grs Documentation

Release 1

Tristan Harmel

Nov 06, 2019

CONTENTS:

CHAPTER

ONE

DOCUMENTATION FOR THE GRS ALGORITHM.

1.1 GRS (Glint Removal for Sentinel-2-like sensors)

The GRS processing was designed to correct Sentinel-2-like satellite images for atmospheric effects and Sun reflection (sunglint) above water surfaces (e.g., ocean, coastal, estuary, river and lake waters). Theoretical background can be found in [Harmel et al., 2018].

The overall structure of the algorithm can be summarized by the following flowchart:

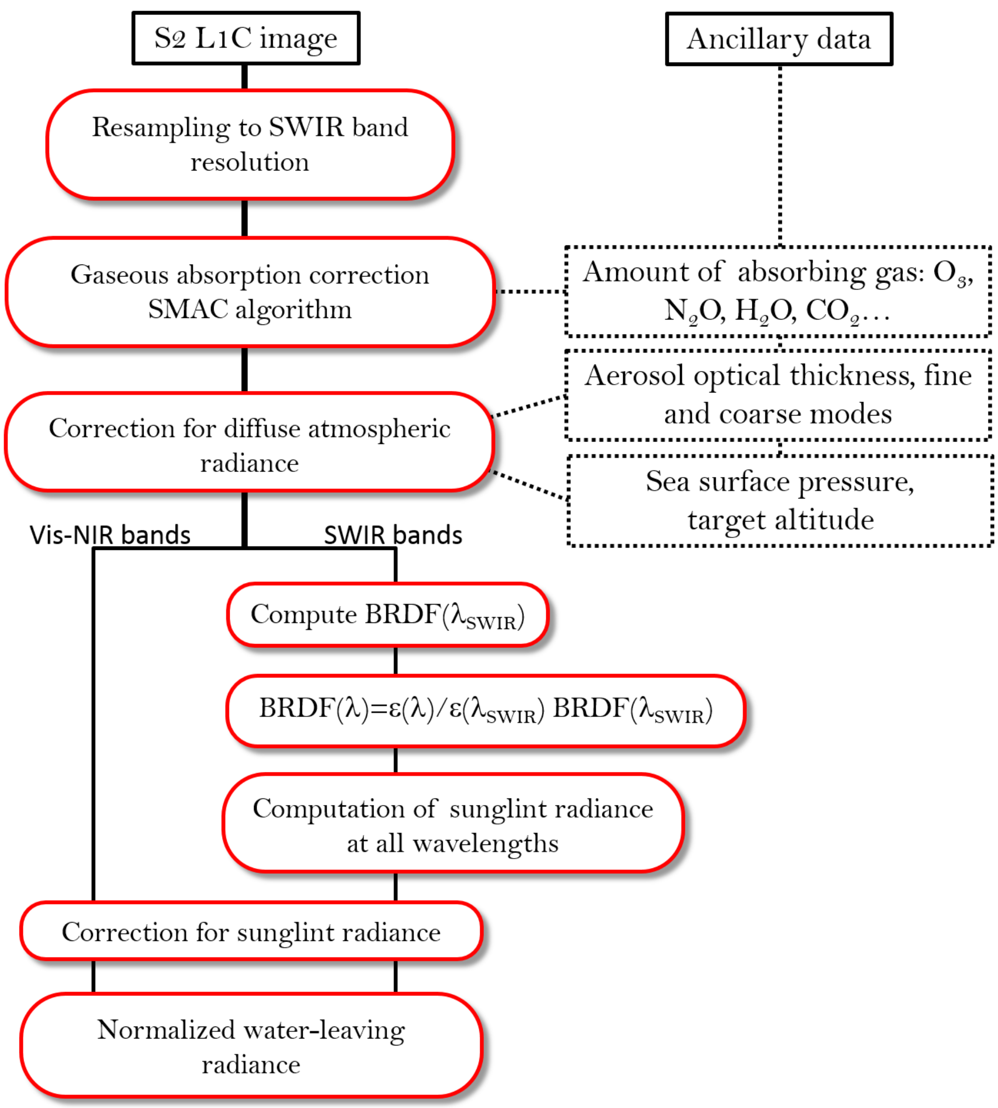

Fig. 1: Flowchart of the proposed GRS algorithm to remove the sunglint contribution from the MSI/Sentinel-2 images.

1.2 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. See deployment for notes on how to deploy the project on a live system.

1.2.1 Prerequisites

Register and ask for a key to use ECMWF API

Download and install the SNAP software. Configure the SNAP-python interface and link the obtained `snappy` folder to your python site-packages as `esasnappy`. For example:

```
ln -s /FULL_PATH/.snap/snap-python/snappy /PATH_TO_LIB_PYTHON/lib/python3.6/site-
↪packages/esasnappy
```

Compilers such gcc and gfortran are needed to install the package.

Compile all C and fortran files into shared libraries:

```
make
```

Generate the *config.py* file:

- In the ./grs/grs folder, copy *config_local.py* to *config.py*.
- Then, edit *config.py* according to your folders tree and path to your grs installation folder.

1.2.2 Installing

To install the package:

```
python setup.py install
```

or

```
python setup.py install --user
```

If the installation is successful, you should have:

```
$ grs
Usage:
  grs <input_file> [--sensor <sensor>] [-o <ofile>] [--odir <odir>] [--shape <shp>] [-
↪-wkt <wktfile>]   [--longlat <longmax,longmin,latmax,latmin> ]    [--altitude=alt]␣
↪[--dem] [--aerosol=DB] [--aeronet=<afile>]    [--aot550=aot] [--angstrom=ang] [--
↪output param]   [--resolution=res] [--levname <lev>] [--no_clobber] [--memory_safe]␣
↪[--unzip]
  grs -h | --help
  grs -v | --version
```

1.2.3 Running the tests

From terminal:

```
grs test/data/S2B_MSIL1C_20180927T103019_N0206_R108_T31TGK_20180927T143835.SAFE --
↪shape test/data/shape/SPO04.shp --odir test/results/ --aerosol cams_forecast --dem -
↪-resolution 20
```

You should get something like:

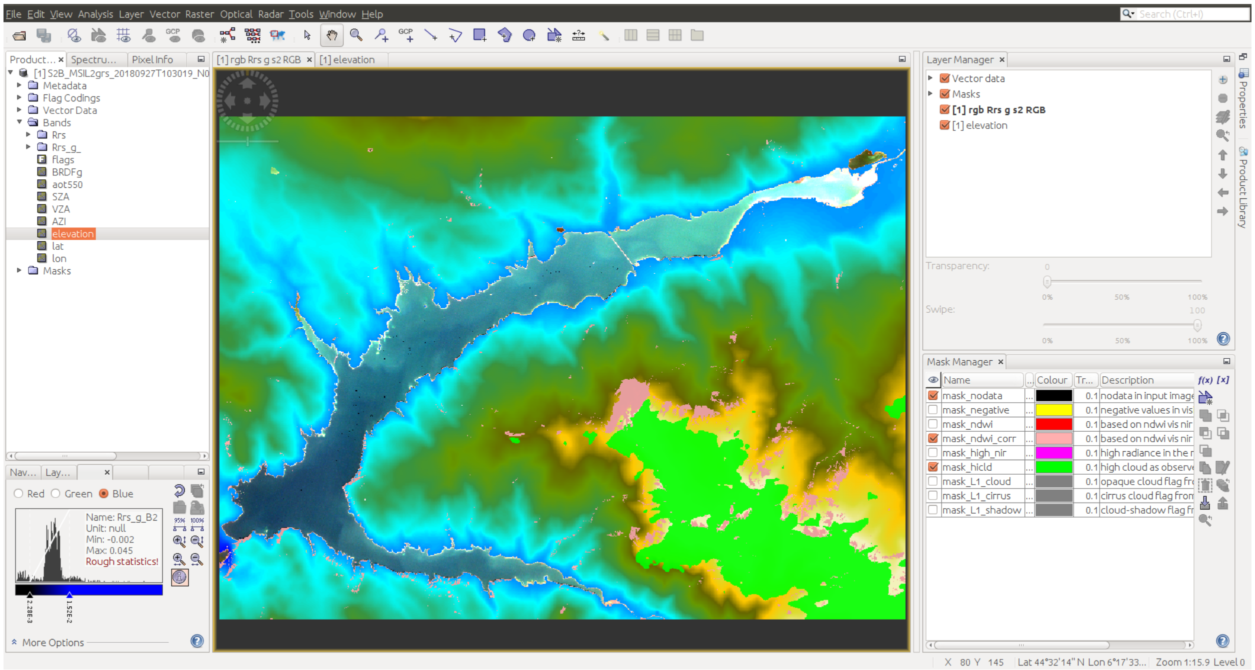

Fig. 2: Example of an output image visualized in the SNAP software.

Another examples of output images before (1st column) and after (2nd column) sunglint correction:

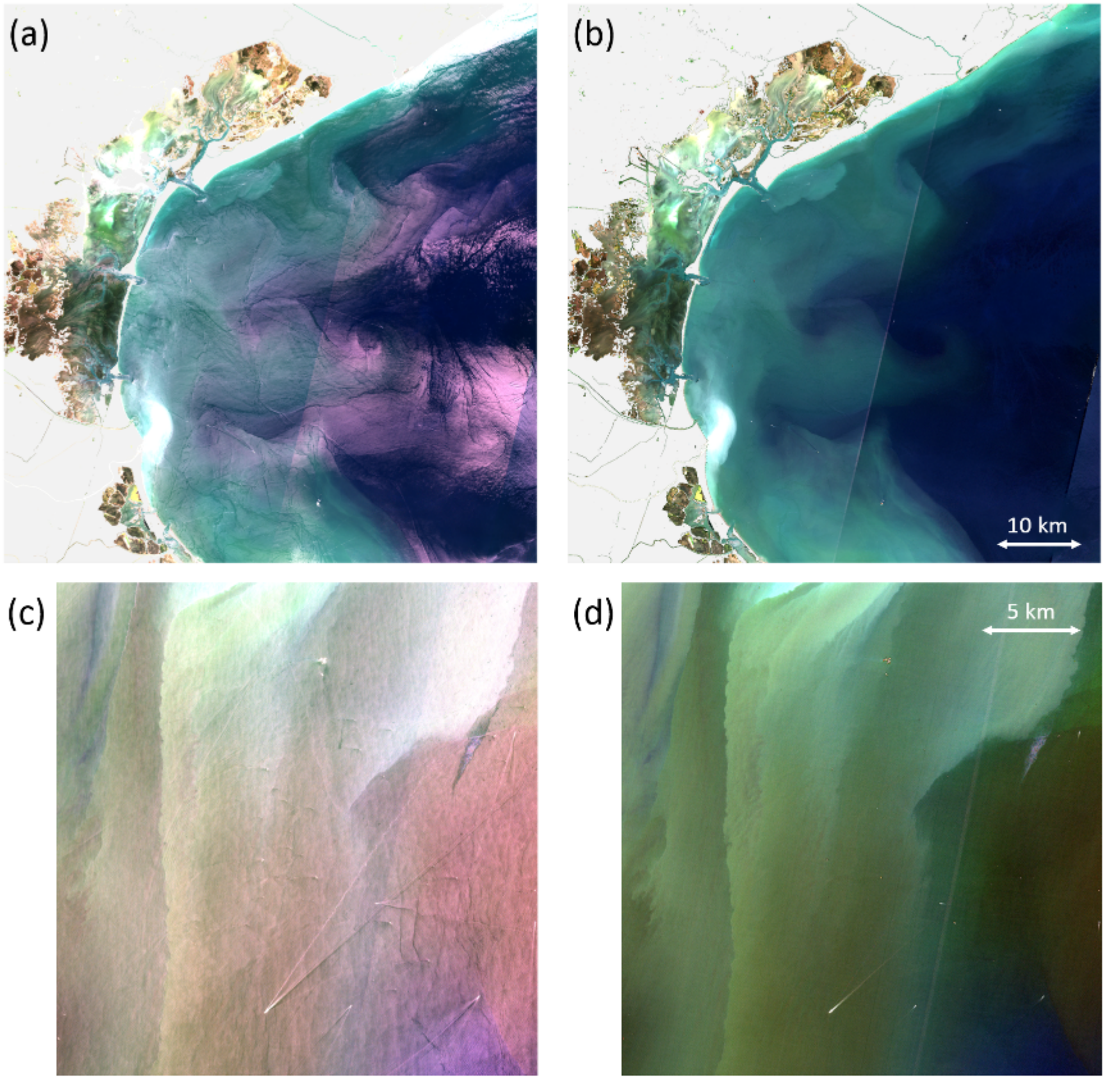

Fig. 3: RGB images obtained after subtraction of the atmospheric radiance from TOA signal but before (left column) and after (right column) removing the sunglint radiance. These images correspond to the areas surrounding the AERONET-OC sites of (a, b) Venice (July 18, 2016) and (c, d) WaveCIS (April 23, 2016). Note that the same color scale was used to generate the RGB images before and after removing the sunglint radiance.

1.2.4 Deployment

See examples in exe.

1.3 Package development

1.3.1 Contributing

Please contact authors for details on our code of conduct, and the process for submitting pull requests to us.

1.3.2 Authors

- **Tristan Harmel** - *Initial work* - contact

See also the list of [contributors](...) who participated in this project.

1.3.3 License

This project is licensed under the MIT License - see the `LICENSE.md` file for details.

1.3.4 Acknowledgments

- The Step forum and Marco Peters are acknowledged for their useful help to process Sentinel-2 data with the snappy API.
- The authors are very grateful to Olivier Hagolle for providing open source codes to perform gaseous absorption correction and massive Sentinel-2 data download.

CHAPTER

TWO

PROJECT MODULES

Version history:

1.0.0 grs for Sentinel2

1.1.0 adaptation to Landsat 4, 5, 7, 8

1.1.1

1.1.2 output in Rrs unit

1.1.3

1.1.4: set cloud mask; (compressed) netcdf4 output format

1.2.0: load full data matrix from image instead of line by line pixel extraction (preparation for multipixel retrieval algorithm

`acutils`	Atmospheric Correction utilities to manage LUT and atmosphere parameters (aerosols, gases)
`anglegen`	
`auxdata`	
`config`	Set absolute path
`grs_process`	Main program
`mask`	Setting custom masks
`run`	Executable to process L1C images from Sentinel-2 and Landsat mission series
`s2_angle`	Optional code to compute viewing configuration for S2
`smac`	Correction for gaseous absorption based on SMAC method (Rahman and Dedieu, 1994)
`utils`	Defines main python objects and image manipulation functions (linked to the ESA snappy library)

2.1 grs.acutils

Atmospheric Correction utilities to manage LUT and atmosphere parameters (aerosols, gases)

Classes

`aerosol`()	aerosol parameters and parameterizations

Continued on next page

Table 2 – continued from previous page

lut(band)	Load LUT FROM RT COMPUTATION (OSOAA_h)
misc	Miscelaneous utilities
smac(smac_bands, smac_dir)	Gaseous absorption and transmission from pre-calculated 6S/SMAC data

class grs.acutils.**lut**(*band*)

Bases: object

Load LUT FROM RT COMPUTATION (OSOAA_h)

load_lut(*lut_file, ind_wl, aot=[0.01, 0.05, 0.1, 0.3, 0.5, 0.8], vza_max=20, reflectance=True*)

load lut calculated from OSOAA code

Arguments:

- `lut_file` – netcdf file where lut data are stored for a given aerosol model
- `ind_wl` – indices of the desired central wavelength in `lut_file`
- `aot` – aerosol optical thickness at 550 nm for which lut are loaded
- `vza_max` – load all lut data for vza <= vza_max
- `reflectance` – if true: return data in reflectance unit, return normalized radiane otherwise

Construct object with:

- `aot` – aerosol optical thickness at 550 nm for which lut are loaded
- `Cext` – aerosol extinction coefficient (spectral)
- `Cext550` – aerosol extinction coefficient at 550 nm
- `vza` – viewing zenith angle (in deg)
- `sza` – solar zenith angle (in deg)
- `azi` – relative azimuth between sun and sensor (in opposition when azi = 0)
- `wl` – central wavelength of the sensor bands
- **`refl` – Top-of-atmosphere reflectance (or normalized radiance if reflectance == False);** array of dims: [wl, sza, azi, vza]

interp_lut(*points*, *values*, *x*)

expected x dims: [[sza1, azi1, vza1],[sza2, azi2, vza2]…]

class grs.acutils.**aerosol**

Bases: object

aerosol parameters and parameterizations

func(*wl*, *a*, *b*, *c*)

function for spectral variation of AOT

fit_spectral_aot(*wl*, *aot*)

call to get fitting results on AOT data

get_spectral_aot(*wl*)

set aot for a given set of wavelengths

func_aero(*Cext*, *fcoef*)

function to fit spectral behavior of bimodal aerosols onto aeronet optical thickness

fit_aero(*nCext_f*, *nCext_c*, *naot*)
Call to get fine mode coefficient based on fitting on AOT data.

Arguments:

- `nCext_f` – Normalized extinction coefficient of the fine mode aerosols
- `nCext_c` – Normalized extinction coefficient of the coarse mode aerosols
- `naot` – Normalized spectral aerosol optical thickness

Return values: The mixing ratio of the fine mode aerosol

Notes: .

class grs.acutils.**smac**(*smac_bands*, *smac_dir*)
Bases: `object`

Gaseous absorption and transmission from pre-calculated 6S/SMAC data

set_values(*o3du=300*, *h2o=0*, *no2=0*)
set atmospheric concentration values: :param o3du: ozone in DU :param h2o: water vapor in g/cm^2 :param no2: nitrous dioxide in . . .

set_gas_param()
Load gaseous absorption parameters as computed for SMAC from 6S RT code

compute_gas_trans(*iband*, *pressure*, *mu0*, *muv*)
Compute gaseous transmittances (up and down) from SMAC parameters and ancillary data pressure : actual pressure in hPA mu0 : cosine of solar zenith angle muv : cosine of viewing zenith angle

class grs.acutils.**misc**
Bases: `object`

Miscelaneous utilities

static get_pressure(*alt*, *psl*)
Compute the pressure for a given altitude alt : altitude in meters (float or np.array) psl : pressure at sea level in hPa palt : pressure at the given altitude in hPa

2.2 grs.anglegen

Classes

`angle_generator`

2.3 grs.auxdata

Classes

Aeronet	Contains functions for importing AERONET measurements.
cams()	Unit Conversion PWC (Precipitable Water Content), Grib Unit [kg/m^2] MSL (Mean Sea Level pressure), Grib Unit [Pa] OZO (Ozone), Grib Unit [kg/m^2]

Continued on next page

Table 4 – continued from previous page

sensordata(sensor)	Dictionnaries of the auxilliary data, functions to be applied for the sensors data to process.

class grs.auxdata.**sensordata**(*sensor*)

Bases: object

Dictionnaries of the auxilliary data, functions to be applied for the sensors data to process.

Arguments:

- **sensor – Name of the sensor to process:**
  - LANDSAT_4
  - LANDSAT_5
  - LANDSAT_7
  - LANDSAT_8
  - S2A
  - S2B
- indband – indexes of the requested bands (bands that will be processed)

Construct object with values:

- rot – Rayleigh optical thickness for each band
- band names for angles files format
- smac_band – band names for SMAC files format
- rg – Cox-Munk Fresnel reflection factor ratio (R(wl)/Rref(2190nm) [Harmel et al., 2018]
- angle_generator – function to compute angles from metadata

class grs.auxdata.**cams**

Bases: object

Unit Conversion PWC (Precipitable Water Content), Grib Unit [kg/m^2] MSL (Mean Sea Level pressure), Grib Unit [Pa] OZO (Ozone), Grib Unit [kg/m^2]

GRIB_UNIT = [kg/m^2] standard ozone column is 300 DU (Dobson Units), equals to an air column of 3 mm at STP (standard temperature (0 degree C) and pressure of 1013 mbar).

Thus, molecular weight of O3 (M = 48): 2.24 g (equals to 22.4 liter at STP)

300 DU = 3 mm (equals to (0.3*48 / 2.24) [g/m^2]) = 6.428 [g/m^2] = 6.428 E-3 [kg/m^2]

Example:

ozone (kg/m^2 = 300 /6.428E-3 DU) pressure (Pa = 1/100 hPa) water vapor (kg/m^2 = 10^3/10^4 = 0.1 g/cm^2)

get_tile_dir(*file*)

file : absolute path of S2 directory set full path of tile

get_aux_dir()

Returns

get_ecmwf_file(*product*)

Parameters product –

Returns

get_ecmwf_data()

load_cams_data(*target*, *date*, *grid='0.125/0.125'*, *param='125.210/137.128/151.128/165.128/166.128/167.128/206.128/207*
data_type='cams_reanalysis')

generate aerosol data from cams of ECMWF subset on the image grid (POLYGON wkt) reproject on the image coordinate reference system (crs)

get_cams_aerosol(*target, date, wkt, param=['aod469', 'aod550', 'aod670', 'aod865', 'aod1240']*)

Nearest neighbor in time :param target: :param date: :param wkt: :return:

get_cams_ancillary(*target, date, wkt, param=['msl', 'tco3', 'tcwv', 'tcno2', 't2m']*)

Nearest neighbor in time :param target: :param date: :param wkt: :return:

get_cams_aerosol_old(*target*, *date*, *wkt*, *crs*, *data_type*)

generate aerosol data from cams of ECMWF subset on the image grid (POLYGON wkt) reproject on the image coordinate reference system (crs)

download_erainterim(*target*, *date*, *time='00:00:00'*, *grid='0.125/0.125'*, *param='137.128/151.128/206.210/207.210/213.210/214.210/215.210/216.210'*, *area=None*, *data_type=''*)

This function open a connexion through an existing CAMS/ERAIterim account and download the requested data.

Arguments:

- `target` – path name for output file
- `time` – time of forecast or reanalysis
- `date` – acquisition date (ex: '2015-01-31')
- `grid` – resolution in degrees (ex: '0.125/0.125')
- `param` – ID numbers for the requsted data
- `area` – option to restrict area (for global dataset None or '90/-180/-90/180')

Notes:

- `server` – connexion au server via API ECMWF
- `step` – forecast time hours

class grs.auxdata.**Aeronet**

Bases: `object`

Contains functions for importing AERONET measurements. Modified from:

Copyright 2012 Robin Wilson and contributors listed in the CONTRIBUTORS file.

pandas = <module 'pandas' from '/home/harmel/.local/lib/python3.6/site-packages/pandas/

classmethod import_aeronet_data(*data*, *filename*, *time*)

Imports data from an AERONET data file to a given `aeronet` object.

Arguments:

- `data` – Aeronet object to store the paramaters of interest (e.g., wavelenghts, aot, ozone)
- `filename` – The filename of the AERONET file described above

- `time` – The date and time of the simulation you want to run, used to choose the AERONET data which is closest in time. Provide this as a string in almost any format, and Python will interpret it. For example, `'12/03/2010 15:39'`. When dates are ambiguous, the parsing routine will favour DD/MM/YY rather than MM/DD/YY.

Return value: The function will return `s` with the `aero_profile` and `aot550` fields filled in from the AERONET data.

Notes: Beware, this function returns `data` from `filename` for the closest measurement time in the limit of plus or minus 2 days from `time`.

2.4 grs.config

Set grs absolute path and global variables

2.5 grs.grs_process

Main program

Classes

process()	

class grs.grs_process.**process**

Bases: `object`

execute(*file*, *outfile*, *wkt*, *sensor=None*, *altitude=0*, *aerosol='cams_forecast'*, *ancillary=None*, *dem=True*, *aeronet_file=None*, *aot550=0.1*, *angstrom=1*, *resolution=None*, *unzip=False*, *untar=False*, *startrow=0*, *memory_safe=False*, *angleonly=False*, *grs_a=False*, *output='Rrs'*)

Main program calling all GRS steps

Parameters

- **file** – Input file to be processed
- **sensor** – Set the sensor type: S2A, S2B, LANDSAT_5, LANDSAT_7, LANDSAT_8 (by default sensor type is retrieved from input file name)
- **wkt** – Well-Known-Text format defining the area of interest for which the image is subsetted
- **outfile** – Absolute path of the output file
- **dem** – if True digital elevation model is applied for per-pixel pressure calculation (data from SNAP/SRTM)
- **altitude** – provide altitude if *dem* is set as *False*
- **aeronet_file** – optional aeronet file to be used for aerosol calculations
- **resolution** – pixel resolution in meters (integer)
- **unzip** – if True input file is unzipped before processing, NB: unzipped files are removed at the end of the process

- **startrow** – row number of the resampled and subset image on which the process starts, recommended value 0 NB: this option is used to in the context of operational processing of massive dataset
- **angleonly** – if true, grs is used to compute angle parameters only (no atmo correction is applied)
- **output** – set the unit of the retrievals:
  - ’Lwn’, normalized water-leaving radiance (in $mWcm^{-2}sr^{-1}\mu m^{-1}$)
  - ’Rrs’, remote sensing reflectance (in sr^{-1})

  {default: ‘Rrs’}
- **grs_a** – switch to grs-a algorithm (Lwn and aerosol) if True

Returns

2.6 grs.mask

Setting custom masks

Classes

mask()	Class for coding supplementary masks

class grs.mask.**mask**

Bases: object

Class for coding supplementary masks

water_detection()

Very simple water detector based on Otsu thresholding method of NDWI.

2.7 grs.run

Executable to process L1C images from Sentinel-2 and Landsat mission series

Usage: grs <input_file> [–grs_a] [–sensor <sensor>] [-o <ofile>] [–odir <odir>] [–shape <shp>] [–wkt <wktfile>] [–longlat <longmax,longmin,latmax,latmin>] [–altitude=alt] [–dem] [–aerosol=DB] [–aeronet=<afile>] [–aot550=aot] [–angstrom=ang] [–output param] [–resolution=res] [–levname <lev>] [–no_clobber] [–memory_safe] [–unzip] grs -h | –help grs -v | –version

Options: -h –help Show this screen. -v –version Show version.

<input_file> Input file to be processed –grs_a Apply the advanced (beta) version of GRS if set –sensor sensor Set the sensor type: S2A, S2B, LANDSAT_5, LANDSAT_7, LANDSAT_8

(by default sensor type is retrieved from input file name)

-o ofile	Full (absolute or relative) path to output L2 image.
--odir odir	Ouput directory [default: ./]
--levname lev	Level naming used for output product [default: L2grs]

--no_clobber	Do not process <input_file> if <output_file> already exists.
--shape shp	Process only data inside the given shape
--wkt wktfile	Process only data inside the given wkt file
--longlat <longmax,longmin,latmax,latmin>	Restrict ROI to long max, long min, lat max, lat min in decimal degree [default: 180, -180, 90, -90]
--altitude=alt	altitude of the scene to be processed in meters [default: 0]
--dem	Use SRTM digital elevation model instead of –altitude (need internet connection)
--aerosol=DB	aerosol data base to use within the processing DB: cams_forecast, cams_reanalysis, aeronet, user_model [default: cams_reanalysis]
--aeronet=afile	if *–aerosol* set to 'aeronet', provide aeronet file to use
--aot550=aot	if *–aerosol* set to 'user_model', provide aot550 value to be used [default: 0.1]
--angstrom=ang	if *–aerosol* set to 'user_model', provide aot550 value to be used [default: 1]
--output param	set output unit: 'Rrs' or 'Lwn' [default: Rrs]

–resolution=res spatial resolution of the scene pixels –unzip to process zipped images seamlessly –memory_safe use generic resampler instead of S2resampler to save memory

(induces loss in angle resolution per pixel for S2)

Functions

`main()`	
`shp2wkt(shapefile)`	

2.8 grs.s2_angle

Optional code to compute viewing configuration for S2

Classes

s2angle	Calculations of the viewing geometries for each pixel and band [kept for records; those calculations are now done through the ESA snappy library]

class grs.s2_angle.**s2angle**

Bases: object

Calculations of the viewing geometries for each pixel and band [kept for records; those calculations are now done through the ESA snappy library]

2.9 grs.smac

Correction for gaseous absorption based on SMAC method (Rahman and Dedieu, 1994)

Functions

`PdeZ`(Z)	PdeZ : Atmospheric pressure (in hpa) as a function of altitude (in meters)
`smac_dir`(r_surf, tetas, phis, tetav, phiv, . . .)	r_toa=smac_dir (r_surf, tetas, phis, tetav, phiv,pressure,taup550, uo3, uh2o, coef) Application des effets atmosphÃ©riques
`smac_inv`(r_toa, tetas, phis, tetav, phiv, . . .)	r_surf=smac_inv(r_toa, tetas, phis, tetav, phiv,pressure,taup550, uo3, uh2o, coef) Corrections atmosphÃ©riques

Classes

`coeff`(smac_filename)	library for atmospheric correction using SMAC method (Rahman and Dedieu, 1994) Contains : smac_inv : inverse smac model for atmospheric correction TOA==>Surface smac dir : direct smac model Surface==>TOA coefs : reads smac coefficients PdeZ : # PdeZ : Atmospheric pressure (in hpa) as a function of altitude (in meters)

`grs.smac.`**`PdeZ`**(*Z*)

PdeZ : Atmospheric pressure (in hpa) as a function of altitude (in meters)

class `grs.smac.`**`coeff`**(*smac_filename*)

Bases: `object`

library for atmospheric correction using SMAC method (Rahman and Dedieu, 1994) Contains :

smac_inv [inverse smac model for atmospheric correction] TOA==>Surface

smac dir [direct smac model] Surface==>TOA

coefs : reads smac coefficients PdeZ : # PdeZ : Atmospheric pressure (in hpa) as a function of altitude (in meters)

`grs.smac.`**`smac_inv`**(*r_toa*, *tetas*, *phis*, *tetav*, *phiv*, *pressure*, *taup550*, *uo3*, *uh2o*, *coef*)

r_surf=smac_inv(r_toa, tetas, phis, tetav, phiv,pressure,taup550, uo3, uh2o, coef) Corrections atmosphÃ©riques

`grs.smac.`**`smac_dir`**(*r_surf*, *tetas*, *phis*, *tetav*, *phiv*, *pressure*, *taup550*, *uo3*, *uh2o*, *coef*)

r_toa=smac_dir (r_surf, tetas, phis, tetav, phiv,pressure,taup550, uo3, uh2o, coef) Application des effets atmosphÃ©riques

2.10 grs.utils

Defines main python objects and image manipulation functions (linked to the ESA snappy library)

Classes

`info`(product, sensordata[, aerosol, ...])	**param product**
`utils`()	utils for arrays, images manipulations

class grs.utils.**info**(*product*, *sensordata*, *aerosol='cams_forecast'*, *ancillary='cams_forecast'*, *output='Rrs'*)

Bases: `object`

Parameters

- **product** –
- **sensordata** –
- **aerosol** –
- **ancillary** –
- **output** – set the unit of the retrievals: * 'Lwn', normalized water-leaving radiance (in mW cm-2 sr-1 mum-1) * 'Rrs', remote sensing reflectance (in sr-1) {default: 'Rrs']

set_outfile(*file*)

Parameters **file** –

Returns

set_aeronetfile(*file*)

Parameters **file** –

Returns

get_product_info()

Returns

get_bands(*band_names=None*)

get wavelengths, bands, geometries :param band_names: :return:

get_flag(*flag_name*)

get binary flag raster of row *rownum*

get_elevation()

load elevation data into numpy array

load_data()

load ta from input (subsetted) satellite image :return:

load_flags()

Set flags from L1 data (must be called after *load_data* :return:

unload_data()

unload data (not efficient due to ESA snappy issue with java jvm

create_product()

Create output product dimensions, variables, attributes, flags.... :return:

checksum(*info*)

Save info on the current processing stage :param info: :return:

finalize_product()
remove checksum file remove extension ".incomplete" from output file name convert into netcdf (compressed) from gpt and ncdump/nco tool

print_info()
print info, can be used to check if object is complete

class grs.utils.**utils**
Bases: object

utils for arrays, images manipulations

static init_arrayofarrays(*number_of_array*, *dim*)
Initialize Nd array of Nd dimensions (given by dim)

example: arr1, arr2 = init_array(2,(2,3,10))

gives arr1.shape –> (2,3,10) arr2.shape –> (2,3,10)

Parameters

- **number_of_array** – int
- **dim** – list

Returns *number_of_array* numpy arrays of shape *dim*

static init_fortran_array(*number_of_array*, *dim*, *dtype=<class 'numpy.float32'>*)
Initialize Nd array of Nd dimensions (given by dim) in FORTRAN memory order (i.e., compliant with JAVA order provided by snappy)

example: arr1, arr2 = init_array(2,(2,3,10))

gives arr1.shape –> (2,3,10) arr2.shape –> (2,3,10)

Parameters

- **number_of_array** – int
- **dim** – list
- **dtype** – numpy type for arrays

Returns *number_of_array* numpy arrays of shape *dim*

generic_resampler(*s2_product*, *resolution=20*, *method='Bilinear'*)
method: Nearest, Bilinear

static resampler(*product*, *resolution=20*, *upmethod='Bilinear'*, *downmethod='First'*, *flag='FlagMedianAnd'*, *opt=True*)
Resampling operator dedicated to Sentinel2-msi characteristics (e.g., viewing angles)

Parameters

- **product** – S2-msi product as provided by esasnappy ProductIO.readProduct()
- **resolution** – target resolution in meters
- **upmethod** – interpolation method ('Nearest', 'Bilinear', 'Bicubic')
- **downmethod** – aggregation method ('First', 'Min', 'Max', 'Mean', 'Median')
- **flag** – method for flags aggregation ('First', 'FlagAnd', 'FlagOr', 'FlagMedianAnd', 'FlagMedianOr')
- **opt** – resample on pyramid levels (True/False)

Returns interpolated target product

static s2_resampler(*product*, *resolution=20*, *upmethod='Bilinear'*, *downmethod='First'*, *flag='FlagMedianAnd'*, *opt=True*)
Resampling operator dedicated to Sentinel2-msi characteristics (e.g., viewing angles)

Parameters

- **product** – S2-msi product as provided by esasnappy ProductIO.readProduct()
- **resolution** – target resolution in meters (10, 20, 60)
- **upmethod** – interpolation method ('Nearest', 'Bilinear', 'Bicubic')
- **downmethod** – aggregation method ('First', 'Min', 'Max', 'Mean', 'Median')
- **flag** – method for flags aggregation ('First', 'FlagAnd', 'FlagOr', 'FlagMedianAnd', 'FlagMedianOr')
- **opt** – resample on pyramid levels (True/False)

Returns interpolated target product

get_subset(*product*, *wkt*)
subset from wkt POLYGON

get_extent(*product*)
Get corner coordinates of the ESA SNAP product(getextent)

int step - the step given in pixels

getReprojected(*product*, *crs='EPSG:4326'*, *method='Bilinear'*)
Reproject a esasnappy product on a given coordinate reference system (crs)

get_sensor(*file*)
Get sensor type from file name :param file: file in standard naming :return: sensor type

CHAPTER

THREE

FORTRAN MODULES

Use MINPACK and INTERP package

subroutine main_algo(*npix*, *nband*, *naot*, *vza*, *sza*, *azi*, *rtoa*, *mask*, *wl*, *pressure_corr*, *aotlut*, *rlut_f*, *rlut_c*, *cext_f*, *cext_c*, *rg_ratio*, *f0*, *rot*, *aot*, *aot550*, *fine_coef*, *nodata*, *rrs*, *rcorr*, *rcorrg*, *aot550_est*, *brdf_est*)

f2py -c -m main_algo main_algo.f95

Parameters

- **npix** *[integer,in]*
- **nband** *[integer,in]*
- **naot** *[integer,in]*
- **vza** (nband,npix) *[real,in]*
- **sza** (npix) *[real,in]*
- **azi** (nband,npix) *[real,in]*
- **rtoa** (nband,npix) *[real,in]*
- **mask** (npix) *[integer,in]*
- **wl** (nband) *[real,in]*
- **pressure_corr** (npix) *[real,in]*
- **aotlut** (naot) *[real,in]*
- **rlut_f** (naot,nband,npix) *[real,in]*
- **rlut_c** (naot,nband,npix) *[real,in]*
- **cext_f** (nband) *[real,in]*
- **cext_c** (nband) *[real,in]*
- **rg_ratio** (nband) *[real,in]*
- **f0** (nband) *[real,in]*
- **rot** (nband) *[real,in]*
- **aot** (nband) *[real,inout/in]*
- **aot550** (npix) *[real,inout/in]*
- **fine_coef** *[real,in]*
- **nodata** *[real,inout]*
- **rrs** *[logical,in]*

- **rcorr** (nband,npix) *[real,out]*
- **rcorrg** (nband,npix) *[real,out]*
- **aot550_est** (npix) *[real,out]*
- **brdf_est** (npix) *[real,out]*

#.. f:autosrcfile:: main_algo.f95

CHAPTER

FOUR

DEPLOYMENT MODULES

grs_from_list	
procutils	utils module dedicated to processing of massive dataset

4.1 exe.procutils

utils module dedicated to processing of massive dataset

Classes

misc()	Miscellaneous utilities
multi_process()	Utilities for multicore processing

class exe.procutils.**misc**

Bases: object

Miscellaneous utilities

wktbox(*center_lon*, *center_lat*, *width=1*, *height=1*)

Parameters

- **center_lon** – decimal longitude
- **center_lat** – decimal latitude
- **width** – width of the box in km
- **height** – height of the box in km

Returns wkt of the box centered on provided coordinates

get_sensor(*file*)

Get sensor type from file name :param file: file in standard naming :return: sensor type

get_tile(*file*)

Get tile from file name :param file: file in standard naming :return: tile ID

set_ofile(*file*, *odir=''*, *level_name='l2grs'*, *suffix=''*)

get satellite type andset output file name

chunk(*it*, *n*)

return a tuple of n items found in it :param it: :param n: :return:

class exe.procutils.**multi_process**

Bases: object

Utilities for multicore processing

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX

M

P

R

S

U

W