

# pyjeo

## Course for Spatial Ecology 2023

*Pieter Kempeneers*  
*European Commission, Joint Research Centre, 03 May 2023*

# Time table

Morning	<b>Introduction and basic processing</b>
17:00-17:30	Installation
17:30-17:45	Introduction to the documentation
17:45-18:00	Basic data model: Jim and JimVect
18:00-18:20	<b>Coffee break</b>
18:20-18:45	Hands-on tutorial: reading/writing images
18:45-19:30	Bridging to third party libraries: numpy/scipy, (geo)pandas, and xarray
19:30-19:45	Conclusions and outlook

# Installation

- ▶ on your local computer
  - ▶ Linux: install from **source**
  - ▶ Windows / Mac: via Docker (**Dockerfile** available)

# Methods

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

```
jim.geometry.cropBand(0)  
# jim has been cropped in-place
```

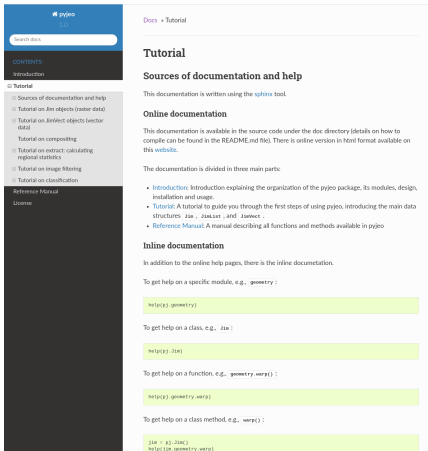
# Functions

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

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

# Documentation

The documentation is **online**.



**pygeo**  
1.0

Search docs

CONTENTS

Introduction

**Tutorial**

- Sources of documentation and help
- Tutorial on Jim objects (raster data)
- Tutorial on JirVect 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 `pygeo` package, its modules, design, installation and usage.
- Tutorial:** A tutorial to guide you through the first steps of using `pygeo`, introducing the main data structures `Jim`, `JimVect`, and `JimVect`.
- Reference Manual:** A manual describing all functions and methods available in `pygeo`.

### 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(py.geometry)
```

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

```
help(py.Jim)
```

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

```
help(py.geometry.warp)
```

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

```
Jim = py.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)
```

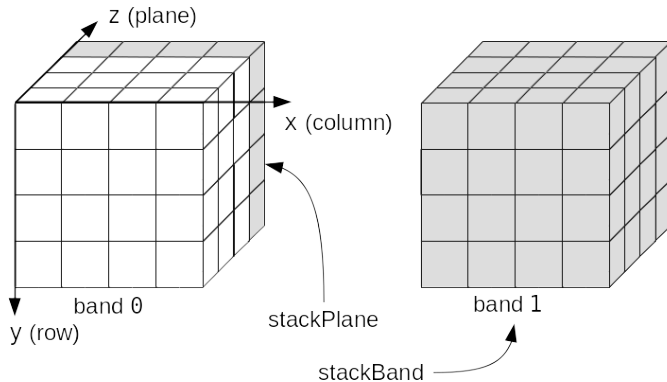
- ▶ To get help on a function, e.g., geometry.warp():

```
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
- ▶ A data cube is defined in a single spatial reference system (geotransform and projection)
- ▶ Planes and bands can be labeled

## Exercise 2

# Data model: JimVect

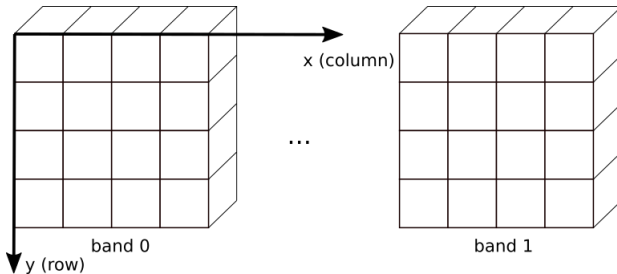
**JimVect: pyjeo object for vector data**

- ▶ 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.

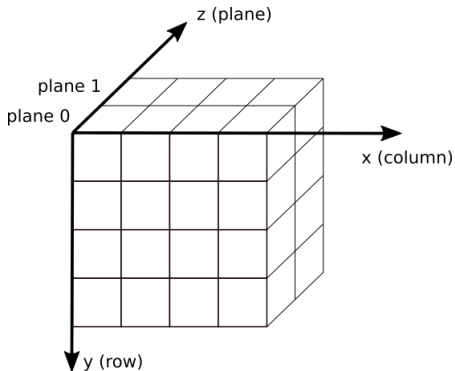
```
jim = pj.Jim('/path/to/raster.tif')
```



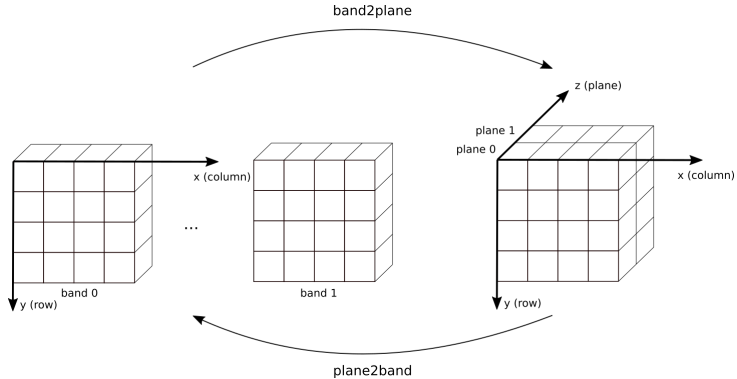
# Reading/writing geospatial data

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



## Exercise 3

# 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)
```

## Exercise 4



# Compositing multi-plane images

Compositing is the process of resolving overlapping pixels in a multi-plane image  
Pixels indicated as *no data* are not considered for the composite rule

```
jim.geometry.reducePlane('median', nodata = 0)
```

built-in rules: 'mean', 'median', 'min', 'max', 'overwrite'

# Custom compositing rules

Custom compositing rules can be created via a call-back function.

```
def getMax(reduced, plane):  
    return pj.pixops.supremum(reduced, plane)  
jim_stacked.geometry.reducePlane(getMax)
```

Call-back functions cannot be combined with the parameters `ref_band` and `nodata`

# 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
vjjson = pj.JimVect(v,output='/vsimem/pj.json', oformat = 'GeoJSON')
vjjson.io.close()
#create geopandas dataframe from GeoJSON file in memory
gdf = gpd.read_file('/vsimem/pj.json')
```

# Data extraction and regional statistics

The `extract` method in the `geometry` module deals with data extraction and regional statistics.

Typical example: calculate the mean value of all pixels in a Jim object that are covered by a polygon in a JimVect object.

Result: new JimVect object (statistics are stored in the field data of the features)

The number of fields depend on:

- ▶ the number of statistical measures (e.g., mean, standard deviation)
- ▶ number of bands
- ▶ number of planes

# Data extraction and regional statistics

```
v = pj.geometry.extract(sample, jim, rule='mean',  
output='/vsimem/pj.json', oformat='GeoJSON')
```

- ▶ Check the online documentation for the list of **supported rules**
- ▶ To exclude pixel values masked as “no data”, set the parameter `srcnodata` (combine with `bndnodata`)
- ▶ To exclude pixels near the border of a polygon, set the parameter **buffer** to a negative value

# Conclusions pyjeo

- ▶ open source and released under GPLv3 license
- ▶ can be installed locally or in docker container
- ▶ documentation available online and inline
- ▶ automatic tiling mechanism for upscaling