pygaarst Documentation

Release 0.1pre-alphaRC1

Chris Waigl

March 04, 2015

Contents

Pygaarst is a Python package that is designed to make it easy to access geospatial raster data such as remote sensing imagery, and perform frequently needed processing steps in a human-friendly way.

Loading data, accessing the intended band or dataset, converting it to meaningful units and calculating standardised indices should be easy. As should be plotting the imagery on a map or combining raster with vector datasets, such as ESRI shapefiles.

Some examples:

```
>>> from pygaarst import raster
>>> basedir = "path/to/data"
>>> landsatscene = "LE70690142004201EDC01"
>>> sc = raster.Landsatscene(os.path.join(basedir, landsatscene))
>>> swirband = sc.band7
>>> type(swirband.data)
<type 'numpy.ndarray'>
    >>> type(swirband.radiance)
<type 'numpy.ndarray'>
>>> swirband.radiance.shape
(8021, 8501)
    >>> ndvi = sc.ndvi
```

As of the moment this documentation is written, pygaarst is under development and not all of the API design should be considered stable. The following capabilities are supported:

- Python 2 (2.6+)
- `GeoTiff` base class (in `pygaarst.raster`), with `Landsatband`, `ALIband` and `Hyperionband` inheriting from it
- `Landsatscene`, `Hyperionscene` and `ALIscene` to represent a scene directory as retrieved from the USGS data portal
- `VIIRSHDF5` for NPP VIIRS SDS dataset files in HDF5 format, as retrieved from NOAA's portals
- Reading georeference files or metadata and calculation of pixel-center and, for gridded data, pixel-corner coordinate references
- A metadata parser for USGS-style MTL files in `pygaarst.mtlutils`
- Multiple helper methods, functions and properties for frequently repeated tasks: transformation to at-sensor radiance and reflectance, NDVI and NBR (for Landsat) as well as generic normalized difference indices, radiant temperature from thermal infrared bands, calculation and export of radiance spectra for Hyperion, LTK cloud masking algorithm for Landsat...
- A client for the NASA's modaps data download API in `pygaarst.modapsclient`

The following capabilities are planned, roughly in order of priority:

- HDF4 (EOS) swath datasets (MODIS and ASTER Level 1B)
- Basic geometric and statistical operations involving raster and vector structures such as overlays, find-nearest, is-in...
- Python 3 support
- MODIS gridded data products

A step-by-step example of using `pygaarst.raster` to access and plot VIIRS and Landsat data is worked through in this IPython Notebook.

Release 0.1

Date March 04, 2015

Contents:

CHAPTER 1

Getting started

1.1 Prerequisites

As of now, Pygaarst requires Python 2.7. Python 3 (probably >= 3.3) is on the roadmap.

In case you only need the Landsat/Hyperion/ALI metadata parsing features or the client for the NASA MODAPS API there are no dependencies on external libraries.

The following Python packages are required for using any raster manipulation functionality. This includes having the corresponding binary libraries installed:

- numpy
- pyproj
- GDAL

The following Python packages are prerequisites for optional functionality:

- h5py (for reading and processing HDF5 files such as VIIRS scenes)
- python-hdf4 (for reading and processing HDF4 files such as MODIS scenes – but in its absence, GDAL will be tried)
- matplotlib (for plotting)
- mpl_toolkits.basemap (for plotting on a map)
- pytest (for unit tests)
- fiona (for reading and processing GIS vector files)
- shapely (for operations on vector data)

Future functionality is expected to require the following packages:

- netCDF4 (for reading and processing NetCDF files)

Please note that installing the prerequisites may, depending on your configuration, require some planning:

- Installing **Numpy** via `pip` will require a C and a Fortran 77 compiler to build the extension modules. A good alternative is to use a binary distribution. The same is true for Matplotlib.
- **GDAL**, **pyproj**, **h5py**, **python-hdf4**, **netCDF4** and **GEOS** (which is a prerequisite for shapely and the Basemap toolkit) require libraries to be installed before the Python bindings can be installed. The easiest way to do this in a consistent manner is often to use a package manager on GNU/Linux or OS X. For Windows, binary packages are available.

- The **Basemap** toolkit for Matplotlib is a very large install. For Windows, a binary package is available. For GNU/Linux and OS X, it needs to be built from source. If you don't need to plot on maps, you don't need it!
- In many cases, it may be easiest to install a complete scientific Python

software package such as the Enthought Python Distribution or Anaconda, or to use a package manager (on GNU/Linux or OS X) There is one exception to this: Pygaarst can use GDAL to access HDF-EOS (HDF4) files such as MODIS and ASTER L1B data and georeference files. GDAL is, however, not by default compiled with HDF4 support. On OS X, it is rather difficult to install a GDAL with HDF4 support via either Homebrew or the Anaconda Python distribution. The frameworks provided by kyngchaos, while otherwise a sub-optimal way to provide the necessary environment, however, have the required support.

To account for individual user preferences, dependencies are listed in `setup.py` as optional (using the `extras_required` key) for automatic install. The `requirements.txt` file lists them, however, and can be used to install them via `pip`. You may edit `requirements.txt` to comment out the optional dependencies you **know** you won't need.

My personal installation flow, which works well on OS X is to:

- first install all the system libraries with the correct support options
- then install the Python packages via `pip`

Pygaarst is distributed under the terms of the MIT License.

1.2 Installation

The use of a virtualenv is recommended.

At the current stage, there is no stable release for Pygaarst yet. The latest code from the master branch can be installed via:

```
$ pip install git+https://github.com/chryss/pygaarst.git
```

1.3 Usage example

A step-by-step example of using `pygaarst.raster` to access and plot VIIRS and Landsat data is worked through in this IPython Notebook.

CHAPTER 2

Working with generic raster formats

2.1 GeoTIFF

Pygaarst `raster` module provides a custom `GeoTIFF` class with methods and properties that aim at making it easy to open and process GeoTIFF files. It works with both single and multi-band GeoTIFF files. Under the hood, the GeoTIFF class uses GDAL, so the instance attributes it makes available are will look familiar to those who work with GDAL. The default public methods and attributes are:

class `pygaarst.raster.`**`GeoTIFF`**(*filepath*)

Represents a GeoTIFF file for data access and processing and provides a number of useful methods and attributes.

Arguments: filepath (str): the full or relative file path

`clone`(*newpath*, *newdata*)

Creates new GeoTIFF object from existing: new data, same georeference.

Arguments: newpath: valid file path newdata: numpy array, 2 or 3-dim

Returns: A raster.GeoTIFF object

`ij2xy`(*i*, *j*)

Converts array index pair(s) to easting/northing coordinate pairs(s).

NOTE: array coordinate origin is in the top left corner whereas easting/northing origin is in the bottom left corner. Easting and northing are floating point numbers, and refer to the top-left corner coordinate of the pixel. i runs from 0 to nrow-1, j from 0 to ncol-1. For i=nrow and j=ncol, the bottom-right corner coordinate of the bottom-right pixel will be returned. This is identical to the bottom- right corner.

Arguments: i (int): scalar or array of row coordinate index j (int): scalar or array of column coordinate index

Returns: x (float): scalar or array of easting coordinates y (float): scalar or array of northing coordinates

`simpleplot`()

Quick and dirty plot of each band (channel, dataset) in the image. Requires Matplotlib.

`xy2ij`(*x*, *y*, *precise=False*)

Convert easting/northing coordinate pair(s) to array coordinate pairs(s).

NOTE: see note at ij2xy()

Arguments: x (float): scalar or array of easting coordinates y (float): scalar or array of northing coordinates precise (bool): if true, return fractional array coordinates

Returns: i (int, or float): scalar or array of row coordinate index j (int, or float): scalar or array of column coordinate index

Lat
Latitude coordinate of each pixel corner, as an array

Lat_pxcenter
Latitude coordinate of each pixel center, as an array

Lon
Longitude coordinate of each pixel corner, as an array

Lon_pxcenter
Longitude coordinate of each pixel center, as an array

coordtrans
A PROJ4 Proj object, which is able to perform coordinate transformations

data
2D numpy array for single-band GeoTIFF file data. Otherwise, 3D.

delx
The sampling distance in x-direction, in physical units (eg metres)

dely
The sampling distance in y-direction, in physical units (eg metres). Negative in northern hemisphere.

easting
The x-coordinates of first row pixel corners, as a numpy array: upper-left corner of upper-left pixel to upper-right corner of upper-right pixel (ncol+1).

northing
The y-coordinates of first column pixel corners, as a numpy array: lower-left corner of lower-left pixel to upper-left corner of upper-left pixel (nrow+1).

proj4
The dataset's coordinate reference system as a PROJ4 string

projection
The dataset's coordinate reference system as a Well-Known String

x_pxcenter
The x-coordinates of pixel centers, as a numpy array ncol.

y_pxcenter
y-coordinates of pixel centers, nrow.

2.2 HDF4 and HDF-EOS

[TBC]

2.3 HDF5

Pygaarst `raster` module provides a custom `HDF5` class with methods and properties that aim at making it easy to open and process files in

class `pygaarst.raster.`**HDF5** (*filepath*)
A class providing access to a generic HDF5

Arguments: filepath (str): full or relative path to the data file

CHAPTER 3

Working with specific remote sensing data

3.1 Landsat

3.1.1 Usage

Pygaarst offers an intuitive and friendly way to access Landsat scene data and metadata as distributed as at-sensor scaled radiance data files ("Level1") in GeoTIFF format, one file per imaging band, as zipped tar archives.

The work is done by two classes: `Landsatscene`, which takes the name of the unzipped data directory, and `Landsatband`, which can be instantiated from a `Landsatscene` object:

```
>>> from pygaarst import raster
>>> basedir = "path/to/data"
>>> landsatscene = "LE70690142004201EDC01"
>>> sc = raster.Landsatscene(os.path.join(basedir, landsatscene))
>>> type(sc)
<class 'pygaarst.raster.Landsatscene'>
>>> b2 = sc.band2
>>> type(b2.data)
<type 'numpy.ndarray'>
>>> b2.data.shape
(8021, 8501)
```

The `Landsatscene` knows how to calculate often used radiometric quantities such as vegetation indices, or access the Landsat metadata. Both old and new metadata format is supported:

```
>>> ndvi = sc.NDVI
>>> type(ndvi)
<type 'numpy.ndarray'>
>>> sc.meta['IMAGE_ATTRIBUTES']['SUN_ELEVATION']
44.26873508
>>> sc.meta['PRODUCT_METADATA']['DATE_ACQUIRED']
datetime.date(2004, 7, 19)
```

`Landsatscene` also supports a filename infix in case all scene files have been pre-processed. A typical example would be if the GeoTIFF band files have been sub-setted and saved under a new file name that adds a label at the end of the filename, before the extension. For example:

```
>>> b2.data.shape
(8021, 8501)
>>> sc.infix = '_CLIP'
>>> b3 = sc.band3
>>> b3.data.shape
```

```
(347, 373)
>>> b2.filepath
'path/to/data/LE70690142004201EDC01_B2.TIF'
>>> b3.filepath
'path/to/data/LE70690142004201EDC01_B3_CLIP.TIF'
```

The `Landsatband` object knows its own geographic attributes and can translate the raw data into radiance and reflectance. For thermal IR bands, calculation of radiant (brightness) temperatures is supported:

```
>>> b3.radiance
array([[ 23.59606299,  24.83937008,  23.59606299, ...,  29.81259843,
         29.81259843,  29.81259843],
       [ 25.46102362,  24.21771654,  23.59606299, ...,  30.43425197,
         30.43425197,  29.19094488],
       ...,
       [ 41.0023622 ,  39.75905512,  40.38070866, ...,  90.11299213,
         91.35629921,  92.5996063 ],
       [ 40.38070866,  38.51574803,  41.0023622 , ...,  90.11299213,
         90.73464567,  90.73464567]])
>>> b3.Lat
array([[ 65.23978665,  65.23979005,  65.23979346, ...,  65.24086228,
         65.24086467,  65.24086706],
       [ 65.2400558 ,  65.2400592 ,  65.24006261, ...,  65.24113144,
         65.24113383,  65.24113622],
       ...,
       [ 65.33291157,  65.332915  ,  65.33291841, ...,  65.3339918 ,
         65.3339942 ,  65.3339966 ],
       [ 65.33318072,  65.33318414,  65.33318756, ...,  65.33426096,
         65.33426336,  65.33426576]])
>>> b6 = sc.band6L
>>> b6.tKelvin
array([[ 298.51857637,  298.51857637,  299.01776653, ...,  301.97175896,
         301.48420756,  301.48420756],
       [ 297.51409689,  298.01736166,  298.51857637, ...,  302.45745107,
         301.97175896,  301.48420756],
       ...,
       [ 294.96609241,  294.96609241,  294.96609241, ...,  290.23752626,
         290.77248753,  290.23752626],
       [ 294.96609241,  294.96609241,  294.96609241, ...,  290.77248753,
         290.77248753,  290.77248753]])
```

All parameters are, where provided, taken from the scene-specific metadata. Where not, they are taken from the published literature.

3.1.2 The Landsatscene class

class `pygaarst.raster.`**`Landsatscene`**(*dirname*)

A container object for TM/ETM+ L5/7 and OLI/TIRS L8 scenes

Arguments: dirname (str): the full or relative file path that contains all files

`NBR`

The Normalized Burn Index

`NDVI`

The Normalized Difference Vegetation Index

TIRband
Label of a suitable thermal infrared band for the scene. Used in loops over Landsat scenes from multiple platform/sensor combinations. Will return B6 for L4/5, B6H for L7, B10 for L8.

ltkcloud
Cloud masking and landcover classification with the Luo–Trishchenko–Khlopenkov algorithm (doi:10.1109/LGRS.2010.2095409).

Returns: A numpy array of the same shape as the input data The classes signify: - 1.0 - bare ground - 2.0 - ice/snow - 3.0 - water - 4.0 - cloud - 5.0 - vegetated soil

naivecloud
Heuristic cloud masking via thermal IR threshold

3.1.3 The Landsatband class

class pygaarst.raster.**Landsatband**(*filepath*, *band=None*, *scene=None*)
Represents a band of a Landsat scene.

Implemented: TM/ETM+ L5/7 and OLI/TIRS L8, old and new metadata format

newmetaformat
Checks if band uses old or new metadata format

radiance
Radiance in W/um/m^2/sr derived from DN and metadata, as numpy array

reflectance
Reflectance (0 .. 1) derived from DN and metadata, as numpy array

tKelvin
Radiant (brightness) temperature at the sensor in K, implemented for Landsat thermal infrared bands.

3.2 EO-1 Hyperion, ALI and general USGS Level 1 data

3.2.1 Hyperion and ALI

Hyperion imaging spectroscopy data and ALI multi-spectral imagery, processed to a Level 1 product, are distributed by the USGS in a format very similar to Landsat data. Pygaarst provides classes similar to the Landsat-specific classes for these sensors.

class pygaarst.raster.**Hyperionscene**(*dirname*)
A container object for EO-1 Hyperion scenes. Input: directory name, which is expected to contain all scene files.

get_datacube(*outfn*, *bandlist*, *islice=None*, *jslice=None*, *set_fh=False*)
Creates a rasterhelpers.Datacube object from a bandlist and the radiance data from the whole Hyperion scene.

Arguments: outfn (str): file path of the HDF5 file that stores the cube bandlist (sequence of str): a list or array of band names islice (sequence of int): list or array of row indices jslice (sequence of int): list or array of column indices set_fh (bool): should an open filehandle be set as an argument?

spectrum(*i_idx*, *j_idx*, *bands='calibrated'*, *bdsel=[ ]*)
Calculates the radiance spectrum for one pixel.

Arguments: i_idx (int): first coordinate index of the pixel j_idx (int): second coordinate index of the pixel bands (str): indicates the bands that are used

'calibrated' (default): only use calibrated bands 'high': use uncalibrated bands 225-242 'low': use uncalibrated bands 1-7 'all': use all available bands 'selected': use bdsel attribute or argument

bdsel: sequence data type containing band indices to select

class pygaarst.raster.**Hyperionband**(*filepath*, *band=None*, *scene=None*)
Represents a band of an EO-1 Hyperion scene.

radiance
Radiance in W / um / m^2 / sr derived from digital number and metadata, as numpy array

class pygaarst.raster.**ALIscene**(*dirname*)
A container object for EO-1 ALI scenes. Input: directory name, which is expected to contain all scene files.

class pygaarst.raster.**ALIband**(*filepath*, *band=None*, *scene=None*)
Represents a band of an EO-1 ALI scene.

radiance
Radiance in W / um / m^2 / sr derived from digital number and metadata, as numpy array

3.2.2 The USGSL1 parent classes

All of them inherit from parent classes called USGSL1scene and USGSL1band respectively.

class pygaarst.raster.**USGSL1scene**(*dirname*)
A container object for multi- and hyperspectral satellite imagery scenes as provided as Level 1 (at-sensor calibrated scaled radiance data) by various USGS data portals: Landsat 4/5 TM, Landsat 7 ETM+, Landsat 7 OLI/TIRS, EO1 ALI and EO1 Hyperion

Arguments: dirname (str): name of directory that contains all scene files.

get_normdiff(*label1*, *label2*)
Calculate a generic normalized difference index

Arguments: label1, label2 (str): valid band labels, usually of the form 'bandN'

class pygaarst.raster.**USGSL1band**(*filepath*, *band=None*, *scene=None*)
Represents a band of a USGSL1scene.

This class is intended to be used via chlid classes: Landsatband, Hyperionband, ALIband

sensor
Returns sensor name

spacecraft
Returns spacecraft name (L4, L5, L7, L8)

3.2.3 The metatadata parser

All these USGS-provided satellite remote sensing data use essentially the same metadata format. It is provided in a text file that can be recognized by the letters *MTL* in the filename. The data structure is nested naturally maps onto a dictionary of dictionaries, with unlimited nesting depth. Unfortunately it appears to be quite badly documented, or in any event I have been unable to locate a data description or specification.

For objects of type pygaarst.raster.USGSL1scene or its children (pygaarst.raster.Landsatscene, pygaarst.raster.ALIscene or pygaarst.raster.Hyperionscene), the parsed metadata dictionary is provided in the *meta* attribute:

```
>>> print sc.meta
{'IMAGE_ATTRIBUTES': {'GEOMETRIC_RMSE_MODEL': 2.868, 'CLOUD_COVER': 31.0, 'GEOMETRIC_RMSE_MODEL_X': 1
```

The parser is implemented via the module `pygaarst.mtlutils`. An arbitrary MTL file can be parsed by calling `pygaarst.mtlutils.parsemeta()`: pygaarst.mtlutils

Helpers for parsing MTL metadata files used by USGS Landsat and EO-1 (Hyperion and ALI) Level 1 GeoTIFF scene data archives

Created by Chris Waigl on 2014-04-20.

[2014-04-20] Refactoring original landsatutils.py, as MTL file format is also used by Hyperion and ALI.

exception `pygaarst.mtlutils.`**`MTLParseError`**
Custom exception: parse errors in Landsat or EO-1 MTL metadata files

`pygaarst.mtlutils.`**`parsemeta`**(*metadataloc*)
Parses the metadata.

Arguments: metadataloc: a filename or a directory.

Returns metadata dictionary

3.3 MODIS swath data

[This is a little harder...]

3.4 Suomi/NPP VIIRS

class `pygaarst.raster.`**`VIIRSHDF5`**(*filepath*, *geofilepath=None*, *variable=None*)
A class providing access to a VIIRS SDS HDF5 file or dataset Parameters: filepath: full or relative path to the data file geofilepath (optional): override georeference array file from metadata; full or relative path to georeference file variable (optional): name of a variable to access

`geodata`
Object representing the georeference data, in its entirety

`lats`
Latitudes as provided by georeference array

`lons`
Longitudes as provided by georeference array

3.5 MODAPS data download API client

This is a REST-full web service client that implements the NASA LAADSWEB data API (http://ladsweb.nascom.nasa.gov/data/api.html). Usage:

```
>>> from pygaarst import modapsclient as m
>>> a = m.ModapsClient()
>>> retvar = a.[methodname](args)
```

Module pygaarst.modapsclient

A client to do talk to MODAPS web services. See http://ladsweb.nascom.nasa.gov/data/web_services.html

Created by Chris Waigl on 2013-10-22.

class pygaarst.modapsclient.**ModapsClient**

Implements a client for MODAPS web service retrieval of satellite data, without post-processing

See http://ladsweb.nascom.nasa.gov/data/quickstart.html

getAllOrders(*email*)
All orders for an email address

getBands(*product*)
Available bands for a product

getBrowse(*fileId*)
fileIds is a single file-ID

getCollections(*product*)
Available collections for a product

getDataLayers(*product*)
Available data layers for a product

getDateCoverage(*collection*, *product*)
Available dates for a collection/product combination

TODO: add some result postprocessing - not a good format

getFileOnlineStatuses(*fileIds*)
fileIds is a comma-separated list of file-IDs

getFileProperties(*fileIds*)
fileIds is a comma-separated list of file-IDs

getFileUrls(*fileIds*)
fileIds is a comma-separated list of file-IDs

getMaxSearchResults()
Max number of search results that can be returned

getOrderStatus(*OrderID*)
Order status for an order ID. TODO: implement

getOrderUrl(*OrderID*)
Order URL(?) for order ID. TODO: implement

getPostProcessingTypes(*products*)
Products: comma-concatenated string of valid product labels

listCollections()
Lists all collections. Deprecated: use getCollections

listMapProjections()
Lists all available map projections

listProductGroups(*instrument*)
Lists all available product groups

listProducts()
Lists all available products

listProductsByInstrument(*instrument*, *group=None*)
Lists all available products for an instrument

listReprojectionParameters(*reprojectionName*)
Lists reprojection parameters for a reprojection

listSatelliteInstruments()
Lists all available satellite instruments

orderFiles(*FileIDs*)
Submits an order. TODO: implement

searchForFiles(*products*, *startTime*, *endTime*, *north*, *south*, *east*, *west*, *coordsOrTiles=u'coords'*, *dayNightBoth=u'DNB'*, *collection=5*)
Submits a search based on product, geography and time

searchForFilesByName(*collection*, *pattern*)
Submits a search based on a file name pattern

CHAPTER 4

Indices and tables

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

p

A

C

D

E

G

H

I

L