

pyjeo

an open source image processing library
in Python

Pieter Kempeneers
Matera, Italy, 13 June 2022

Installation

- ▶ source code available in [github](#)
- ▶ dependencies (C/C++ code):
 - ▶ [miallib](#)
 - ▶ [jiplib](#) (derived from pktools with Python interface)

Installation

Build docker image using **Dockerfile** (based on a debian11 image):

```
docker build -t deb11_pyjeo_public:0.1.8 -f  
Dockerfile_deb11_pyjeo_public .
```

Run pyjeo in Docker container:

```
docker run --rm deb11_pyjeo_public:0.1.8 python3 -c "import _pyjeo as _  
pj; _jim = _pj.Jim(ncol = _10, _nrow = _10, _nband = _3); _print(jim.  
properties.nrofBand())"
```

Methods

- ▶ Methods directly operate on objects, i.e., instances of a class
- ▶ Methods can change objects in-place (overwrite input)
- ▶ No object is returned

```
jim.geometry.cropBand(0)
```

jim has been cropped in-place and `None` is returned

Functions

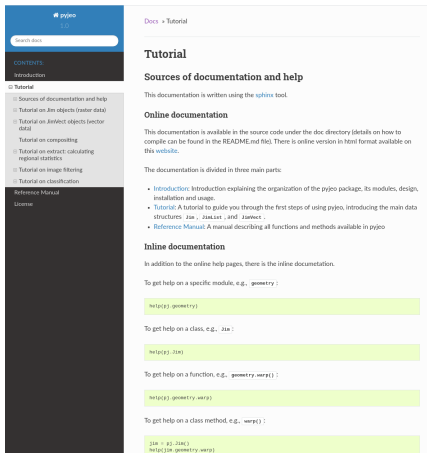
- ▶ Functions that operate on objects must have the objects passed as arguments
- ▶ Functions leave their arguments unaltered
- ▶ A new object (newobject) is returned

```
jim_cropped = pj.geometry.cropBand(jim, 0)
```

`jim` is unaltered and a `Jim` object is returned

Documentation

The documentation is **online** and can be accessed outside the Commission.



The screenshot displays the Pyjeo documentation website. On the left is a dark navigation sidebar with a search bar and a 'CONTENTS' section. The 'Tutorial' section is expanded, showing links to 'Sources of documentation and help', 'Tutorial on JIM objects (raster data)', 'Tutorial on JIM objects (vector data)', 'Tutorial on compositing', 'Tutorial on extract: calculating regional statistics', 'Tutorial on image filtering', and 'Tutorial on classification'. Below these are links for 'Reference Manual' and 'License'. The main content area on the right is titled 'Tutorial' and includes sections for 'Sources of documentation and help', 'Online documentation', and 'Inline documentation'. The 'Online documentation' section mentions that the documentation is available in the source code under the 'doc' directory and provides a link to the online version. The 'Inline documentation' section explains that in addition to online help pages, there is inline documentation, and provides examples of how to use the `help()` function for modules, classes, and functions.

pyjeo 0.0

Search docs

CONTENTS

Introduction

Tutorial

- Sources of documentation and help
- Tutorial on JIM objects (raster data)
- Tutorial on JIM objects (vector data)
- Tutorial on compositing
- Tutorial on extract: calculating regional statistics
- Tutorial on image filtering
- Tutorial on classification

Reference Manual

License

Docs » Tutorial

Tutorial

Sources of documentation and help

This documentation is written using the [sphinx](#) tool.

Online documentation

This documentation is available in the source code under the `doc` directory (details on how to compile can be found in the `README.md` file). There is online version in html format available on [this website](#).

The documentation is divided in three main parts:

- Introduction:** Introduction explaining the organization of the `pyjeo` package, its modules, design, installation and usage.
- Tutorial:** A tutorial to guide you through the first steps of using `pyjeo`, introducing the main data structures: `Jim`, `JimList`, and `JimVec`.
- Reference Manual:** A manual describing all functions and methods available in `pyjeo`.

Inline documentation

In addition to the online help pages, there is the inline documentation.

To get help on a specific module, e.g., `geometry` :

```
help(pyjeo.geometry)
```

To get help on a class, e.g., `Jim` :

```
help(pyjeo.Jim)
```

To get help on a function, e.g., `geometry.warp()` :

```
help(pyjeo.geometry.warp)
```

To get help on a class method, e.g., `warp()` :

```
Jim = pyjeo.Jim()  
help(jim.geometry.warp)
```

Documentation (inline)

- ▶ To get help on a specific module, e.g., geometry:

```
help (pj.geometry)
```

- ▶ To get help on a class, e.g., Jim:

```
help (pj.Jim)
```

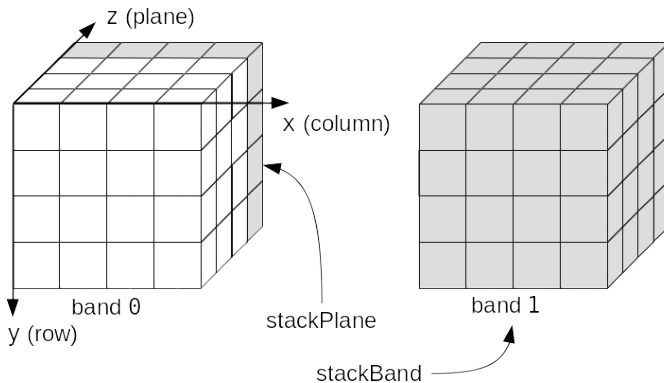
- ▶ To get help on a function, e.g., geometry.warp():

```
help (pj.geometry.warp)
```

Check also the online [tutorial](#).

Data model: Jim

Jim: pyjeo object for multi-band 3D raster data



Data model: Jim

- ▶ Each band represents a 3D contiguous array in memory: space (2) + plane (1)
- ▶ Planes are typically used for temporal/spectral/volumetric data
- ▶ data cube is defined in a single spatial reference system (geotransform and projection)

Data model: JimVect

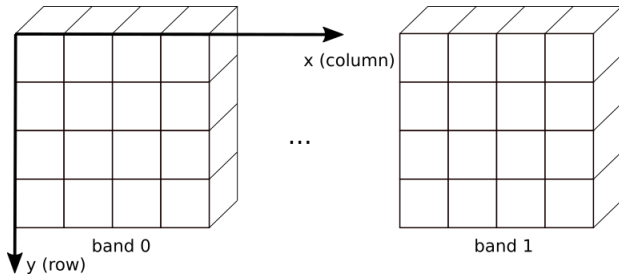
JimVect: pyjeo object for vector data

- ▶ References to file path that represents a vector
- ▶ File format must be supported by GDAL
- ▶ File can be virtual (in memory only)

Reading/writing geospatial data

As a default, a multi-band raster file is read as a single plane multi-band Jim object.

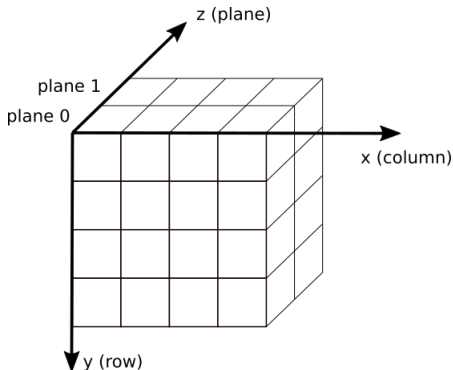
```
jim = pj.Jim('/path/to/raster.tif')
```



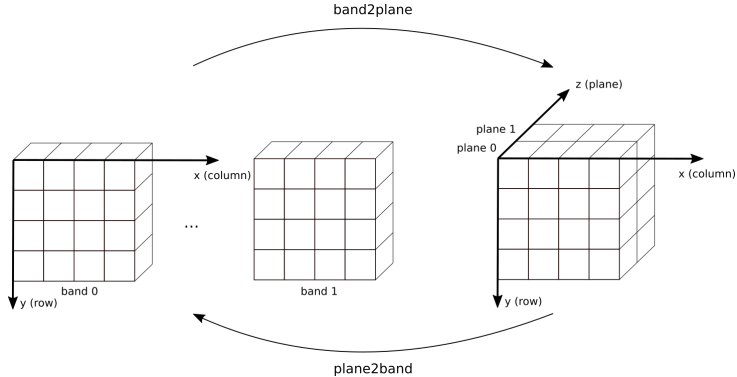
Reading/writing geospatial data

To open the image as a 3D multi-plane Jim object, use the `band2plane` argument

```
jim = pj.Jim('/path/to/raster.tif', band2plane = True)
```



Converting bands and planes



Bridging Jim to third party libraries

pyjeo Jim objects can be converted to:

- ▶ **Numpy** array objects
- ▶ **xarray** objects)

Conversion can be performed with memory copy:

```
jim = pj.np2jim(nparray)
nparray = pj.jim2np(jim)
```

Conversion can be performed without memory copy:

```
jim.np()[:] = nparray
nparray = jim.np() #careful!
```

The Jim object should remain the owner of the data and the referenced Numpy array object cannot be altered in shape and data type nor destroyed.

Bridging Jim to third party libraries

Numpy arrays do not have an attribute for a spatial reference system.

```
jim = pj.np2jim(nparray)
jim.properties.setGeoTransform([a,b,c,d,e,f])
jim.properties.setProjection('epsg:3035')
```

where the geotransform array `[a,b,c,d,e,f]` can also be copied from another Jim object.

```
gt = jim0.properties.getGeoTransform()
proj = jim0.properties.getProjection()
```

Bridging Jim to third party libraries

Example: in-place Gaussian filtering using ndimage

```
from scipy import ndimage  
jim.numpy()[:] = ndimage.gaussian_filter(jim.numpy(), 2)
```


Bridging JimVect to third party libraries

pyjeo JimVect objects can be converted to:

- ▶ Python dictionaries
- ▶ **Numpy** array objects
- ▶ **pandas** objects
- ▶ **geopandas** objects

```
dictobject = v.dict()
```

```
nparray = v.np()
```

Bridging JimVect to third party libraries

Convert JimVect to pandas object

```
import pandas as pd
panda_object = pd.DataFrame(v.dict())
```

Convert JimVect to geopandas object

```
import geopandas as gpd
v = pj.JimVect('vector.shp')
#convert to GeoJSON in memory
vjson = pj.JimVect(v,output='/vsimem/pj.json', oformat = 'GeoJSON')
vjson.io.close()
#create geopandas dataframe from GeoJSON file in memory
gdf = gpd.read_file('/vsimem/pj.json')
```

Conclusions pyjeo

- ▶ open source and released under GPLv3 license
- ▶ documentation available online and inline
- ▶ automatic tiling mechanism for upscaling

Thank you



©European Union 2020

Unless otherwise noted the reuse of this presentation is authorised under the CC BY 4.0 license. For any use or reproduction of elements that are not owned by the EU, permission may need to be sought directly from the respective right holders.

Slide xx: element concerned, source: e.g. Fotolia.com; Slide xx: element concerned, source: e.g. iStock.com