'aug', 'sep', 'oct', 'nov', 'dec']

# -
# - Loop over months
# -
for imonth in range(len(months)):
```

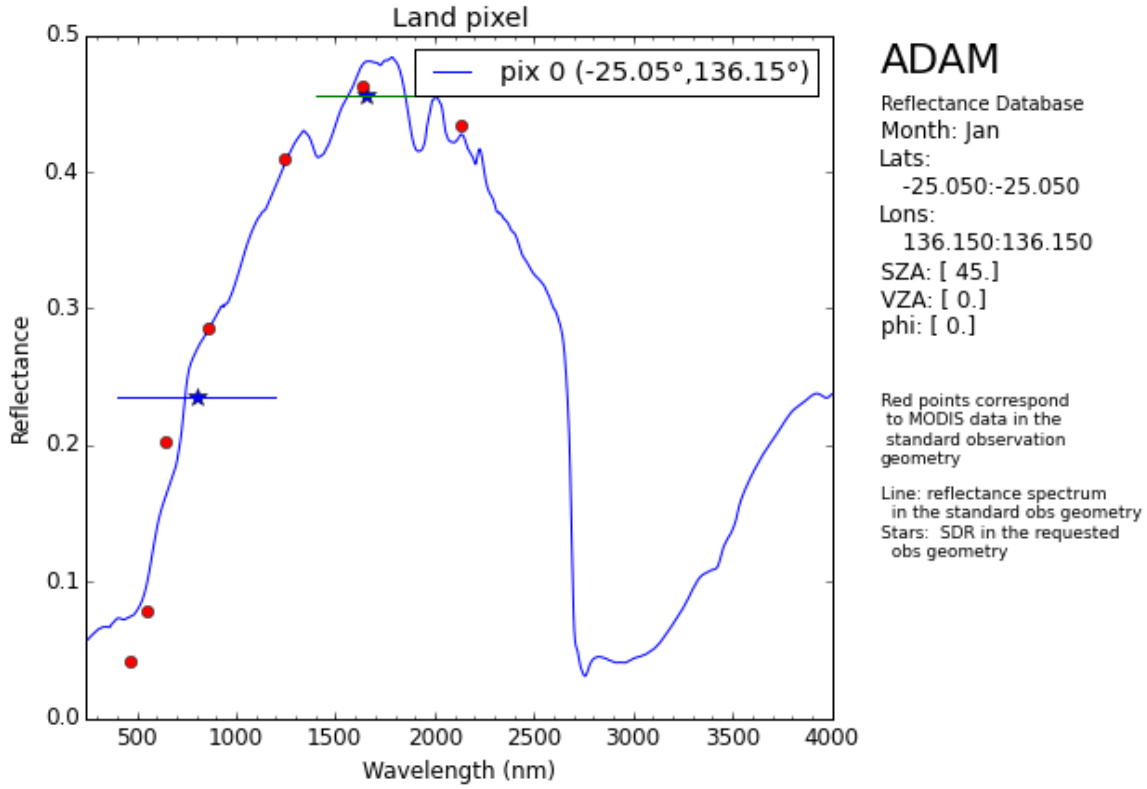
```
month = months[imonth]
print "# Treatment of month %s" %(month.upper())

# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = {'fieldOperationType': 'spectrum',
                'fieldMonth'        : month,
                'fieldSunZenith'    : 0,
                'fieldViewZenith'   : 0,
                'fieldRelAzimuth'   : 0,
                'fieldInstrument'   : 'calipso',
                'fieldComputeError' : False,

                # global land
                'fieldLatMin'    : -90.,
                'fieldLonMin'    : -180.,
                'fieldLatMax'    : 90.,
                'fieldLonMax'    : 180.
                }


file_out = os.path.join(my_output_root_dir,'ADAM_CALIPSO_%02i.nc'%(imonth+1,))

# initialise the job, providing us with a blank object ('job') to work with
job = adam.adam_job( cfg=my_config )
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')
print '  loading data..'
job.load_data()

# restrict the calculations to land pixels only
job.data['idx_ocean'] = []
job.data['idx_snow'] = []

# print a little output to prove we know where we are in the world
print "  %s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                             len(job.data['idx_ocean']),
                                                             len(job.data['idx_snow']))



# call the convenience functions to populate the various variables
print '  calculate reflectance..'
job.process_reflectance()
job.process_brdf()


# - Save as a netCDF file
print '  saving netcdf..'
nlat = len(job.data['latitude'])
nlon = len(job.data['longitude'])

fid = netCDF4.Dataset(file_out,'w')
fid.createDimension('band', len(job.lmbd))
fid.createDimension('latitude',  nlat)
fid.createDimension('longitude', nlon)
varLat = fid.createVariable('lat',job.data['latitude'].dtype,('latitude'), zlib = True)
varLon = fid.createVariable('lon',job.data['longitude'].dtype,('longitude'), zlib = True)
varRef = fid.createVariable('ref',job.data['reflectance'].dtype,('longitude','latitude','band
varSDR = fid.createVariable('SDR',job.data['SDR'].dtype,('longitude','latitude','band'), zlib
varLat[:] = job.data['latitude']
```

```
    varLon[:] = job.data['longitude']
    varRef[:,:] = job.data['reflectance']
    varSDR[:,:] = job.data['SDR']
    fid.close()

    print '  done.'
```

1.6.8 spectrum_graph_broadBands.py

Script using the ADAM Toolkit demonstrating making a spectrum graph. The scripts calculate the SDR over the user defined broad bands for an ensemble of pixels (contiguous area) for one observation geometry.

Example graph output should look similar to this:



The contents of spectrum_graph_broadBands.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : spectrum_graph_broadBands.py
description     : Script using the ADAM Toolkit demonstrating making a spectrum graph
                  The scripts calculate the SDR over the user defined broad bands for an ensemble
                  for one observation geometry
version         : 3.0
usage           : python spectrum_graph_broadbands.py
notes           :
"""


import sys
import numpy as np

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
```

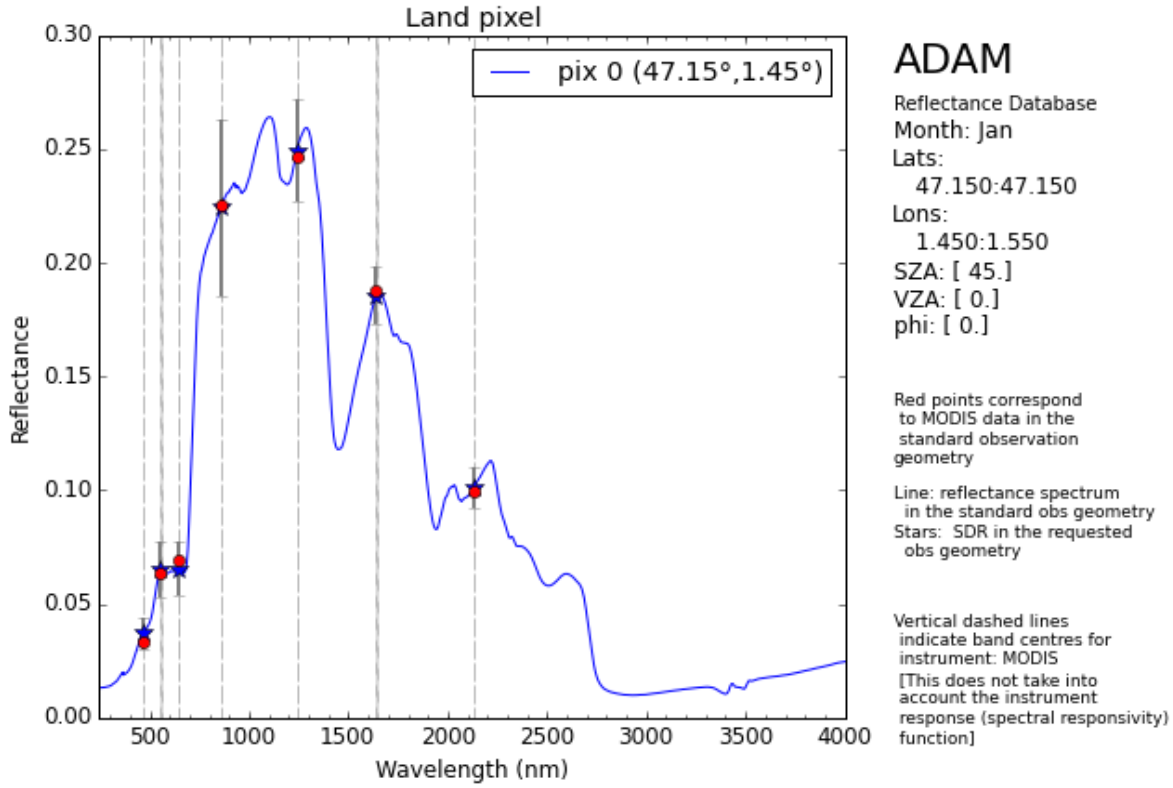
```
    sys.path.insert(0, '../')


# -
# - Configuration
# -
# import the main adam library
import adam
import adam_config


# input directory containing
#  - the ADAM-V3 netCDF files
#  - the ascii/ directory containing additional text files required by the API
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'

# output directory
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'


my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_API'

# -
# - API parameters
# -
# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = {'fieldOperationType': 'spectrum',
                'fieldMonth'        : 'jan',
                'fieldSunZenith'    : 45,
                'fieldViewZenith'   : 0,
                'fieldRelAzimuth'   : 0,
                'fieldComputeError': False,
                'fieldSpectralDomains': '400-1200 nm; 1400-1900',

                # soil
                'fieldLatMin'    : -25.1,
                'fieldLatMax'   : -25,
                'fieldLonMin'   : 136.1,
                'fieldLonMax'   : 136.2

                }



# -
# - Processing
# -
# initialise the job, providing us with a blank object ('job') to work with
print 'initialise job..'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir ) # here we provide the job_ou

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')

# now we are ready to load the data from our source database into our job object
print 'loading data..'
```

```
job.load_data()


# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))


# call the convenience functions to populate the various variables

# - spectral extrapolation of the normalized MODIS reflectance
print 'calculate reflectance..'
# populates job.data['reflectance']
job.process_reflectance()

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'calculate brdf..'
job.process_brdf()

# - now we can make our graphs
print 'make graphs..'

# graphs are created differently depending on how many pixels are to be rendered
# if there are only a few pixels we generate graphs depicting individual pixels
if job.is_pixel_request():
    print 'generating pixel graphs'
    # unlike ocean and snow, land spectrum graphs are treated one graph per pixel
    for i, pixel_index in enumerate(job.data['idx_land']):
        idx = [pixel_index]
        # the land error matrix is only populated for land pixels, so it uses a different counter
        job.graph_main_ref_spectra(case='pixels', indices=idx, title='Land pixel')

    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_snow'],  title='Snow pixel st
    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_ocean'], title='Ocean pixel s
else:
    # or if there are many pixels we take the mean/max/min and graph only those stats
    print 'generating stats graphs'
    job.calculate_stats(job.data['SDR'])
    job.graph_main_ref_spectra(case='stats')


# - Output netCDF file
print 'saving netcdf..'
job.save_netcdf()


print 'done.'
```

1.6.9 spectrum_graph_instrument.py

Script using the ADAM Toolkit demonstrating making a spectrum graph. The scripts calculate the SDR in the spectral bands of a predefined instrument (adam_config.py) for an ensemble of pixels (contiguous area) and for one observation geometry. The uncertainty on the SDR is calculated.

Example graph output should look similar to this:



The contents of spectrum_graph_instrument.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : spectrum_graph_instrument.py
description     : Script using the ADAM Toolkit demonstrating making a spectrum graph
                  The scripts calculate the SDR in the spectral bands of a predefined instrument
                  for an ensemble of pixels (contiguous area) and for one observation geometry.
                  The uncertainty on the SDR is calculated
version         : 3.0
usage           : python spectrum_graph_instrument.py
notes           :
"""


import sys
import numpy as np

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')


# -
# - Configuration
# -
# import the main adam library
import adam
import adam_config


# input directory containing
#  - the ADAM-V3 netCDF files
#  - the ascii/ directory containing additional text files required by the API
```

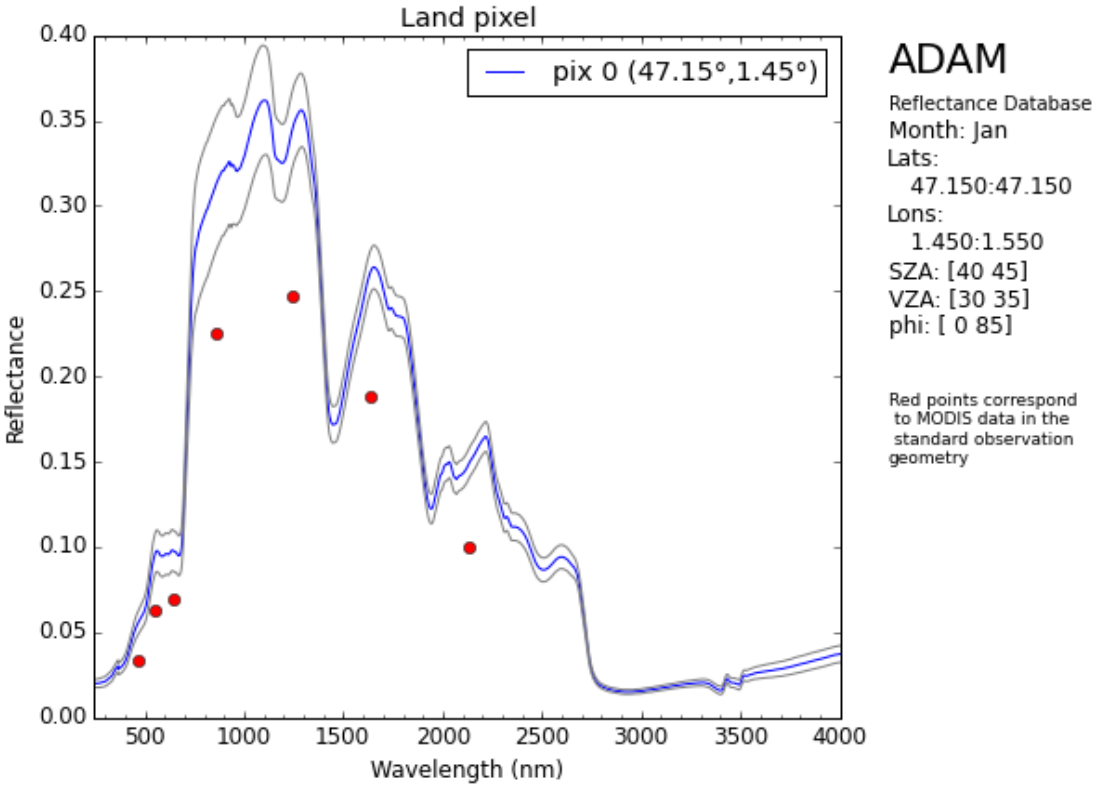
```
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'

# output directory
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'


my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_API'

# -
# - API parameters
# -
# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = {'fieldOperationType': 'spectrum',
                'fieldMonth'        : 'jan',
                'fieldSunZenith'    : 45,
                'fieldViewZenith'   : 0,
                'fieldRelAzimuth'   : 0,
                'fieldComputeError' : True,
                'fieldInstrument': 'modis',

# land
                 'fieldLatMin'   : 47.2,
                 'fieldLatMax'   : 47.1,
                 'fieldLonMin'   : 1.5,
                 'fieldLonMax'   : 1.505

                 }



# -
# - Processing
# -
# initialise the job, providing us with a blank object ('job') to work with
print 'initialise job..'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir ) # here we provide the job_ou

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')

# now we are ready to load the data from our source database into our job object
print 'loading data..'
job.load_data()


# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))


# call the convenience functions to populate the various variables

# - spectral extrapolation of the normalized MODIS reflectance
print 'calculate reflectance..'
# populates job.data['reflectance']
```

```
job.process_reflectance()

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'calculate brdf..'
job.process_brdf()

# - now we can make our graphs
print 'make graphs..'

# graphs are created differently depending on how many pixels are to be rendered
# if there are only a few pixels we generate graphs depicting individual pixels
if job.is_pixel_request():
    print 'generating pixel graphs'
    # unlike ocean and snow, land spectrum graphs are treated one graph per pixel
    for i, pixel_index in enumerate(job.data['idx_land']):
        idx = [pixel_index]
        # the land error matrix is only populated for land pixels, so it uses a different counter
        job.graph_main_ref_spectra(case='pixels', indices=idx, title='Land pixel')

    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_snow'],  title='Snow pixel sta
    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_ocean'], title='Ocean pixel st
else:
    # or if there are many pixels we take the mean/max/min and graph only those stats
    print 'generating stats graphs'
    job.calculate_stats(job.data['SDR'])
    job.graph_main_ref_spectra(case='stats')


# - Output netCDF file
print 'saving netcdf..'
job.save_netcdf()


print 'done.'
```

1.6.10 spectrum_graph_multidir.py

Script using the ADAM Toolkit demonstrating making a spectrum graph. The scripts calculate the SDR over the 240-4000 spectral domain for an ensemble of pixels (contiguous area) for several observation geometries.

Example graph output should look similar to this:



The contents of spectrum_graph_multidir.py are as follows:

```
# -*- coding: utf-8 -*-
"""
title           : spectrum_graph_multidir.py
description     : Script using the ADAM Toolkit demonstrating making a spectrum graph
                  The scripts calculate the SDR over the 240-4000 spectral domain for an ensemble
                  for several observation geometries
version         : 3.0
usage           : python spectrum_graph_multidir.py
notes           :
"""


import sys
import numpy as np

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')


# -
# - Configuration
# -
# import the main adam library
import adam
import adam_config


# input directory containing
#  - the ADAM-V3 netCDF files
#  - the ascii/ directory containing additional text files required by the API
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'
```

```
# output directory
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'


my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_API'

# -
# - API parameters
# -
# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = {'fieldOperationType': 'spectrum',
                'fieldMonth'        : 'jan',
                'fieldSunZenith'    : np.array([40,45]),     # the observation geometries are pro
                'fieldViewZenith'   : np.array([30,35]),
                'fieldRelAzimuth'   : np.array([0,85]),
                'fieldComputeError' : True,

# land
                 'fieldLatMin'   : 47.2,
                 'fieldLatMax'   : 47.1,
                 'fieldLonMin'   : 1.5,
                 'fieldLonMax'   : 1.505



 # ocean
                # 'fieldLatMin'   : -15.6,
                # 'fieldLonMin'   : -15.6,
                # 'fieldLatMax'   : -15.,
                # 'fieldLonMax'   : -15.1

                }



# -
# - Processing
# -
# initialise the job, providing us with a blank object ('job') to work with
print 'initialise job..'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir ) # here we provide the job_ou

# now validate our request against certain logic rules, and populate any unpopulated required fie
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')

# now we are ready to load the data from our source database into our job object
print 'loading data..'
job.load_data()


# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))
```

```
# call the convenience functions to populate the various variables

# - spectral extrapolation of the normalized MODIS reflectance
print 'calculate reflectance..'
# populates job.data['reflectance']
job.process_reflectance()

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'calculate brdf..'
job.process_brdf()

# - now we can make our graphs
print 'make graphs..'

# graphs are created differently depending on how many pixels are to be rendered
# if there are only a few pixels we generate graphs depicting individual pixels
if job.is_pixel_request():
    print 'generating pixel graphs'
    # unlike ocean and snow, land spectrum graphs are treated one graph per pixel
    for i, pixel_index in enumerate(job.data['idx_land']):
        idx = [pixel_index]
        # the land error matrix is only populated for land pixels, so it uses a different counter
        job.graph_main_ref_spectra(case='pixels', indices=idx, title='Land pixel')

    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_snow'],  title='Snow pixel st
    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_ocean'], title='Ocean pixel s
else:
    # or if there are many pixels we take the mean/max/min and graph only those stats
    print 'generating stats graphs'
    job.calculate_stats(job.data['SDR'])
    job.graph_main_ref_spectra(case='stats')


# - Output netCDF file
print 'saving netcdf..'
job.save_netcdf()


print 'done.'
```

1.6.11 spectrum_graph_userInstrument.py

Script using the ADAM Toolkit demonstrating making a spectrum graph. The scripts calculate the SDR in the narrow spectral band of a User Defined instrument for an ensemble of pixels (contiguous area) and for one observation geometry.

The contents of spectrum_graph_userInstrument.py are as follows:

```
    # -*- coding: utf-8 -*-
"""
title           : spectrum_graph_userInstrument.py
description     : Script using the ADAM Toolkit demonstrating making a spectrum graph
                  The scripts calculate the SDR in the narrow spectral band of a User Defined ins
                  for an ensemble of pixels (contiguous area) and for one observation geometry
version         : 3.0
usage           : python spectrum_graph_userInstrument.py
notes           :
"""


import sys
```

```
import numpy as np

# this script may live in the test/ directory and the adam class my not be in the PATH,
# so we add this here to allow for that case
if not '../' in sys.path:
    sys.path.insert(0, '../')


# -
# - Configuration
# -
# import the main adam library
import adam
import adam_config


# input directory containing
#  - the ADAM-V3 netCDF files
#  - the ascii/ directory containing additional text files required by the API
my_path_data       = '/home/users/cbacour/LIB/PROJETS/ADAM/Inputs_API/ADAM_V3/'

# output directory
my_output_root_dir = '/home/scratch01/cbacour/ADAM/'


my_config = adam_config.adam_config(path_data = my_path_data, output_root_dir = my_output_root_di

# impose a standard the name for the output directory (data may be overwritten)
output_subdir = 'TESTS_API'

# -
# - API parameters
# -
# define the type of adam calculation we wish to perform..
# NOTE that you can build a request like this using the web site and copy / paste
# the 'request_dictionary' from the logfile into this script
request_dict = {'fieldOperationType': 'spectrum',
                'fieldMonth'        : 'jan',
                'fieldSunZenith'    : 45,
                'fieldViewZenith'   : 0,
                'fieldRelAzimuth'   : 0,
                'fieldDefineInstrument': ['MyInstrument-1Band',[532]],
                #'fieldDefineInstrument': ['MyInstrument-MultiBands',[400, 500, 600, 800, 900]],

# land

                'fieldLatMin'   : -6.,
                'fieldLatMax'   : -4,
                'fieldLonMin'   : 17.9,
                'fieldLonMax'   : 18.1

                }



# -
# - Processing
# -
# initialise the job, providing us with a blank object ('job') to work with
print 'initialise job..'
job = adam.adam_job( cfg=my_config, job_output_dir = output_subdir ) # here we provide the job_ou

# now validate our request against certain logic rules, and populate any unpopulated required fie
```

```
# with default values
if not job.validate_input(request_dict):
    raise Exception('request input invalid !')

# now we are ready to load the data from our source database into our job object
print 'loading data..'
job.load_data()


# print a little output to prove we know where we are in the world
print "%s land pixels, %s ocean pixels, %s snow pixels" % (len(job.data['idx_land']),
                                                           len(job.data['idx_ocean']),
                                                           len(job.data['idx_snow']))


# call the convenience functions to populate the various variables

# - spectral extrapolation of the normalized MODIS reflectance
print 'calculate reflectance..'
# populates job.data['reflectance']
job.process_reflectance()

# - calculates the spectro-directional reflectances in the requested observation geometry
print 'calculate brdf..'
job.process_brdf()

# - now we can make our graphs
print 'make graphs..'

# graphs are created differently depending on how many pixels are to be rendered
# if there are only a few pixels we generate graphs depicting individual pixels
if job.is_pixel_request():
    print 'generating pixel graphs'
    # unlike ocean and snow, land spectrum graphs are treated one graph per pixel
    for i, pixel_index in enumerate(job.data['idx_land']):
        idx = [pixel_index]
        # the land error matrix is only populated for land pixels, so it uses a different counter
        job.graph_main_ref_spectra(case='pixels', indices=idx, title='Land pixel')

    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_snow'],  title='Snow pixel st
    job.graph_main_ref_spectra(case='pixels', indices=job.data['idx_ocean'], title='Ocean pixel s
else:
    # or if there are many pixels we take the mean/max/min and graph only those stats
    print 'generating stats graphs'
    job.calculate_stats(job.data['SDR'])
    job.graph_main_ref_spectra(case='stats')


# - Output netCDF file
print 'saving netcdf..'
job.save_netcdf()


print 'done.'
```

1.7 Module Documentation

The following are links to the documentation for the individual modules in the ADAM Library.

For normal use you will only ever need to interface with the ADAM Job class, however more advanced usage will require understanding of the other components.

- *ADAM Job*: Main helper class coordinating ADAM calculations in a job-based manner. Used to interface with the other components.
- *Config*: Config class allowing for script portability between systems.
- *Process Reflectance*
- *Process BRDF*
- *Analysis*
- *Show*

CHAPTER

TWO

BACKGROUND

This documentataion serves as a manual for the API (Application Programming Interface) component of the ADAM Surface Reflectance Database.

The API allows users familuar with the Python programming language to write their own code that interfaces directly with the existing ADAM code. This allows for the most flexible and efficient use of the ADAM system.

For information relating to the ADAM project in more general terms, visit the website: http://adam.noveltis.com/

CHAPTER

THREE

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*