pygeobase Documentation

Release 0.6.2.post1.dev1+g18baa63

TU Wien

Sep 26, 2023

CONTENTS

This is the documentation of **pygeobase**.

The pygeobase package implements base class definitions for the I/O interface used in pytesmo, pynetcf, and other packages.

CHAPTER

ONE

USAGE

The Abstract base classes in this package are used to provide a consistent interface for reading various kinds of data.

1.1 Image datasets

When we talk about a image dataset we are generally talking about a dataset that can be represented through one or several two dimensional arrays. Such a dataset might consist of multiple layers or bands. It can have implicit or explicit geolocation information attached. In the simplest case all the datapoints of a image are referenced to the the same time. But we can also envision a reference timestamp for a image with a layer of exact time stamps for each observation.

The *pygeobase.io_base.ImageBase* implements the reading and writing of a single file whereas *pygeobase.io_base.MultiTemporalImageBase* is responsible for building the filename for a reference timestamp and using the ImageBase class for the io. In this way any number of underlying file formats can be supported.

Figure 1 shows *pygeobase.io_base.ImageBase* which is the abstract base class for implementing a reader for a single image linked to one file on disk. The `read`, `write`, `flush` and `close` methods have to be implemented. For reading from a dataset it is generally enough to implement the `read` and `close` methods and use dummy methods for `write` and `flush`. The `read` method must return a *pygeobase.object_base.Image* instance.

Fig. 1: Figure 1

A implemented class for the ImageBase can then be used in *pygeobase.io_base.MultiTemporalImageBase*. This class models a dataset consisting of several files on disk. Each file is linked to a reference timestamp from which the filename can be built. *pygeobase.io_base.MultiTemporalImageBase* can be used directly when configured correctly. Configuration means setting the `fname_templ` and the `datetime_format` so that it fits to the dataset. If the single files are stored in subfolders by e.g. month or day then the keyword `subpath_templ` can be used to specify that. Please see *pygeobase.io_base.MultiTemporalImageBase* for detailed information about each keyword.

1.1.1 Example for implementing a new image dataset

Let's imagine we have a regular daily dataset stored on a global regular grid of 0.1 degrees. The folder structure and filenames of the dataset are e.g.

- /2015/01/dataset_2015-01-01.dat
- /2015/01/dataset_2015-01-02.dat
- …
- /2015/02/dataset_2015-02-01.dat

For simplicities sake lets assume that the dat files are just pickled python dictionaries.

We could now write a new class based on *pygeobase.io_base.ImageBase* that reads one of these files:

```
from pygeobase.object_base import Image
from pygeobase.io_base import ImageBase
import pygeogrids.grids as grids
import pickle


class PickleImg(ImageBase):

    def __init__(self, filename, mode='r'):
        super(PickleImg, self).__init__(filename, mode=mode)
        self.grid = grids.genreg_grid(1, 1)

    def read(self, timestamp=None):

        data = pickle.load(self.filename)
        metadata = {'Type': 'pickle'}

        return Image(self.grid.arrlon,
                     self.grid.arrlat,
                     data,
                     metadata,
                     timestamp)

    def write(self, data):
        raise NotImplementedError()

    def flush(self):
        pass

    def close(self):
        pass
```

This new class `PickleImg` will read a pickled dictionary of data from the given filename. For the representation of the longitude and latitude of each datapoint the attributes of a `pygeogrids.grids.BasicGrid` object can be used but a regular numpy array would also do.

The next code snippet shows how this newly written class can be used in an implementation of *pygeobase.io_base.MultiTemporalImageBase*:

```
class PickleDs(MultiTemporalImageBase):

    def __init__(self, root_path):
        sub_path = ['%Y', '%m']
        fname_templ = "dataset_{datetime}.dat"
        datetime_format = "%Y-%m-%d"

        super(PickleDs, self).__init__(root_path, PickleImg,
                                       fname_templ=fname_templ,
                                       datetime_format=datetime_format,
                                       subpath_templ=sub_path)
```

The `sub_path` variable is a list of strings that build the path to the file from the python datetime object. The strftime syntax is used. `fname_templ` specifies the filename template in which `{datetime}` will be substituted by the string built by `datetime_format` according to the strftime syntax. There are more options to customize how the filename is build from a given python datetime. Please see the *pygeobase.io_base.MultiTemporalImageBase* documentation.

Please see the modindex for more details.

1.2 Working with a lot of small files

Some datasets are distributed in very small files like e.g. 3 minute parts of an orbit. Numerous applications can be sped up if a number of these files are read together and concatenated before furhter processing. For this use case the `pygeobase.io_base.IntervalReadingMixing` was developed.

The class does only work if the `tstamps_for_daterange` method is implemented. If this is the case it can be used to generate a new class based on an existing reader class like this:

```
class IntervalReadingPickeDs(IntervalReadingMixin, PickleDs):
    pass
```

Please consult the working test example `IntervalReadingTestDataset` in `tests/test_io_base.py`.

CHAPTER

TWO

CONTENTS

2.1 pygeobase

The pygeobase package implements base class definitions for the I/O interface used in pytesmo, pynetCF, and other packages.

2.1.1 Citation

If you use the software in a publication then please cite it using the Zenodo DOI. Be aware that this badge links to the latest package version.

Please select your specific version at https://doi.org/10.5281/zenodo.846761 to get the DOI of that version. You should normally always use the DOI for the specific version of your record in citations. This is to ensure that other researchers can access the exact research artefact you used for reproducibility.

You can find additional information regarding DOI versioning at http://help.zenodo.org/#versioning

2.1.2 Installation

This package should be installable through pip:

```
pip install pygeobase
```

Its only dependecy is `numpy`. But to use it effectively you will also probably want to install pygeogrids.

2.1.3 Contribute

We are happy if you want to contribute. Please raise an issue explaining what is missing or if you find a bug. We will also gladly accept pull requests against our master branch for new features or bug fixes.

Development setup

For Development we recommend a `conda` environment. You can create one including test dependencies and debugger by running `conda env create -f conda_requirements.yml`. This will create a new `pygeobase_env` environment which you can activate by using `source activate pygeobase_env`.

Example installation script

The following script will install miniconda and setup the environment on a UNIX like system. Miniconda will be installed into `$HOME/miniconda`.

```
wget https://repo.continuum.io/miniconda/Miniconda-latest-Linux-x86_64.sh -O miniconda.sh
bash miniconda.sh -b -p $HOME/miniconda
export PATH="$HOME/miniconda/bin:$PATH"
git clone git@github.com:TUW-GEO/pygeobase.git pygeobase
cd pygeobase
conda env create -f conda_environment.yml
source activate pygeobase_env
```

This script adds `$HOME/miniconda/bin` temporarily to the `PATH` to do this permanently add `export PATH="$HOME/miniconda/bin:$PATH"` to your `.bashrc` or `.zshrc`

The last line in the example activates the `pygeobase_env` environment.

After that you should be able to run:

```
python setup.py test
```

to run the test suite.

Guidelines

If you want to contribute please follow these steps:

- Fork the pygeobase repository to your account
- Clone the repository
- make a new feature branch from the pygeobase master branch
- Add your feature
- Please include tests for your contributions in one of the test directories. We use py.test so a simple function called test_my_feature is enough
- submit a pull request to our master branch

2.1.4 Note

This project has been set up using PyScaffold 4.5. For details and usage information on PyScaffold see https://pyscaffold.org/.

2.2 License

2.3 Contributors

- Sebastian Hahn <sebastian.hahn@geo.tuwien.ac.at>
- Christoph Paulik <cpaulik@vandersat.com>
- Thomas Mistelbauer <thomas.mistelbauer@eodc.eu>
- Christoph Reimer <christoph.reimer@eodc.eu>

2.4 Changelog

2.4.1 Unreleased changes in master

-

2.4.2 Version 0.6.2

- Extract max_dist keyword from kwargs in read_lon_lat and pass on to pygeogrids nearest neighbor search

2.4.3 Version 0.6.1

- Replace read_ts with read call from io class (see https://github.com/TUW-GEO/pygeobase/pull/46)

2.4.4 Version 0.6.0

- Replace read_ts with read in time series base class

2.4.5 Version 0.5.0

- Use github actions for CI
- Remove deprecated classes and methods
- Update copyright year

2.4.6 Version 0.4.0

- Update readme
- Specify sphinx version to fix rtd

2.4.7 Version 0.3.18

- Update pyscaffold version in setup.py because of compatibility issues with setuptools 39

2.4.8 Version 0.3.17

- Update classifier for pipy upload

2.4.9 Version 0.3.16

- Fix cell switch bug
- Update copyright year

2.4.10 Version 0.3.15

- Allow reading of data in append mode.

2.4.11 Version 0.3.14

- iter_gp does not longer stop if an IOError occurs in the subclass. It catches the error warns the user and returns None as the dataset object.

2.4.12 Version 0.3.13

- Add `IntervalReadingMixin` for reading files in bigger chunks based on intervals.

2.4.13 Version 0.3.12

- change `resample_data` interface to get weights directly instead of windowRadius.
- Add `dtype` and `__getitem__` to `Image` class.

2.4.14 Version 0.3.11

- Fix issue 10 and add new spatial subset method

2.4.15 Version 0.3.10

- allow iteration over time series to take keyword arguments.

2.4.16 Version 0.3.9

- Fix bug in passing of keyword arguments to Image readers.

2.4.17 Version 0.3.8

- Make the Image object backward compatible.
- Improve documentation.

2.4.18 Version 0.3.7

- Fix issue #28, reading from non-existing cell

2.4.19 Version 0.3.6

- Add exception handling for opening a files
- Add lon, lat information in writing operation of GriddedBase

2.4.20 Version 0.3.5

- Remove unnecessary dependencies and improve documentation.

2.4.21 Version 0.3.4

- Add conda requirements file
- Add object base classes for time series and image
- Add image base and multitemporal-image base classes

2.4.22 Version 0.3.3

- Update requirements

2.4.23 Version 0.3.2

- Add ImageBase class (moved from pytesmo)
- Fixing documentation

2.4.24 Version 0.3.1

- Updating pyscaffold version to

2.4.25 Version 0.3.0

- New GriddedBase class
- Slight changes in the method names

2.4.26 Version 0.2.0

- Support of ioclass keyword arguments
- Fix iteration inconsistency

2.4.27 Version 0.1

- First developer release

2.5 pygeobase

2.5.1 pygeobase package

Submodules

pygeobase.io_base module

class pygeobase.io_base.**GriddedBase**(*path*, *grid*, *ioclass*, *mode='r'*, *fn_format='{:04d}'*, *ioclass_kws=None*)

Bases: `object`

The GriddedBase class uses another IO class together with a grid object to read/write a dataset under the given path.

Parameters

- **path** (*string*) – Path to dataset.
- **grid** (*pygeogrids.BasicGrid of CellGrid instance*) – Grid on which the time series data is stored.
- **ioclass** (*class*) – IO class.
- **mode** (*str, optional*) – File mode and can be read 'r', write 'w' or append 'a'. Default: 'r'
- **fn_format** (*str, optional*) – The string format of the cell files. Default: '{:04d}'
- **ioclass_kws** (*dict, optional*) – Additional keyword arguments for the ioclass. Default: None

close()

Close file.

flush()

Flush data.

get_spatial_subset(*gpis=None*, *cells=None*, *ll_bbox=None*, *grid=None*)

Select spatial subset and return data set with new grid.

Parameters

- **gpis** (*numpy.ndarray*) – Grid point indices.
- **cells** (*numpy.ndarray*) – Cell number.
- **ll_bbox** (*tuple (latmin, latmax, lonmin, lonmax)*) – Lat/Lon bounding box
- **grid** (*pygeogrids.CellGrid*) – Grid object.

Returns

dataset – New data set with for spatial subset.

Return type

GriddedBase or child

iter_gp(***kwargs*)

Yield all values for all grid points.

Yields

- **data** (*pandas.DataFrame*) – Data set.
- **gp** (*int*) – Grid point.

read(**args*, ***kwargs*)

Takes either 1 or 2 arguments and calls the correct function which is either reading the gpi directly or finding the nearest gpi from given lat,lon coordinates and then reading it

write(**args*, ***kwargs*)

Takes either 1 or 2 arguments and calls the correct function which is either writing the gpi directly or finding the nearest gpi from given lat,lon coordinates and then writing it.

class pygeobase.io_base.**GriddedTsBase**(*path*, *grid*, *ioclass*, *mode='r'*, *fn_format='{:04d}'*, *ioclass_kws=None*)

Bases: *GriddedBase*

The GriddedTsBase class uses another IO class together with a grid object to read/write a time series dataset under the given path.

class pygeobase.io_base.**ImageBase**(*filename*, *mode='r'*, ***kwargs*)

Bases: object

ImageBase class serves as a template for i/o objects used for reading and writing image data.

Parameters

- **filename** (*str*) – Filename path.
- **mode** (*str, optional*) – Opening mode. Default: r

abstract close()

Close file.

abstract flush()

Flush data.

abstract read(***kwargs*)

Read data of an image file.

Returns
image – pygeobase.object_base.Image object

Return type
object

read_masked_data(***kwargs*)

Read data of an image file and mask the data according to specifications.

Returns
image – pygeobase.object_base.Image object

Return type
object

resample_data(*image*, *index*, *distance*, *weights*, ***kwargs*)

Takes an image and resample (interpolate) the image data to arbitrary defined locations given by index and distance.

The default implementation just takes the weighted mean of all defined distances.

Parameters

- **image** (*:py:class`pygeobase.object_base.Image` or numpy.recarray*) – Image or numpy.recarray like object with shape = (x,)
- **index** (*np.array*) – Index into image data defining a look-up table for data elements used in the interpolation process for each defined target location. For each point in image the neighbors in the targed grid are in the index array. This array is of shape (x, max_neighbors)
- **distance** (*np.array*) – Array representing the distances of the image data to the arbitrary defined locations. The distances of points not to use are set to np.inf This array is of shape (x, max_neighbors)
- **weights** (*np.array*) – Array representing the weights of the image data that should be used during resampling. The weights of points not to use are set to np.nan This array is of shape (x, max_neighbors)

Returns

target – dictionary with a numpy.ndarray for each field in the input image. We can not return a image here since we do not know the target latitudes and longitudes.

Return type

dict

abstract write(*image*, ***kwargs*)

Write data to an image file.

Parameters

image (*object*) – pygeobase.object_base.Image object

class pygeobase.io_base.**IntervalReadingMixin**(**args*, ***kwargs*)

Bases: object

Class overwrites functions to enable reading of multiple images in a time interval as one chunk. E.g. reading 3 minute files in 50 minute half-orbit chunks.

read(*interval*, ***kwargs*)

Return an image for a specific interval.

Parameters

interval (*tuple*) – (start, end)

Returns

image – pygeobase.object_base.Image object

Return type

object

tstamps_for_daterange(*startdate*, *enddate*)

Here we split the period between startdate and enddate into intervals of size self.chunk_minutes. These interval reference dates are then translated to the actual file dates during reading of the chunks.

Returns

intervals – list of (start, end) of intervals

Return type
list of tuples

class pygeobase.io_base.**MultiTemporalImageBase**(*path*, *ioclass*, *mode='r'*, *fname_templ=''*, *datetime_format=''*, *subpath_templ=None*, *ioclass_kws=None*, *exact_templ=True*, *dtime_placeholder='datetime'*)

Bases: object

The MultiTemporalImageBase class make use of an ImageBase object to read/write a sequence of multi temporal images under a given path.

Parameters

- **path** (*string*) – Path to dataset.
- **ioclass** (*class*) – IO class.
- **mode** (*str, optional*) – File mode and can be read 'r', write 'w' or append 'a'. Default: 'r'
- **fname_templ** (*str*) – Filename template of the data to read. Default placeholder for parsing datetime information into the fname_templ is "{datetime}". e.g. "ASCAT_{datetime}_image.nc" will be translated into the filename ASCAT_20070101_image.nc for the date 2007-01-01.
- **datetime_format** (*str*) – String specifying the format of the datetime object to be parsed into the fname_template. e.g. "%Y/%m" will result in 2007/01 for datetime 2007-01-01 12:15:00
- **subpath_templ** (*list, optional*) – If given it is used to generate a sub-paths from the given timestamp. Each item in the list represents one folder level. This can be used if the files for May 2007 are e.g. in folders 2007/05/ then the files can be accessed via the list ['%Y', '%m'].
- **ioclass_kws** (*dict*) – Additional keyword arguments for the ioclass.
- **exact_templ** (*boolean, optional*) – If True then the fname_templ matches the filename exactly. If False then the fname_templ will be used in glob to find the file.
- **dtime_placeholder** (*str*) – String used in fname_templ as placeholder for datetime. Default value is "datetime".

close()

Close file.

daily_images(*day*, ***kwargs*)

Yield all images for a day.

Parameters
day (*datetime.date*) –

Returns
img – pygeobase.object_base.Image object

Return type
object

flush()

Flush data.

get_tstamp_from_filename(*filename*)

Return the timestamp contained in a given file name in accordance to the defined fname_templ.

Parameters
filename (*string*) – File name.

Returns
tstamp – Time stamp according to fname_templ as datetime object.

Return type
datetime.dateime

iter_images(*start_date*, *end_date*, ***kwargs*)

Yield all images for a given date range.

Parameters

- **start_date** (*datetime.date or datetime.datetime*) – start date
- **end_date** (*datetime.date or datetime.datetime*) – end date

Returns
image – pygeobase.object_base.Image object

Return type
object

read(*timestamp*, ***kwargs*)

Return an image for a specific timestamp.

Parameters
timestamp (*datetime.datetime*) – Time stamp.

Returns
image – pygeobase.object_base.Image object

Return type
object

resample_image(**args*, ***kwargs*)

tstamps_for_daterange(*start_date*, *end_date*)

Return all valid timestamps in a given date range. This method must be implemented if iteration over images should be possible.

Parameters

- **start_date** (*datetime.date or datetime.datetime*) – start date
- **end_date** (*datetime.date or datetime.datetime*) – end date

Returns
dates – list of datetimes

Return type
list

write(*timestamp*, *data*, ***kwargs*)

Write image data for a given timestamp.

Parameters

- **timestamp** (*datetime.datetime*) – exact timestamp of the image
- **data** (*object*) – pygeobase.object_base.Image object

class pygeobase.io_base.StaticBase(*filename*, *mode='r'*, ***kwargs*)

Bases: object

The StaticBase class serves as a template for i/o objects used in GriddedStaticBase.

Parameters

- **filename** (*str*) – File name.
- **mode** (*str, optional*) – Opening mode. Default: r

abstract close()

Close file.

abstract flush()

Flush data.

abstract read(*gpi*)

Read data for given grid point.

Parameters
gpi (*int*) – Grid point index.

Returns
data – Data set.

Return type
numpy.ndarray

abstract write(*data*)

Write data.

Parameters
data (*numpy.ndarray*) – Data records.

class pygeobase.io_base.TsBase(*filename*, *mode='r'*, ***kwargs*)

Bases: object

The TsBase class serves as a template for i/o objects used in GriddedTsBase.

Parameters

- **filename** (*str*) – File name.
- **mode** (*str, optional*) – Opening mode. Default: r

close()

Close file.

flush()

Flush data.

abstract read(*gpi*, ***kwargs*)

Read time series data for given grid point.

Parameters
gpi (*int*) – Grid point index.

Returns
data – pygeobase.object_base.TS object.

Return type
object

abstract write(*gpi*, *data*, ***kwargs*)

Write data.

Parameters

- **gpi** (*int*) – Grid point index.
- **data** (*object*) – pygeobase.object_base.TS object.

pygeobase.object_base module

class pygeobase.object_base.**Image**(*lon*, *lat*, *data*, *metadata*, *timestamp*, *timekey=None*)

Bases: object

The Image class represents the base object of an image.

Parameters

- **lon** (*numpy.array*) – array of longitudes
- **lat** (*numpy.array*) – array of latitudes
- **data** (*dict*) – dictionary of numpy arrays that holds the image data for each variable of the dataset
- **metadata** (*dict*) – dictionary that holds metadata
- **timestamp** (*datetime.datetime*) – exact timestamp of the image
- **timekey** (*str, optional*) – Key of the time variable, if available, stored in data dictionary.

property dtype

Fake numpy recarray dtype field based on the dictionary keys and the dtype of the numpy array.

class pygeobase.object_base.**TS**(*gpi*, *lon*, *lat*, *data*, *metadata*)

Bases: object

The TS class represents the base object of a time series.

Parameters

- **lon** (*float*) – Longitude of the time series
- **lat** (*float*) – Latitude of the time series
- **data** (*pandas.DataFrame*) – Pandas DataFrame that holds data for each variable of the time series
- **metadata** (*dict*) – dictionary that holds metadata

plot(**args*, ***kwargs*)

wrapper for pandas.DataFrame.plot which adds title to plot and drops NaN values for plotting

Returns

ax – matplotlib axes of the plot

Return type

axes

pygeobase.utils module

pygeobase.utils.**split_daterange_in_intervals**(*start*, *end*, *mi*)

Split a daterange in non overlapping intervals of mi minutes - 1 microsecond.

Parameters

- **start** (*datetime.datetime*) – start of the daterange
- **end** (*datetime.datetime*) – end of the daterange
- **mi** (*int*) – Minutes of the intervals

Returns

intervals – list of (start, end) of intervals

Return type

list of tuples

Module contents

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

T

W