SIAC Documentation

Release 2.1.1

Feng Yin

Jul 11, 2020

CONTENTS:

This atmospheric correction method uses MODIS MCD43 BRDF product to get a coarse resolution simulation of earth surface. A model based on MODIS PSF is built to deal with the scale differences between MODIS and other sensors, and linear spectral mapping is used to map between different sensors spectrally. We uses the ECMWF CAMS prediction as a prior for the atmospheric states, coupling with 6S model to solve for the atmospheric parameters, then the solved atmospheric parameters are used to correct the TOA reflectances. The whole system is built under Bayesian theory and the uncertainty is propagated through the whole system. Since we do not rely on specific bands' relationship to estimate the atmospheric states, but instead a more generic and consistent way of inversion those parameters. The code can be downloaded from SIAC github directly and futrher updates will make it more independent and can be installed on different machines.

Development of this code has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687320, under project H2020 MULTIPLY. f this code has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 687320, under project H2020 MULTIPLY.

CHAPTER

ONE

SIAC

1.1 Introduction

Land surface reflectance is the fundamental variable for the most of earth observation (EO) missions, and corrections of the atmospheric disturbs from the cloud, gaseous, aerosol help to get accurate spectral description of earth surface. Unlike the previous empirical ways of atmospheric correction, we propose a data fusion method for atmospheric correction of satellite images, with an initial attempt to include the uncertainty information from different data source. It takes advantage of the high temporal resolution of MODIS observations to get BRDF description of the earth surface as the prior information of the earth surface property, uses the ECMWF CAMS Near-real-time as the prior information of the atmospheric sates, to get optimal estimations of the atmospheric parameters. The code is written in python and we have tested it with Sentinel 2, Landsat 8, Landsat 5, Sentinel 3 and MODIS data and it shows SIAC can correct the atmospheric effects reasonablely well.

1.2 Sentinel 2 and Landsat 8 correction examples

A page shows some correction samples.

A map shows the validation results against AERONET measurements. If you click points on the scatter plot, it will the comparison between the TOA and BOA refletance and you can drag to compare between them and click on the image to have spectral comparison over your clicked pixel as well.

CHAPTER

TWO

INSTALLATION

2.1 1. The standard python way

You can download the source code from the project website. Unpack the file you obtained and then run:

```
python setup.py install
```

2.2 2. Using pip

```
pip install SIAC
```

2.3 3. Using anaconda

```
conda install -c f0xy -c conda-forge siac
```

2.4 4. From code repository for developing

Installation from the recent stable code repository can be done by:

```
pip install https://github.com/multiply-org/atmospheric_correction/archive/master.zip
```

CHAPTER

THREE

QUICKSTART

The usage of SIAC for Sentinel 2 and Landsat 8 Top Of Atmosphere (TOA) reflectance is very straight forward:

Sentinel 2

```
from SIAC import SIAC_S2
SIAC_S2('/directory/where/you/store/S2/data/') # this can be either from AWS or Senitinel offical package
```

Landsat 8

```
from SIAC import SIAC_L8
SIAC_L8('/directory/where/you/store/L8/data/')
```

An example usage of SIAC over other sensors and the way to access all the ancillary data is shown with a jupyter notebook example.

CHAPTER

FOUR

SIAC DATA ACCESSING AND USAGE ON OTHER SENSORS

In this Chapter, I will introduce the SIAC (Sensor Invariant Atmospheric Correction) developed under the European Union's Horizon 2020 MULTIPLY project can be used to generate global **uncertainty quantified analysis ready datasets** after 2003, which covered by NASA Landsat 5-8 missions and ESA Senitinel 2 mission.

4.1 SIAC

This atmospheric correction method uses MODIS MCD43 BRDF product to get a coarse resolution simulation of earth surface. A model based on MODIS PSF is built to deal with the scale differences between MODIS and other sensors, and linear spectral mapping is used to map between different sensors spectrally. We uses the ECMWF CAMS prediction as a prior for the atmospheric states, coupling with 6S model to solve for the atmospheric parameters, then the solved atmospheric parameters are used to correct the TOA reflectances. The whole system is built under Bayesian theory and the uncertainty is propagated through the whole system. Since we do not rely on specific bands' relationship to estimate the atmospheric states, but instead a more generic and consistent way of inversion those parameters. The code can be downloaded from SIAC github directly and futrher updates will make it more independent and can be installed on different machines.

4.1.1 Inputs:

- MCD43A1: 16 days before and 16 days after the sensing date
- ECMWF CAMS Near Real Time prediction or MACC reanalysis: a time step of 3 hours with the start time of 00:00:00 over the date and a easier access option is hosted at http://www2.geog.ucl.ac.uk/~ucfafyi/cams/ but only after 01/04/2015, when Sentinel 2A was just lucnched.
- Global dem: Global DEM VRT file built from ASTGTM2 DEM, and a bash script under eles/ can be used to generate with the individual files, and here we use ASTER Global Digital Elevation Model V002 and a easier option of accessing the dataset with gdal vertual file system is hosted at http://www2.geog.ucl.ac.uk/~ucfafyi/eles/global_dem.vrt.
- Emulators: emulators for the 6S for different senros, can be found at: http://www2.geog.ucl.ac.uk/~ucfafyi/emus/

4.1.2 Outputs:

The outputs are the corrected BOA images saved as `B0*_sur.tif` for each band and uncertainty `B0*_sur_unc.tif`. `TOA_RGB.tif` and `BOA_RGB.tif` are generated for a visual check of correction results. They are all under the same folder as the TOA images.

4.2 Data access:

MCD43A1

The MODIS MCD43A1 Version 6 Bidirectional reflectance distribution function and Albedo (BRDF/Albedo) Model Parameters data set is a 500 meter daily 16-day product. The Julian date in the granule ID of each specific file represents the 9th day of the 16 day retrieval period, and consequently the observations are weighted to estimate the BRDF/Albedo for that day. The MCD43A1 algorithm, as is with all combined products, has the luxury of choosing the best representative pixel from a pool that includes all the acquisitions from both the Terra and Aqua sensors from the retrieval period. The MCD43A1 provides the three model weighting parameters (isotropic, volumetric, and geometric) for each of the MODIS bands 1 through 7 and the visible (vis), near infrared (nir), and shortwave bands used to derive the Albedo and BRDF products (MCD43A3 and MCD43A4). Along with the 3 dimensional parameter layers for these bands are the Mandatory Quality layers for each of the 10 bands. The MODIS BRDF/ALBEDO products have achieved stage 3 validation. (From the website)

We use the BRDF descriptors inverted from MODIS high temporal multi-angular observations to get simulation of surface reflectance by using the Landsat or Sentinel 2 scanning geometry, and the reason of using 32 days MCD43 is due to the gaps in the current MCD43 products which cause issues for the inversion of reliable atmospheric paraneters. This dataset has to be downloaded from the NASA Data Pool with username and passoword registered at EARTHDATA LOGIN. The function `get_MCD43.py` inside `util` can be used for a easier acess to the data, but remember to change the username and passoword in the `util/earthdata_auth` file:

```
!cat util/earthdata_auth
```

```
username
password
```

```
import sys
sys.path.insert(0, 'util')
from get_MCD43 import get_mcd43, find_files
from datetime import datetime
# the great gdal virtual file system and google cloud landsat public datasets
google_cloud_base = '/vsicurl/https://storage.googleapis.com/gcp-public-data-landsat/'
aoi = google_cloud_base + 'LE07/01/202/034/LE07_L1TP_202034_20060611_20170108_01_T1/
↪LE07_L1TP_202034_20060611_20170108_01_T1_B1.TIF'
obs_time = datetime(2006, 6, 11)
# based on time and aoi find the MCD43
# within 16 days temporal window
ret = find_files(aoi, obs_time, temporal_window = 16)
print(ret[0])
```

```
https://e4ftl01.cr.usgs.gov/MOTA/MCD43A1.006/2006.05.26/MCD43A1.A2006146.h17v05.006.
↪2016102175833.hdf
```

To download them and used them and creat a daily global VRT file:

```
#get_mcd43(aoi, obs_time, mcd43_dir = './MCD43/', vrt_dir = './MCD43_VRT')
```

```
ls ./MCD43/ ./MCD43_VRT/ ./MCD43_VRT/2006-05-27/*.vrt
```

```
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_Band1.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_Band2.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_Band3.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_Band4.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_Band5.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_Band6.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_Band7.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_nir.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_shortwave.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Band_Mandatory_Quality_vis.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band1.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band2.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band3.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band4.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band5.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band6.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band7.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_nir.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_shortwave.vrt
./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_vis.vrt

./MCD43/:
2006_05_26/
flist.txt
MCD43A1.A2006146.h17v05.006.2016102175833.hdf
MCD43A1.A2006147.h17v05.006.2016102184226.hdf
MCD43A1.A2006148.h17v05.006.2016102192740.hdf
MCD43A1.A2006149.h17v05.006.2016102201522.hdf
MCD43A1.A2006150.h17v05.006.2016102210019.hdf
MCD43A1.A2006151.h17v05.006.2016102215923.hdf
MCD43A1.A2006152.h17v05.006.2016102223051.hdf
MCD43A1.A2006153.h17v05.006.2016102225753.hdf
MCD43A1.A2006154.h17v05.006.2016102232936.hdf
MCD43A1.A2006155.h17v05.006.2016102235934.hdf
MCD43A1.A2006156.h17v05.006.2016103003425.hdf
MCD43A1.A2006157.h17v05.006.2016103010516.hdf
MCD43A1.A2006158.h17v05.006.2016103013644.hdf
MCD43A1.A2006159.h17v05.006.2016103020849.hdf
MCD43A1.A2006160.h17v05.006.2016103023658.hdf
MCD43A1.A2006161.h17v05.006.2016103030549.hdf
MCD43A1.A2006162.h17v05.006.2016103034045.hdf
MCD43A1.A2006163.h17v05.006.2016103041018.hdf
MCD43A1.A2006164.h17v05.006.2016103044452.hdf
MCD43A1.A2006165.h17v05.006.2016103051319.hdf
MCD43A1.A2006166.h17v05.006.2016103054336.hdf
MCD43A1.A2006167.h17v05.006.2016103062221.hdf
MCD43A1.A2006168.h17v05.006.2016103064243.hdf
MCD43A1.A2006169.h17v05.006.2016103123307.hdf
MCD43A1.A2006170.h17v05.006.2016103125809.hdf
MCD43A1.A2006171.h17v05.006.2016103131624.hdf
MCD43A1.A2006172.h17v05.006.2016103133334.hdf
MCD43A1.A2006173.h17v05.006.2016103134900.hdf
MCD43A1.A2006174.h17v05.006.2016103140345.hdf
MCD43A1.A2006175.h17v05.006.2016103141904.hdf
MCD43A1.A2006176.h17v05.006.2016103143227.hdf
```

(continues on next page)

(continued from previous page)

```
MCD43A1.A2006177.h17v05.006.2016103144345.hdf
MCD43A1.A2006178.h17v05.006.2016103150133.hdf
MCD43A1.A2011232.h17v05.006.2016135180015.hdf
MCD43A1.A2011233.h17v05.006.2016135180956.hdf
MCD43A1.A2011234.h17v05.006.2016135182004.hdf
MCD43A1.A2011235.h17v05.006.2016135183413.hdf
MCD43A1.A2011236.h17v05.006.2016135184405.hdf
MCD43A1.A2011237.h17v05.006.2016135185631.hdf
MCD43A1.A2011238.h17v05.006.2016135190844.hdf
MCD43A1.A2011239.h17v05.006.2016135192024.hdf
MCD43A1.A2011240.h17v05.006.2016135193355.hdf
MCD43A1.A2011241.h17v05.006.2016135194900.hdf
MCD43A1.A2011242.h17v05.006.2016135200116.hdf
MCD43A1.A2011243.h17v05.006.2016135201359.hdf
MCD43A1.A2011244.h17v05.006.2016135202542.hdf
MCD43A1.A2011245.h17v05.006.2016135203726.hdf
MCD43A1.A2011246.h17v05.006.2016135204754.hdf
MCD43A1.A2011247.h17v05.006.2016135210022.hdf
MCD43A1.A2011248.h17v05.006.2016135211314.hdf
MCD43A1.A2011249.h17v05.006.2016135212256.hdf
MCD43A1.A2011250.h17v05.006.2016137122909.hdf
MCD43A1.A2011251.h17v05.006.2016137125638.hdf
MCD43A1.A2011252.h17v05.006.2016137131949.hdf
MCD43A1.A2011253.h17v05.006.2016137134319.hdf
MCD43A1.A2011254.h17v05.006.2016137140557.hdf
MCD43A1.A2011255.h17v05.006.2016137143341.hdf
MCD43A1.A2011256.h17v05.006.2016137144846.hdf
MCD43A1.A2011257.h17v05.006.2016137151738.hdf
MCD43A1.A2011258.h17v05.006.2016137153854.hdf
MCD43A1.A2011259.h17v05.006.2016137155602.hdf
MCD43A1.A2011260.h17v05.006.2016137161759.hdf
MCD43A1.A2011261.h17v05.006.2016137164021.hdf
MCD43A1.A2011262.h17v05.006.2016137170721.hdf
MCD43A1.A2011263.h17v05.006.2016137172812.hdf
MCD43A1.A2011264.h17v05.006.2016137175047.hdf

./MCD43_VRT/:
2006-05-26/  2006-06-06/  2006-06-17/  2011-08-20/  2011-08-31/  2011-09-11/
2006-05-27/  2006-06-07/  2006-06-18/  2011-08-21/  2011-09-01/  2011-09-12/
2006-05-28/  2006-06-08/  2006-06-19/  2011-08-22/  2011-09-02/  2011-09-13/
2006-05-29/  2006-06-09/  2006-06-20/  2011-08-23/  2011-09-03/  2011-09-14/
2006-05-30/  2006-06-10/  2006-06-21/  2011-08-24/  2011-09-04/  2011-09-15/
2006-05-31/  2006-06-11/  2006-06-22/  2011-08-25/  2011-09-05/  2011-09-16/
2006-06-01/  2006-06-12/  2006-06-23/  2011-08-26/  2011-09-06/  2011-09-17/
2006-06-02/  2006-06-13/  2006-06-24/  2011-08-27/  2011-09-07/  2011-09-18/
2006-06-03/  2006-06-14/  2006-06-25/  2011-08-28/  2011-09-08/  2011-09-19/
2006-06-04/  2006-06-15/  2006-06-26/  2011-08-29/  2011-09-09/  2011-09-20/
2006-06-05/  2006-06-16/  2006-06-27/  2011-08-30/  2011-09-10/  2011-09-21/
```

```
!gdalinfo ./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band5.vrt
```

```
Driver: VRT/Virtual Raster
Files: ./MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band5.vrt
Size is 2400, 2400
Coordinate System is:
PROJCS["unnamed",
```

(continues on next page)

(continued from previous page)

```
    GEOGCS["Unknown datum based upon the custom spheroid",
        DATUM["Not specified (based on custom spheroid)",
            SPHEROID["Custom spheroid",6371007.181,0]],
        PRIMEM["Greenwich",0],
        UNIT["degree",0.0174532925199433]],
    PROJECTION["Sinusoidal"],
    PARAMETER["longitude_of_center",0],
    PARAMETER["false_easting",0],
    PARAMETER["false_northing",0],
    UNIT["Meter",1]]
Origin = (-1111950.519667000044137,4447802.078666999936104)
Pixel Size = (463.312716527916677,-463.312716527916677)
Corner Coordinates:
Upper Left  (-1111950.520, 4447802.079) ( 13d 3'14.66"W, 40d 0' 0.00"N)
Lower Left  (-1111950.520, 3335851.559) ( 11d32'49.22"W, 30d 0' 0.00"N)
Upper Right (       0.000, 4447802.079) (  0d 0' 0.01"E, 40d 0' 0.00"N)
Lower Right (       0.000, 3335851.559) (  0d 0' 0.01"E, 30d 0' 0.00"N)
Center      ( -555975.260, 3891826.819) (  6d 6'13.94"W, 35d 0' 0.00"N)
Band 1 Block=128x128 Type=Int16, ColorInterp=Gray
  NoData Value=32767
Band 2 Block=128x128 Type=Int16, ColorInterp=Gray
  NoData Value=32767
Band 3 Block=128x128 Type=Int16, ColorInterp=Gray
  NoData Value=32767
```

The great part of creating VRT files is that virtual global mosaic of MCD43 for different parameters and different times are created with a very small fraction of storage space, but eliminate a lot of troubles in dealing with different spatial resolutions, data formats and multi-tile coverage… Actually, the best way of storing the datasets is turn it into Cloud optimized GeoTIFF, which enables access of chuncks of data from a virtul mosaic to be possible and saves a lot of unnecessary downloading of data outside the area of interest. And this can be done easily with gdal as well:

```
#!/usr/bin/env python
import os
import sys
import gdal
from datetime import datetime
fname = './MCD43/MCD43A1.A2006146.h17v05.006.2016102175833.hdf'
try:
    g =  gdal.Open(fname)
except:
    raise IOError('File cannot opened!')
subs = [i[0] for i in g.GetSubDatasets()]

def translate(sub):
    ret = sub.split(':')
    path, para = ret[2].split('"')[1], ret[-1]
    base = '/'.join(path.split('/')[:-1])
    fname = path.split('/')[-1]
    day = datetime.strptime(fname.split('.')[1].split('A')[1], '%Y%j').strftime('%Y_
↪%m_%d')
    ret = fname.split('.')
    day = datetime.strptime(ret[-5].split('A')[1], '%Y%j').strftime('%Y_%m_%d')
    fname =  '_'.join(ret[:-2]) + '_%s.tif'%para
    fname = base + '/' + day + '/' + fname
    if os.path.exists(fname):
        pass
```

(continues on next page)

(continued from previous page)

```
    else:
        if os.path.exists(base + '/%s'%day):
            pass
        else:
           os.mkdir(base + '/%s'%day)
        gdal.Translate(fname, sub,creationOptions=["TILED=YES", "COMPRESS=DEFLATE"])
for sub in subs:
    translate(sub)
```

```
ls MCD43/2006_05_26/
```

```
flist.txt
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_Band1.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_Band2.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_Band3.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_Band4.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_Band5.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_Band6.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_Band7.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_nir.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_shortwave.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Band_Mandatory_Quality_vis.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band1.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band1.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band2.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band2.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band3.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band3.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band4.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band4.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band5.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band5.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band6.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band6.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band7.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band7.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_nir.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_nir.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_shortwave.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_shortwave.tif.aux.xml
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_vis.tif
MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_vis.tif.aux.xml
```

So we basically converted the MODIS HDF format into GeoTiff format and an important argument for the `gdal.translate` is `TILED=YES` which will make small chunk access of dataset to be possible.

```
#!/usr/bin/env python
import pylab as plt
import numpy as np
from reproject import reproject_data
# hear we try to reproject the RGB band
# from MCD43 to the aoi used above, whcih
# is just a url to the google cloud file
# we also use the vrt file we created
# as our source file and reproject them
# to the aoi with the same spatial resol.
```

(continues on next page)

(continued from previous page)

```
source = './MCD43_VRT/2006-05-27/MCD43_2006147_BRDF_Albedo_Parameters_Band1.vrt'

r = reproject_data(source, aoi).data[0] * 0.001
g = reproject_data(source.replace('Band1', 'Band4'), aoi).data[0] * 0.001
b = reproject_data(source.replace('Band1', 'Band3'), aoi).data[0] * 0.001

r[r>1] = np.nan
b[b>1] = np.nan
g[g>1] = np.nan
plt.imshow(np.array([r,g,b]).transpose(1,2,0)*4)
```

```
<IPython.core.display.Javascript object>
```

```
<matplotlib.image.AxesImage at 0x7fae7aabf518>
```

I have also put the created tif file into the UCL geography file server at http://www2.geog.ucl.ac.uk/~ucfafyi/test_files/2006_05_26/

```
from IPython.core.display import HTML
#HTML("http://www2.geog.ucl.ac.uk/~ucfafyi/test_files/2006_05_26/")
```

And if we change from the VRT file to the url to the tif files, it will also works!!!

```
url = '/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/test_files/2006_05_26/'
source = url + 'MCD43A1_A2006146_h17v05_006_BRDF_Albedo_Parameters_Band1.tif'

r = reproject_data(source, aoi).data[0] * 0.001
g = reproject_data(source.replace('Band1', 'Band4'), aoi).data[0] * 0.001
b = reproject_data(source.replace('Band1', 'Band3'), aoi).data[0] * 0.001

r[r>1]=np.nan
b[b>1] = np.nan
g[g>1] = np.nan
plt.imshow(np.array([r,g,b]).transpose(1,2,0)*4)
```

```
<IPython.core.display.Javascript object>
```

```
<matplotlib.image.AxesImage at 0x7fae7a228dd8>
```

And if we creat virtual global mosaic VRT file with those GeoTiff images, we can also access them with gdal and do the subset and reprojection easily. . . .And I will demonstrate it with the ASTGTM2 DEM. . .

Global DEM

I have downloaded most of the DEM images from NASA server and put them in the UCL server at: http://www2.geog.ucl.ac.uk/~ucfafyi/eles/ and a global DEM VRT file is generated with:

```
ls *.tif>file_list.txt
gdalbuildvrt -te -180 -90 180 90 global_dem.vrt -input_file_list file_list.txt
```

```
print(gdal.Info('/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/eles/global_dem.vrt',
↪showFileList=False))
```

```
Driver: VRT/Virtual Raster
Files: /vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/eles/global_dem.vrt
```

(continues on next page)

(continued from previous page)

```
Size is 1296000, 648000
Coordinate System is:
GEOGCS["WGS 84",
    DATUM["WGS_1984",
        SPHEROID["WGS 84",6378137,298.257223563,
            AUTHORITY["EPSG","7030"]],
        AUTHORITY["EPSG","6326"]],
    PRIMEM["Greenwich",0],
    UNIT["degree",0.0174532925199433],
    AUTHORITY["EPSG","4326"]]
Origin = (-180.000000000000000,90.000000000000000)
Pixel Size = (0.000277777777778,-0.000277777777778)
Corner Coordinates:
Upper Left  (-180.0000000,  90.0000000) (180d 0' 0.00"W, 90d 0' 0.00"N)
Lower Left  (-180.0000000, -90.0000000) (180d 0' 0.00"W, 90d 0' 0.00"S)
Upper Right ( 180.0000000,  90.0000000) (180d 0' 0.00"E, 90d 0' 0.00"N)
Lower Right ( 180.0000000, -90.0000000) (180d 0' 0.00"E, 90d 0' 0.00"S)
Center      (   0.0000000,  -0.0000000) (  0d 0' 0.00"E,  0d 0' 0.00"S)
Band 1 Block=128x128 Type=Int16, ColorInterp=Gray
```

```
source = '/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/eles/global_dem.vrt'
ele = reproject_data(source, aoi).data * 0.001
plt.imshow(ele)
```

```
<IPython.core.display.Javascript object>
```

```
<matplotlib.image.AxesImage at 0x7fae7a210908>
```

Instantly, we get the DEM with the same resolution and geographic coverage as the aoi, which means if we have the MCD43 in GeoTiff format and a global mosaic can be created for each day then the access of MCD43 data will be much easier, and this actully applies to all different kind of GIS datasets.

CAMS atmospheric composition data

As part of the European Copernicus programme on environmental monitoring, greenhouse gases, aerosols, and chemical species have been introduced in the ECMWF model allowing assimilation and forecasting of atmospheric composition. At the same time, the added atmospheric composition variables are being used to improve the Numerical Weather Prediction (NWP) system itself, most notably through the interaction with the radiation scheme and the use in observation operators for satellite radiance assimilation. (from the website)

In SIAC, the aerosol optical thickness (AOT) at 550nm, total column of water vapour (TCWV) and total column of Ozone(TCO_3) are used as priors for the atmospheric states. The data can be aquired through the official pages, but it needs to wait for the queue to process each time you request it, but actually the dataset is at a coarse grid and only take small storage space and again daily global mosaic for each parameter in tif format at UCL server at http://www2.geog.ucl.ac.uk/~ucfafyi/cams/. There is no need to process it for each user and do the subset each time. . . .

The api access to cams near real time:

```
#!/usr/bin/env python
import os
import sys
import gdal
from glob import glob
from ecmwfapi import ECMWFDataServer
server = ECMWFDataServer()
```

(continues on next page)

(continued from previous page)

```
from datetime import datetime, timedelta
para_names = 'tcwv,gtco3,aod550,duaod550,omaod550,bcaod550,suaod550'.split(',')
this_date = datetime(2015,4,26)
filename = "%s.nc"%this_date
if not os.path.exists(filename):
    server.retrieve({
        "class": "mc",
        "dataset": "cams_nrealtime",
        "date": "%s"%this_date,
        "expver": "0001",
        "levtype": "sfc",
        "param": "137.128/206.210/207.210/209.210/210.210/211.210/212.210",
        "step": "0/3/6/9/12/15/18/21/24",
        "stream": "oper",
        "time": "00:00:00",
        "type": "fc",
        "grid": "0.75/0.75",
        "area": "90/-180/-90/180",
        "format":"netcdf",
        "target": "%s.nc"%this_date,
    })
else:
    pass
header = '_'.join(filename.split('.')[0].split('-'))
if not os.path.exists(header):
    os.mkdir(header)
exists = glob(header+'/*.tif')
if len(sys.argv[2:])>0:
    list_para = sys.argv[2:]
else:
    list_para = para_names
temp = 'NETCDF:"%s":%s'
for i in list_para:
    if header + '/'+header+'_'+i+'.tif' not in exists:
        t = 'Translating %-31s to %-23s'%(temp%(filename,i), header+'_'+i+'.tif')
        print(t)
        gdal.Translate(header + '/'+header+'_'+i+'.tif', temp%(filename,i), outputSRS=
↪'EPSG:4326', creationOptions=["TILED=YES", "COMPRESS=DEFLATE"])
```

and reanalysis data:

```
#!/usr/bin/env python
import os
import sys
import gdal
from glob import glob
from ecmwfapi import ECMWFDataServer
server = ECMWFDataServer()
from datetime import datetime, timedelta
para_names = 'tcwv,gtco3,aod550,duaod550,omaod550,bcaod550,suaod550'.split(',')
this_date = datetime(2012,4,26)
filename = "%s.nc"%this_date
if not os.path.exists(filename):
    server.retrieve({
        "class": "mc",
        "dataset": "macc",
        "date": "%s"%this_date,
```

(continues on next page)

(continued from previous page)

```
        "expver": "rean",
        "levtype": "sfc",
        "param": "137.128/206.210/207.210/209.210/210.210/211.210/212.210",
        "step": "0/3/6/9/12/15/18/21/24",
        "stream": "oper",
        "time": "00:00:00",
        "type": "fc",
        "grid": "0.75/0.75",
        "area": "90/-180/-90/180",
        "format":"netcdf",
        "target": "%s.nc"%this_date,
    })
else:
    pass
header = '_'.join(filename.split('.')[0].split('-'))
if not os.path.exists(header):
    os.mkdir(header)
exists = glob(header+'/*.tif')
if len(sys.argv[2:])>0:
    list_para = sys.argv[2:]
else:
    list_para = para_names
temp = 'NETCDF:"%s":%s'
for i in list_para:
    if header + '/'+header+'_'+i+'.tif' not in exists:
        t = 'Translating %-31s to %-23s'%(temp%(filename,i), header+'_'+i+'.tif')
        print(t)
        gdal.Translate(header + '/'+header+'_'+i+'.tif', temp%(filename,i), outputSRS=
↪'EPSG:4326', creationOptions=["TILED=YES", "COMPRESS=DEFLATE"])
```

```
# here we test with subset of global AOT 550
# over the aoi
source = '/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/cams/2015_09_08/2015_09_08_
↪aod550.tif'
g = gdal.Open(source)
b1 = g.GetRasterBand(1)
scale, offset = b1.GetScale(), b1.GetOffset()
g = None
g = reproject_data(source, aoi).g
aot = scale * g.GetRasterBand(3).ReadAsArray() + offset
plt.imshow(aot)
```

```
<IPython.core.display.Javascript object>
```

```
<matplotlib.image.AxesImage at 0x7fae7a1857f0>
```

TOA reflectance

Unlike many AC method, one needs to convert reflectance to radiance and back to reflectance after AC, SIAC only need the reflectance ranging from 0-1 (or 10, 100, 1000… but you need to give the scale and offset vlaues). To access the Landsat datasets, you can download it from USGS or google has mirrored the whole Landsat datesets which can be accessed from the google public Landsat dateset with url: `https://storage.googleapis.com/gcp-public-data-landsat/`.

Here we show example of accessing different Landsat mission images, with the AOI in Spain, and we stored it as a Geojson file.

```
import requests
# Landsat 5
base = 'https://storage.googleapis.com/gcp-public-data-landsat/'
tile = 'LT05/01/202/034/LT05_L1TP_202034_20110905_20161006_01_T1/'
bands = ['LT05_L1TP_202034_20110905_20161006_01_T1_B%d.TIF'%i for i in [1,2,3,4,5,7]]
metadata = requests.get(base + tile + 'LT05_L1TP_202034_20110905_20161006_01_T1_MTL.
↪txt').content.decode()
scale = []
off = []
for i in metadata.split('\n'):
    if 'REFLECTANCE_ADD_BAND' in i:
        print(i)
        off.append(float(i.split(' = ')[1]))
    if 'REFLECTANCE_MULT_BAND' in i:
        print(i)
        scale.append(float(i.split(' = ')[1]))
rgb = []
for i in [0,1,2]:
    g = gdal.Warp('', '/vsicurl/' + base + tile + bands[i], format = 'MEM',␣
↪warpOptions = \
                ['NUM_THREADS=ALL_CPUS'],srcNodata = 0, dstNodata=0, cutlineDSName=
↪'aoi.json', \
                cropToCutline=True, resampleAlg = gdal.GRIORA_NearestNeighbour)
    rgb.append(g.ReadAsArray())
rgb = np.array(scale[:3])[...,None, None]*rgb + np.array(off[:3])[...,None, None]
alpha = np.any(rgb < 0, axis=0)
```

```
REFLECTANCE_MULT_BAND_1 = 1.2582E-03
REFLECTANCE_MULT_BAND_2 = 2.6296E-03
REFLECTANCE_MULT_BAND_3 = 2.2379E-03
REFLECTANCE_MULT_BAND_4 = 2.7086E-03
REFLECTANCE_MULT_BAND_5 = 1.8340E-03
REFLECTANCE_MULT_BAND_7 = 2.5458E-03
REFLECTANCE_ADD_BAND_1 = -0.003756
REFLECTANCE_ADD_BAND_2 = -0.007786
REFLECTANCE_ADD_BAND_3 = -0.004746
REFLECTANCE_ADD_BAND_4 = -0.007377
REFLECTANCE_ADD_BAND_5 = -0.007472
REFLECTANCE_ADD_BAND_7 = -0.008371
```

```
# since landsat angles has to be produced
# from the angular text file
ang = requests.get(base + tile + 'LT05_L1TP_202034_20110905_20161006_01_T1_ANG.txt').
↪content
with open('landsat/landsat_ang/LT05_L1TP_202034_20110905_20161006_01_T1_ANG.txt', 'wb
↪') as f:
    f.write(ang)

header = 'LT05_L1TP_202034_20110905_20161006_01_T1_'
import os
from glob import glob
from multiprocessing import Pool
import subprocess
from functools import partial
cwd = os.getcwd()
#header += cwd
os.chdir('landsat/landsat_ang/')
```

(continues on next page)

(continued from previous page)

```
def translate_angle(band, header):
    subprocess.call([cwd+'/util/landsat_angles/landsat_angles', \
                     header + 'ANG.txt', '-B', '%d'%band])
    inp = 'angle_sensor_B%02d.img'%band
    oup = header+ 'VZA_VAA_B%02d.TIF'%band
    if os.path.exists(oup):
        os.remove(oup)
    gdal.Translate(oup, inp, creationOptions = \
                   ['COMPRESS=DEFLATE', 'TILED=YES'], noData='-32767').FlushCache()
    if not os.path.exists(header + 'SZA_SAA.TIF'):
        gdal.Translate(header + 'SZA_SAA.TIF', 'angle_solar_B01.img', creationOptions␣
↪= \
                       ['COMPRESS=DEFLATE', 'TILED=YES'], noData='-32767').FlushCache()
p = Pool()
par = partial(translate_angle, header=header)
p.map(par, range(1,8))
list(map(os.remove, glob('angle_s*_B*.img*')))
os.chdir(cwd)
```

```
sza, saa = gdal.Warp('', 'landsat/landsat_ang/'+ header + 'SZA_SAA.TIF',format = 'MEM
↪', warpOptions = \
              ['NUM_THREADS=ALL_CPUS'],srcNodata = 0, dstNodata=0, cutlineDSName=
↪'aoi.json', \
              cropToCutline=True, resampleAlg = gdal.GRIORA_NearestNeighbour).
↪ReadAsArray()/100.
rgb = rgb / np.cos(np.deg2rad(sza))
rgb[rgb<0] = np.nan
rgba = np.r_[rgb, ~alpha[None, ...]]
```

```
plt.figure(figsize=(8,8))
plt.imshow(rgba.transpose(1,2,0)*2)
```

```
<IPython.core.display.Javascript object>
```

```
<matplotlib.image.AxesImage at 0x7fae7aa1cf60>
```

```
# We save all the remote file to local
# also convert it to reflectance...

for i in range(6):
    g = gdal.Warp('', '/vsicurl/' + base + tile + bands[i], format = 'MEM',␣
↪warpOptions = \
              ['NUM_THREADS=ALL_CPUS'],srcNodata = 0, dstNodata=0, cutlineDSName=
↪'aoi.json', \
              cropToCutline=True, resampleAlg = gdal.GRIORA_NearestNeighbour)
    data = (g.ReadAsArray() * scale[i] + off[i])/np.cos(np.deg2rad(sza))
    data[data<0] = -9999
    driver = gdal.GetDriverByName('GTiff')
    ds = driver.Create('landsat/' + bands[i], data.shape[1], data.shape[0], 1, \
                       gdal.GDT_Float32, options=["TILED=YES", "COMPRESS=DEFLATE"])
    ds.SetGeoTransform(g.GetGeoTransform())
    ds.SetProjection(g.GetProjectionRef())
    ds.GetRasterBand(1).WriteArray(data)
    ds.FlushCache()
    ds=None
```

(continues on next page)

(continued from previous page)

```
g = gdal.Warp('landsat/' + bands[0].replace('B1', 'BQA'), '/vsicurl/' + base + tile +␣
↪bands[0].replace('B1', 'BQA'), format = 'GTiff', warpOptions = \
              ['NUM_THREADS=ALL_CPUS'],srcNodata = 0, dstNodata=0, cutlineDSName=
↪'aoi.json', \
              cropToCutline=True, resampleAlg = gdal.GRIORA_NearestNeighbour,␣
↪creationOptions \
                  = ['COMPRESS=DEFLATE', 'TILED=YES'])
g.FlushCache()
aoi_mask = np.isnan(data)
```

```
plt.imshow(gdal.Open('landsat/LT05_L1TP_202034_20110905_20161006_01_T1_B1.TIF').
↪ReadAsArray())
plt.colorbar()
```

```
<IPython.core.display.Javascript object>
```

```
<matplotlib.colorbar.Colorbar at 0x7fadf404d7b8>
```

We now have the angle and reflectance for this AOI, we can pass them into SIAC to run the atmospheric correction.

4.3 Run SIAC

In reality, we need the emulators for this sensor, but at the moment we do not have, so we instead use Landsat 8 emultors just for demostration purpose.

```
import numpy as np
from SIAC.the_aerosol import solve_aerosol
from SIAC.the_correction import atmospheric_correction
sensor_sat = 'OLI', 'L8'
toa_bands = ['landsat/'+i for i in bands]
view_angles = ['landsat/landsat_ang/LT05_L1TP_202034_20110905_20161006_01_T1_VZA_VAA_
↪%s.TIF'%i  \
               for i in ['B01', 'B02', 'B03', 'B04', 'B05', 'B07']]
sun_angles = 'landsat/landsat_ang/LT05_L1TP_202034_20110905_20161006_01_T1_SZA_SAA.TIF
↪'
cloud_mask = gdal.Open('landsat/' + bands[0].replace('B1', 'BQA')).ReadAsArray()
cloud_mask = ~((cloud_mask  >= 672) & ( cloud_mask <= 684)) | aoi_mask
band_wv    = [469, 555, 645, 859, 1640, 2130]
obs_time = datetime(2011, 9, 5, 10, 51, 11)
#get_mcd43(toa_bands[0], obs_time, mcd43_dir = './MCD43/', vrt_dir = './MCD43_VRT')
band_index = [1,2,3,4,5,6]
aero = solve_aerosol(sensor_sat,toa_bands,band_wv, band_index,\
                     view_angles,sun_angles,obs_time,cloud_mask,aot_prior = 0.05, \
                     aot_unc=0.1, mcd43_dir= './MCD43_VRT/',gamma=10., ref_scale = 1.,
↪ ref_off = 0., a_z_order=0)
aero._solving()

base = 'landsat/LT05_L1TP_202034_20110905_20161006_01_T1_'
aot = base + 'aot.tif'
tcwv = base + 'tcwv.tif'
tco3 = base + 'tco3.tif'
aot_unc = base + 'aot_unc.tif'
tcwv_unc = base + 'tcwv_unc.tif'
```

(continues on next page)

(continued from previous page)

```
tco3_unc = base + 'tco3_unc.tif'
rgb = [toa_bands[2], toa_bands[1], toa_bands[0]]
atmo = atmospheric_correction(sensor_sat, toa_bands, band_index,view_angles,sun_
↪angles, \
                              aot = aot, cloud_mask = cloud_mask, tcwv = tcwv, tco3 =␣
↪tco3, \
                              aot_unc = aot_unc, tcwv_unc = tcwv_unc, tco3_unc = tco3_
↪unc, \
                              rgb = rgb, ref_scale = 1., ref_off = 0., a_z_order   =␣
↪0)
ret = atmo._doing_correction()
```

```
2018-09-13 18:41:35,579 - AtmoCor - INFO - Set AOI.
2018-09-13 18:41:35,581 - AtmoCor - INFO - Get corresponding bands.
2018-09-13 18:41:35,581 - AtmoCor - INFO - Slice TOA bands based on AOI.
2018-09-13 18:41:36,815 - AtmoCor - INFO - Parsing angles.
2018-09-13 18:41:36,821 - AtmoCor - INFO - Mask bad pixeles.
2018-09-13 18:41:37,122 - AtmoCor - INFO - Get simulated BOA.
2018-09-13 18:42:25,089 - AtmoCor - INFO - Get PSF.
2018-09-13 18:42:28,284 - AtmoCor - INFO - Solved PSF: 8.67, 11.33, 0, 4, 1, and R␣
↪value is: 0.961.
2018-09-13 18:42:28,288 - AtmoCor - INFO - Get simulated TOA reflectance.
2018-09-13 18:42:31,141 - AtmoCor - INFO - Filtering data.
2018-09-13 18:42:31,199 - AtmoCor - INFO - Loading emulators.
2018-09-13 18:42:33,766 - AtmoCor - INFO - Reading priors and elevation.
2018-09-13 18:42:42,059 - MultiGrid solver - INFO - MultiGrid solver in process...
2018-09-13 18:42:42,060 - MultiGrid solver - INFO - Total 5 level of grids are going␣
↪to be used.
2018-09-13 18:42:42,061 - MultiGrid solver - INFO - [94mOptimizing at grid level 1
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
2018-09-13 18:42:43,462 - MultiGrid solver - INFO - [92mb'CONVERGENCE: REL_REDUCTION_
↪OF_F_<=_FACTR*EPSMCH'
2018-09-13 18:42:43,463 - MultiGrid solver - INFO - [92mIterations: 4
2018-09-13 18:42:43,463 - MultiGrid solver - INFO - [92mFunction calls: 23
2018-09-13 18:42:43,464 - MultiGrid solver - INFO - [94mOptimizing at grid level 2
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
2018-09-13 18:42:46,115 - MultiGrid solver - INFO - [92mb'CONVERGENCE: REL_REDUCTION_
↪OF_F_<=_FACTR*EPSMCH'
2018-09-13 18:42:46,116 - MultiGrid solver - INFO - [92mIterations: 3
2018-09-13 18:42:46,116 - MultiGrid solver - INFO - [92mFunction calls: 25
2018-09-13 18:42:46,117 - MultiGrid solver - INFO - [94mOptimizing at grid level 3
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

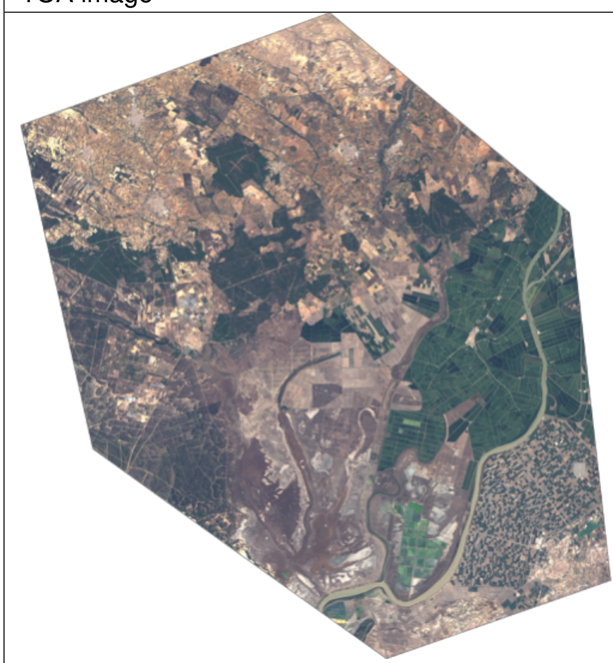
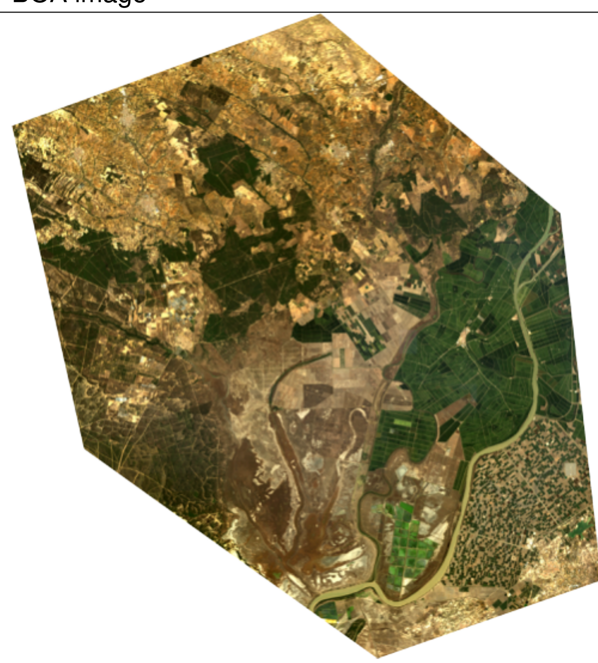
```
2018-09-13 18:42:57,283 - MultiGrid solver - INFO - [92mb'STOP: TOTAL NO. of f AND g␣
↪EVALUATIONS EXCEEDS LIMIT'
2018-09-13 18:42:57,283 - MultiGrid solver - INFO - [92mIterations: 8
2018-09-13 18:42:57,284 - MultiGrid solver - INFO - [92mFunction calls: 75
2018-09-13 18:42:57,284 - MultiGrid solver - INFO - [94mOptimizing at grid level 4
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
2018-09-13 18:43:09,654 - MultiGrid solver - INFO - [92mb'CONVERGENCE: REL_REDUCTION_
↪OF_F_<=_FACTR*EPSMCH'
2018-09-13 18:43:09,655 - MultiGrid solver - INFO - [92mIterations: 10
2018-09-13 18:43:09,655 - MultiGrid solver - INFO - [92mFunction calls: 36
2018-09-13 18:43:09,656 - MultiGrid solver - INFO - [94mOptimizing at grid level 5
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

```
2018-09-13 18:45:07,040 - MultiGrid solver - INFO - [92mb'CONVERGENCE: REL_REDUCTION_
↪OF_F_<=_FACTR*EPSMCH'
2018-09-13 18:45:07,040 - MultiGrid solver - INFO - [92mIterations: 10
2018-09-13 18:45:07,041 - MultiGrid solver - INFO - [92mFunction calls: 32
2018-09-13 18:45:10,977 - AtmoCor - INFO - Finished retrieval and saving them into␣
↪local files.
2018-09-13 18:45:13,309 - AtmoCor - INFO - Set AOI.
2018-09-13 18:45:13,310 - AtmoCor - INFO - Slice TOA bands based on AOI.
2018-09-13 18:45:14,708 - AtmoCor - INFO - Parsing angles.
2018-09-13 18:45:15,728 - AtmoCor - INFO - Parsing auxs.
2018-09-13 18:45:16,584 - AtmoCor - INFO - Parsing atmo parameters.
2018-09-13 18:45:23,338 - AtmoCor - INFO - Loading emus.
2018-09-13 18:45:25,858 - AtmoCor - INFO - Get correction coefficients.
2018-09-13 18:45:32,039 - AtmoCor - INFO - Doing corrections.
2018-09-13 18:45:37,474 - AtmoCor - INFO - Composing RGB.
2018-09-13 18:45:39,423 - AtmoCor - INFO - Done.
```

TOA image	BOA image
	

```
toa = []
boa = []
for i in range(6):
```

(continues on next page)

(continued from previous page)

```
    toa.append(gdal.Open(toa_bands[i]).ReadAsArray(1500, 500, 1,1)[0,0])
    boa.append((gdal.Open(toa_bands[i].replace('.TIF', '_sur.tif')).ReadAsArray(1500,␣
↪500, 1,1)[0,0])/10000.)
plt.plot(band_wv, toa, 's-', label='TOA')
plt.plot(band_wv, np.array(boa), 'o-', label='BOA')
plt.legend()
```

```
<IPython.core.display.Javascript object>
```

```
<matplotlib.legend.Legend at 0x7fade5894080>
```

Here we demonstrate the use of SIAC for the correction of Landsat 5 images, but it can only treated as a test and if one wants to do the real AC of Landsat 5 collection, the emulators and spectral mapping should be created for Landsat 5 TM sensor also a more resonable prior should be used.

CHAPTER FIVE

TECHNICAL DOCUMENTATION

5.1 *SIAC_S2*

`SIAC.SIAC_S2.`**`SIAC_S2`**(*s2_t*, *send_back=False*, *mcd43='/home/docs/MCD43/'*, *vrt_dir='/home/docs/MCD43_VRT/'*, *aoi=None*, *global_dem=None*, *cams_dir=None*, *jasmin=False*)

if not os.path.exists(file_path + '/emus/'): os.mkdir(file_path + '/emus/')

if len(glob(file_path + '/emus/' + 'isotropic_MSI_emulators_*_x?p_S2?.pkl')) < 12: url = 'http://www2.geog.ucl.ac.uk/~ucfafyi/emus/' req = requests.get(url) to_down = [] for line in req.text.split():

if 'MSI' in line: fname = line.split('"')[1].split('<')[0] if 'MSI' in fname:

to_down.append([fname, url])

f = lambda fname_url: downloader(fname_url[0], fname_url[1], file_path + '/emus/') parmap(f, to_down)

5.2 *SIAC_L8*

`SIAC.SIAC_L8.`**`SIAC_L8`**(*l8_dir*, *send_back=False*, *mcd43='/home/docs/MCD43/'*, *vrt_dir='/home/docs/MCD43_VRT/'*, *aoi=None*, *global_dem=None*, *cams_dir=None*, *jasmin=False*)

5.3 *Atmospheric parameters Inversion*

`class` `SIAC.the_aerosol.`**`solve_aerosol`**(*sensor_sat*, *toa_bands*, *band_wv*, *band_index*, *view_angles*, *sun_angles*, *obs_time*, *cloud_mask*, *gamma=10.0*, *a_z_order=1*, *pixel_res=None*, *aoi=None*, *aot_prior=None*, *wv_prior=None*, *o3_prior=None*, *aot_unc=None*, *wv_unc=None*, *o3_unc=None*, *log_file=None*, *ref_scale=0.0001*, *ref_off=0*, *ang_scale=0.01*, *ele_scale=0.001*, *prior_scale=[1.0, 0.1, 46.698, 1.0, 1.0, 1.0]*, *emus_dir='SIAC/emus/'*, *mcd43_dir='~/DATA/Multiply/MCD43/'*, *global_dem='/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/eles/global_dem.vrt'*, *cams_dir='/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/cams/'*, *spec_m_dir='SIAC/spectral_mapping'*, *aero_res=1000*)

Bases: `object`

Prepareing modis data to be able to pass into atmo_cor for the retrieval of atmospheric parameters.

5.4 *Atmospheric parameters Solver*

class SIAC.atmo_solver.**solving_atmo_paras**(*boa, toa, sza, vza, saa, vaa, aot_prior, tcwv_prior, tco3_prior, elevation, aot_unc, tcwv_unc, tco3_unc, boa_unc, Hx, Hy, mask, full_res, aero_res, emulators, band_indexs, band_wavelength, pix_res=10.0, gamma=0.5, alpha=- 1.6, subsample=1, subsample_start=0, log_file=None*)

Bases: object

A simple implementation of dark dense vegitation method for the restieval of prior aot.

5.5 *Atmospheric correction*

class SIAC.the_correction.**atmospheric_correction**(*sensor_sat, toa_bands, band_index, view_angles, sun_angles, aoi=None, aot=None, tcwv=None, tco3=None, aot_unc=None, tcwv_unc=None, tco3_unc=None, cloud_mask=None, obs_time=None, log_file=None, a_z_order=1, ref_scale=0.0001, ref_off=0, ang_scale=0.01, ele_scale=0.001, atmo_scale=[1.0, 1.0, 1.0, 1.0, 1.0, 1.0], global_dem='/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/ele cams_dir='/vsicurl/http://www2.geog.ucl.ac.uk/~ucfafyi/cams/ emus_dir='SIAC/emus/', cams_scale=[1.0, 0.1, 46.698, 1.0, 1.0, 1.0], block_size=600, rgb=[None, None, None]*)

Bases: object

A class doing the atmospheric coprrection with the input of TOA reflectance angles, elevation and emulators of 6S from TOA to surface reflectance.

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search