# FlussTools

***Release latest***

**FlussTeam**

**Feb 25, 2021**

# CONTENTS

The analysis, research and science-based design of fluvial ecosystems involve complex challenges for interdisciplinary experienced teams. We have created *flusstools* to meet the complex challenges and to at least partially automate time-consuming, repetitive processes. "We" stands for individuals with a great passion for rivers (German: "Flüsse") and programming. Most of us work (or have worked) at the University of Stuttgart (Germany) at the Institute for Modelling Hydraulic and Environmental Systems. In the context of our scientific endeavor, we have a strong commitment to transparent open-source applications. With *flusstools*, we want to share our research-based open-source algorithms with a broad interest group in a well documented form. We welcome new team members (for example to add or amend a module) at any time - read more in the *Become a contributor* section.

Currently, *flusstools* comes with the following modules:

- *geotools* - versatile functions for processing spatial data for fluvial ecosystem analyses based on gdal and other open source libraries.
- *fuzzycorr* - a map comparison toolkit that builds on fuzzy sets to assesss the accuracy of (numerical) river models (principal developer: Beatriz Negreiros).
- *lidartools* - *Python* wrappers for lastools (forked and modified from Kenny Larrieu).

**Note:** The documentation is also as available as style-adapted PDF.

CHAPTER

ONE

# INSTALLATION

To work with *flusstools*, a couple of dependencies have to be installed and depending on what platform you are working (*Window* or *Linux*), it might be preferable to use either a *PIP3* or a *CONDA* environment. We have made good experience with *conda* environments in *Windows* and with *pip* environments in *Linux* (*Ubuntu*). However, both *pip* and *conda* both work fine on both platforms (and also with *macOS*). Since *Python 3.4* (and *Python 2.7.9*), *pip* is installed with the basic *Python* installation (download and install Python if you do not want to use *conda*). Find the instructions for installing *Anaconda* tailored for your ooperating system (*Linux*, *Windows*, or *macOS*) in the Anaconda docs. The following paragraphs guide through setting up your system for the best experience with *flusstools* (and basically any geo-spatial *Python* application).

## 1.1 Get ready on Windows

On *Windows*, a convenient option for working with *flusstools* is to use a conda environment. In addition, *GitBash* is necessary to clone (download) *flusstools* (and to keep posted on updates). In detail:

- Install *Anaconda*, for example, as described on hydro-informatics.github.io (IDEs).
- Alternatively, download and install Python, open Windows Command Prompt, and make sure to upgrade *pip* with your basic *Python* installer by typing: `python -m pip install -U pip`.
- Download and install *GitBash*.
- We recommend to work with an *IDE*, such as PyCharm or Spyder, which is natively implemented in the *Anaconda* installation.
- Download and install QGIS to visualize and draw geospatial data.

## 1.2 Get ready on Linux

### 1.2.1 Optional: Use a Virtual Machine (VM)

Either download a net-installer *ISO* of Debian Linux, Ubuntu, or one of its light-weight spin-offs such as the Lubuntu, and install one of theses images as a Virtual Machine (VM). To get started with VMs read the introduction to VMs on hydro-informatics.github.io (about VMs). Installing any other the *Linux* VM works similar, as described on hydro-informatics.github.io (VirtualBox) for *Debian Linux*. Just use the *ISO* image in lieu of the *Debian Linux ISO*. After installing *Linux* as a VM, make sure to:

- Install Guest Additions for *Linux* VMs in *VirtualBox*.
- Enable folder sharing between the host and guest (*Debian*, *Ubuntu*, or *Lubuntu* image).

Other system setups described on hydro-informatics.github.io (VM) (e.g., *Wine*) are not required in the following.

### 1.2.2 Prepare your system

Open *Terminal* and update the system:

```
sudo apt update && sudo apt full-upgrade -y
```

### 1.2.3 Update Python references

Some (basic) *Linux* distributions still have *Python2* implemented as base interpreter to be used when `python` is called in *Terminal*. However, *Python2* usage is deprecated, and therefore, we want to make sure to robustly use *Python3* for running any *Python* script. Check out the installed *Python3* versions:

```
ls /usr/bin/python*


/usr/bin/python  /usr/bin/python2  /usr/bin/python2.7  /usr/bin/python3  /usr/bin/
↪python3.8  /usr/bin/python3.8m  /usr/bin/python3m
```

In this example, *Python2.7* and *Python3.8* are installed. To overwrite *Python2* usage, set the `python` environment variable so that it points at *Python3*:

```
sudo update-alternatives --install /usr/bin/python python /usr/bin/python3.6 2
alias python=python3
```

### 1.2.4 PIP3 and additional libraries for geospatial analyses

Make sure that PyGeos and tkinter are available for use with geopandas:

```
sudo apt install python3-pip
sudo apt-get install python3-tk
sudo apt install tk8.6-dev
sudo apt install libgeos-dev
```

Then install *QGIS* and `GDAL` for *Linux* (this should work with any *Debian* architecture):

```
sudo add-apt-repository ppa:ubuntugis/ppa && sudo apt-get update
sudo apt-get update
sudo apt-get install gdal-bin
sudo apt-get install libgdal-dev
export CPLUS_INCLUDE_PATH=/usr/include/gdal
export C_INCLUDE_PATH=/usr/include/gdal
pip3 install GDAL
```

**Note:** Check on the latest GDAL release on the GDAL website.

More guidance for installing GDAL (also on other platforms) is available at gdal.org.

### 1.2.5 Install an IDE (*PyCharm*)

**Note:** IDE - your choice Any other Python IDE is also OK for working with *hylas*. Setting up *PyCharm* is explained here as just one option for working with *flusstools*.

Install *PyCharm* with snap (requires snap):

```
sudo apt install snapd
sudo snap install pycharm-community --classic
```

## 1.3 Install flusstools

### 1.3.1 conda

1. Download our environment.yml file and save it in a temporary folder (e.g., *C:temp* or *USER/Downloads/*).
2. Open *Anaconda Prompt* (on *Windows*) or *Terminal* (on *Linux*).
3. Navigate to your download directory (e.g., `cd C:\temp` or `cd Downloads/`).
4. Install the *flusstools* environment:
   - `conda env create -f environment.yml`
   - Geospatial libraries and other dependencies (see below) are being installed in a new environment called *flusstools* - this may take a while ...
   - Read more about installing, managing, or removing *conda* environments on hydro-informatics.github.io (install).
5. Activate the *flusstools* environment:
   - `conda activate flusstools`
6. Install *flusstools* in the new environments (yes, use `pip` in *conda*):
   - `pip install flusstools`

### 1.3.2 pip / venv

Consider to create and activate a new virtual environment before installing *flusstools* requirements (read more at python.org). Then, download our requirements.txt file and save it in a temporary folder (e.g., *C:temp* or *USER/Downloads/*). In *Terminal* (*Linux / macOS*) or Windows Command Prompt type:

```
cd C:\temp # or cd Downloads/
pip install -r requirements.txt
pip install flusstools
```

## 1.4 Setup *IDE* environment

Depending on the *IDE* you are using, create a new project and define the above created environment (either *conda* or *pip*) as project interpreter.

- *PyCharm* users get help at jetbrains.com
- *Spyder* users find help at spyder-ide.org
- *Notebook* users are served at jupyter.org

## 1.5 Basic Usage

1. Run *Python* and add the download directory of `flusstools` to the system path:

```
import os, sys
sys.path.append("D:/Target/Directory/flusstools/")  # Of course: replace "D:/Target/
↪Directory/", e.g., with  r'' + os.path.abspath('')
```

2. Import `flusstools`:

```
import flusstools as ft
```

```
from flusstools import geotools as gt
raster, array, geo_transform = gt.raster2array("/sample-data/froude.tif")
type(raster)
<class 'osgeo.gdal.Dataset'>
type(array)
<class 'numpy.ndarray'>
type(geo_transform)
<class 'tuple'>
print(geo_transform)
(6748604.7742, 3.0, 0.0, 2207317.1771, 0.0, -3.0)
```

## 1.6 Requirements

*FlussTools* requires geo-spatial processing libraries, which cannot be directly resolved by running *setup.py*. For this reason, we recommend to either install a virtual environment (*pip / venv*) with requirements.txt (*pip / venv*) or a conda environment (*conda*) with environment.yml (*conda*) to check out the following dependencies:

- pip
- tabulate
- numpy
- platform
- pandas
- matplotlib
- plotly
- alphashape
- earthpy

- gdal
- geopandas
- geojson
- laspy
- mapclassify
- pyshp
- rasterio
- rasterstats
- shapely
- tk

CHAPTER

TWO

# GEOTOOLS

**Geospatial Functions for Hydraulics and Morphodynamics**

`geotools` provides *Python3* functions for many sorts of river-related analyses with geospatial data. The package is intended as support material for the lecture Python programming for Water Resources Engineering and Research, where primarily *conda* environments are used.

**Note:** This documentation is also as available as style-adapted PDF (download).

## 2.1 Usage

### 2.1.1 Import

Import `geotools` from *flusstools*:

```
from flusstools import geotools as gt
```

### 2.1.2 Example (code block)

```
from flusstools import geotools as gt
raster, array, geo_transform = gt.raster2array("/sample-data/froude.tif")
type(raster)
# >>> <class 'osgeo.gdal.Dataset'>
type(array)
# >>> <class 'numpy.ndarray'>
type(geo_transform)
# >>> <class 'tuple'>
print(geo_transform)
# >>> (6748604.7742, 3.0, 0.0, 2207317.1771, 0.0, -3.0)
```

### 2.1.3 Example (showcase)

A showcase is provided with the *ROOT/examples/geotools-showcase/georeference_tifs.py* script that illustrates georeferencing *tif* images that do not have a projection assigned.

## 2.2 Code structure

The following diagram highlights function locations in *Python* scripts and how those are linked to each other.

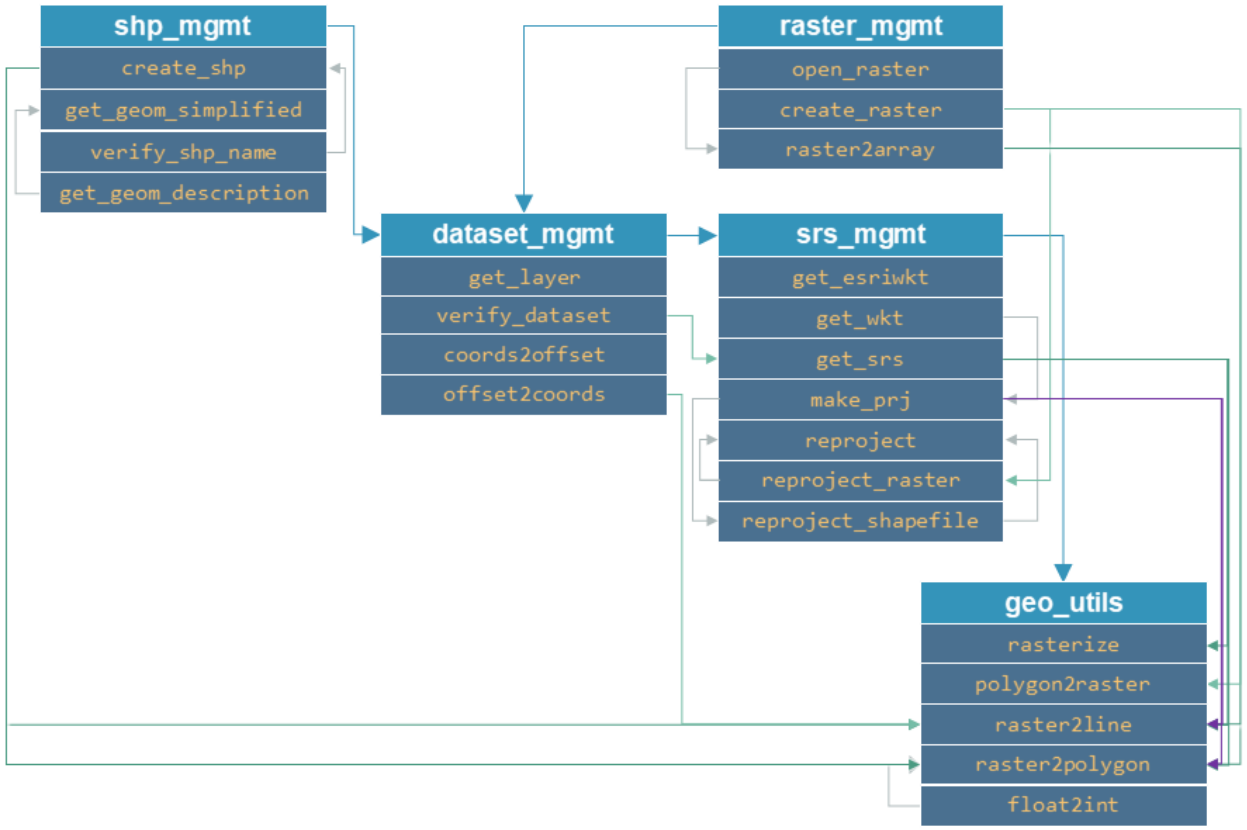


Fig. 2.1: *Diagram of the code structure (needs to be updated).*

## 2.3 Script and function docs

### 2.3.1 `geotools` (MASTER)

`geo_utils` is a package for creating, modifying, and transforming geo-spatial datasets. A detailed documentation of `geo_utils` is available at geo-utils.readthedocs.io.

`flusstools.geotools.geotools.`**`float2int`**(*raster_file_name*, *band_number=1*)

Converts a float number raster to an integer raster (required for converting a raster to a polygon shapefile).

**Parameters**

- **raster_file_name** (*str*) – Target file name, including directory; must end on ".tif".
- **band_number** (*int*) – The raster band number to open (default: 1).

**Returns** "path/to/ew_raster_file.tif"

**Return type** str

flusstools.geotools.geotools.**raster2line**(*raster_file_name*, *out_shp_fn*, *pixel_value*)

Converts a raster to a line shapefile, where pixel_value determines line start and end points.

**Parameters**

- **raster_file_name** (*str*) – of input raster file name, including directory; must end on ".tif".
- **out_shp_fn** (*str*) – of target shapefile name, including directory; must end on ".shp".
- **pixel_value** – Pixel values to connect.

flusstools.geotools.geotools.**raster2polygon**(*file_name*, *out_shp_fn*, *band_number=1*, *field_name='values'*)

Converts a raster to a polygon shapefile.

**Parameters**

- **file_name** (*str*) – Target file name, including directory; must end on ".tif"
- **out_shp_fn** (*str*) – Shapefile name (with directory e.g., "C:/temp/poly.shp")
- **band_number** (*int*) – Raster band number to open (default: 1)
- **field_name** (*str*) – Field name where raster pixel values will be stored (default: "values")
- **add_area** – If True, an "area" field will be added, where the area in the shapefiles unit system is calculated (default: False)

flusstools.geotools.geotools.**rasterize**(*in_shp_file_name*, *out_raster_file_name*, *pixel_size=10*, *no_data_value=-9999*, *rdtype=gdal.GDT_Float32*, *overwrite=True*, *interpolate_gap_pixels=False*, ***kwargs*)

Converts any ESRI shapefile to a raster.

**Parameters**

- **in_shp_file_name** (*str*) – A shapefile name (with directory e.g., "C:/temp/poly.shp")
- **out_raster_file_name** (*str*) – Target file name, including directory; must end on ".tif"
- **pixel_size** (*float*) – Pixel size as multiple of length units defined in the spatial reference (default: 10)
- **no_data_value** (*int OR float*) – Numeric value for no-data pixels (default: -9999)
- **rdtype** (*gdal.GDALDataType*) – The raster data type (default: gdal.GDT_Float32 (32 bit floating point)
- **overwrite** (*bool*) – Overwrite existing files (default: True)
- **interpolate_gap_pixels** (*bool*) – Fill empty pixels that are not touched by a shapefile element with interpolated values (default: False)

**Keyword Arguments**

- **field_name** (*str*) – Name of the shapefile's field with values to burn to raster pixel values.
- **radius1** (*float*) – Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **radius2** (*float*) – Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **power** (*float*) – Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).
- **smoothing** (*float*) – Smoothing parameter for interpolating pixel values (default: `0.0`).
- **min_points** (*int*) – Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).
- **max_points** (*int*) – Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

**Hints:** More information on pixel value interpolation: * `interpolate_gap_pixels=True` interpolates values at pixels that are not touched by any las point. * The pixel value interpolation uses `gdal_grid` (i.e., its Python bindings through `gdal.Grid()`). * Control the interpolation parameters with the keyword arguments `radius1`, `radius2`, `power`, `max_points`, `min_points`, and `smoothing`..

**Returns** Creates the GeoTIFF raster defined with `out_raster_file_name` (success: `0`, otherwise `None`).

**Return type** int

### 2.3.2 raster_mgmt raster management

flusstools.geotools.raster_mgmt.**clip_raster**(*polygon*, *in_raster*, *out_raster*)

Clips a raster to a polygon.

**Parameters**

- **polygon** (*str*) – A polygon shapefile name, including directory; must end on `".shp"`.
- **in_raster** (*str*) – Name of the raster to be clipped, including its directory.
- **out_raster** (*str*) – Name of the target raster, including its directory.

**Returns** Creates a new, clipped raster defined with `out_raster`.

**Return type** None

flusstools.geotools.raster_mgmt.**create_raster**(*file_name*, *raster_array*, *bands=1*, *origin=None*, *epsg=4326*, *pixel_width=10.0*, *pixel_height=10.0*, *nan_val=-9999.0*, *rdtype=gdal.GDT_Float32*, *geo_info=False*, *rotation_angle=None*, *shear_pixels=True*, *options=['PROFILE=GeoTIFF']*)

Converts an `ndarray` (`numpy.array`) to a GeoTIFF raster.

**Parameters**

- **file_name** (*str*) – Target file name, including directory; must end on ".tif".
- **raster_array** (ndarray or list) – Array or list of arrays of values to rasterize. If a list of arrays is provided, the length of the list will correspond to the number of bands added to the raster (supersedes bands).
- **bands** (*int*) – Number of bands to write to the raster (default: 1).
- **origin** (*tuple*) – Coordinates (x, y) of the origin.
- **epsg** (*int*) – EPSG:XXXX projection to use (default: 4326).
- **pixel_height** (*float OR int*) – Pixel height as multiple of the base units defined with the EPSG number (default: 10 meters).
- **pixel_width** (*float OR int*) – Pixel width as multiple of the base units defined with the EPSG number (default: 10 meters).
- **nan_val** (int or float) – No-data value to be used in the raster. Replaces non-numeric and np.nan in the ndarray. (default: geoconfig.nan_value).
- **rdtype** – gdal.GDALDataType raster data type (default: gdal.GDT_Float32 (32 bit floating point).
- **geo_info** (*tuple*) – Defines a gdal.DataSet.GetGeoTransform object and supersedes origin, pixel_width, pixel_height (default: False).
- **rotation_angle** (*float*) – Rotate (in degrees) not North-up rasters. The default value (0) corresponds to north-up (only modify if you know what you are doing).
- **shear_pixels** (*bool*) – Use with rotation_angle to shear pixels as well (default: True).
- **options** (*list*) – Raster creation options - default is ['PROFILE=GeoTIFF']. Add 'PHOTOMETRIC=RGB' to create an RGB image raster.

**Returns** 0 if successful, otherwise -1.

**Return type** int

---

**Hint:** For processing airborne imagery, the roation_angle corresponds to the bearing angle of the aircraft with reference to true, not magnetic North.

---

flusstools.geotools.raster_mgmt.**open_raster**(*file_name*, *band_number=1*)

Opens a raster file and accesses its bands.

**Parameters**

- **file_name** (*str*) – The raster file directory and name.
- **band_number** (*int*) – The Raster band number to open (default: 1).

**Returns** A raster dataset a Python object. osgeo.gdal.Band: The defined raster band as Python object.

**Return type** osgeo.gdal.Dataset

flusstools.geotools.raster_mgmt.**raster2array**(*file_name*, *band_number=1*)

Extracts an ndarray from a raster.

**Parameters**

- **file_name** (*str*) – Target file name, including directory; must end on ".tif".
- **band_number** (*int*) – The raster band number to open (default: 1).

**Returns** Indicated raster band, where no-data values are replaced with np.nan. GeoTransform: The GeoTransformation used in the original raster.

**Return type** ndarray

flusstools.geotools.raster_mgmt.**remove_tif**(*file_name*)
Removes a GeoTIFF and its dependent files (e.g., xml).

**Parameters** **file_name** (*str*) – Directory (path) and name of a GeoTIFF

**Returns** Removes the provided file_name and all dependencies.

flusstools.geotools.raster_mgmt.**raster2array**(*file_name*, *band_number=1*)
Extracts an ndarray from a raster.

**Parameters**

- **file_name** (*str*) – Target file name, including directory; must end on ".tif".
- **band_number** (*int*) – The raster band number to open (default: 1).

**Returns** Indicated raster band, where no-data values are replaced with np.nan. GeoTransform: The GeoTransformation used in the original raster.

**Return type** ndarray

### 2.3.3 shp_mgmt shapefile management

flusstools.geotools.shp_mgmt.**create_shp**(*shp_file_dir*, *overwrite=True*, **args*, ***kwargs*)
Creates a new shapefile with an optionally defined geometry type.

**Parameters**

- **shp_file_dir** (*str*) – of the (relative) shapefile directory (ends on ".shp").
- **overwrite** (*bool*) – If True (default), existing files are overwritten.
- **layer_name** (*str*) – The layer name to be created. If None: no layer will be created.
- **layer_type** (*str*) – Either "point", "line", or "polygon" of the layer_name. If None: no layer will be created.

**Returns** An ogr shapefile

**Return type** osgeo.ogr.DataSource

flusstools.geotools.shp_mgmt.**get_geom_description**(*layer*)
Gets the WKB Geometry Type as string from a shapefile layer.

**Parameters** **layer** (*osgeo.ogr.Layer*) – A shapefile layer.

**Returns** WKB (binary) geometry type

**Return type** str

flusstools.geotools.shp_mgmt.**get_geom_simplified**(*layer*)

**Gets a simplified geometry description (either point, line, or polygon) as a function of** the WKB Geometry Type of a shapefile layer.

**Parameters** **layer** (*osgeo.ogr.Layer*) – A shapefile layer.

**Returns** Either WKT-formatted point, line, or polygon (or unknown if invalid layer).

**Return type** str

flusstools.geotools.shp_mgmt.**polygon_from_shapepoints**(*shapepoints*, *polygon*, *alpha=nan*)

Creates a polygon around a cloud of `shapepoints`.

**Parameters**

- **shapepoints** (*str*) – Point shapefile name, including its directory.
- **polygon** (*str*) – Target shapefile filename, including its directory.
- **alpha** (*float*) – Coefficient to adjust; the lower it is, the more slim will be the polygon.

**Returns** Creates the polygon shapefile defined with `polygon`.

**Return type** None

flusstools.geotools.shp_mgmt.**verify_shp_name**(*shp_file_name*, *shorten_to=13*)

Ensure that the shapefile name does not exceed 13 characters. Otherwise, the function shortens the `shp_file_name` length to N characters.

**Parameters**

- **shp_file_name** (*str*) – A shapefile name (with directory e.g., `"C:/temp/poly.shp"`).
- **shorten_to** (*int*) – The number of characters the shapefile name should have (default: `13`).

**Returns** A shapefile name (including path if provided) with a length of `shorten_to`.

**Return type** str

### 2.3.4 `srs_mgmt` projection management

flusstools.geotools.srs_mgmt.**get_esriwkt**(*epsg*)

Gets esriwkt-formatted spatial references with epsg code online.

**Parameters** **epsg** (*int*) – EPSG Authority Code

**Returns** An esriwkt string (if an error occur, the default epsg=``4326`` is used).

**Return type** str

#### Example

```
get_esriwkt(4326)
```

flusstools.geotools.srs_mgmt.**get_srs**(*dataset*)

Gets the spatial reference of any `gdal.Dataset`.

**Parameters** **dataset** (*gdal.Dataset*) – A shapefile or raster.

**Returns** A spatial reference object.

**Return type** osr.SpatialReference

flusstools.geotools.srs_mgmt.**get_wkt**(*epsg*, *wkt_format='esriwkt'*)

Gets WKT-formatted projection information for an epsg code using the `osr` library.

**Parameters**

- **epsg** (*int*) – epsg Authority code
- **wkt_format** (*str*) – of wkt format (default is esriwkt for shapefile projections)

**Returns** WKT (if error: returns default corresponding to `epsg=4326`).

**Return type** str

`flusstools.geotools.srs_mgmt.`**make_prj**(*shp_file_name*, *epsg*)

Generates a projection file for a shapefile.

**Parameters**

- **shp_file_name** (*str*) – of a shapefile name (with directory e.g., `"C:/temp/poly.shp"`).
- **epsg** (*int*) – EPSG Authority Code

**Returns** Creates a projection file (`.prj`) in the same directory and with the same name of `shp_file_name`.

`flusstools.geotools.srs_mgmt.`**reproject**(*source_dataset*, *new_projection_dataset*)

Re-projects a dataset (raster or shapefile) onto the spatial reference system of a (shapefile or raster) layer.

**Parameters**

- **source_dataset** (*gdal.Dataset*) – Shapefile or raster.
- **new_projection_dataset** (*gdal.Dataset*) – Shapefile or raster with new projection info.

**Returns**

- If the source is a raster, the function creates a GeoTIFF in same directory as `source_dataset` with a `"_reprojected"` suffix in the file name.
- If the source is a shapefile, the function creates a shapefile in same directory as `source_dataset` with a `"_reprojected"` suffix in the file name.

`flusstools.geotools.srs_mgmt.`**reproject_raster**(*source_dataset*, *source_srs*, *target_srs*)

Re-projects a raster dataset. This function is called by the `reproject` function.

**Parameters**

- **source_dataset** (*osgeo.ogr.DataSource*) – Instantiates with an `ogr.Open(SHP-FILE)`.
- **source_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(source_dataset)`
- **target_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(DATASET-WITH-TARGET-PROJECTION)`.

**Returns** Creates a new GeoTIFF raster in the same directory where `source_dataset` lives.

`flusstools.geotools.srs_mgmt.`**reproject_shapefile**(*source_dataset*, *source_layer*, *source_srs*, *target_srs*)

Re-projects a shapefile dataset. This function is called by the `reproject` function.

**Parameters**

- **source_dataset** (*osgeo.ogr.DataSource*) – Instantiates with `ogr.Open(SHP-FILE)`.
- **source_layer** (*osgeo.ogr.Layer*) – Instantiates with `source_dataset.GetLayer()`.

- **source_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(source_dataset)`.
- **target_srs** (*osgeo.osr.SpatialReference*) – Instantiates with `get_srs(DATASET-WITH-TARGET-PROJECTION)`.

**Returns** Creates a new shapefile in the same directory where `source_dataset` lives.

### 2.3.5 `dataset_mgmt` dataset conversion

`flusstools.geotools.dataset_mgmt.`**`coords2offset`**(*geo_transform*, *x_coord*, *y_coord*)

Returns x-y pixel offset (inverse of the `offset2coords` function).

**Parameters**

- **geo_transform** – osgeo.gdal.Dataset.GetGeoTransform() object
- **x_coord** (*float*) – x-coordinate
- **y_coord** (*float*) – y-coordinate

**Returns** Number of pixels `(offset_x, offset_y)`, both `int` .

**Return type** tuple

`flusstools.geotools.dataset_mgmt.`**`get_layer`**(*dataset*, *band_number=1*)

Gets a `layer=band` (`RasterDataSet`) or `layer=ogr.Dataset.Layer` of any dataset.

**Parameters**

- **dataset** (`osgeo.gdal.Dataset` or `osgeo.ogr.DataSource`) – Either a raster or a shapefile.
- **band_number** (*int*) – Only use with rasters to define a band number to open (default=``1``).

**Returns** `{GEO-TYPE: if raster: raster_band, if vector: GetLayer(), else: None}`

**Return type** dict

`flusstools.geotools.dataset_mgmt.`**`offset2coords`**(*geo_transform*, *offset_x*, *offset_y*)

Returns x-y coordinates from pixel offset (inverse of `coords2offset` function).

**Parameters**

- **geo_transform** (*osgeo.gdal.Dataset.GetGeoTransform*) – The geo transformation to use.
- **offset_x** (*int*) – x number of pixels.
- **offset_y** (*int*) – y number of pixels.

**Returns** Two `float` numbers of x-y-coordinates `(x_coord, y_coord)`.

**Return type** tuple

`flusstools.geotools.dataset_mgmt.`**`verify_dataset`**(*dataset*)

Verifies if a dataset contains raster or vector data.

**Parameters** **dataset** (`osgeo.gdal.Dataset` or `osgeo.ogr.DataSource`) – Dataset to verify.

**Returns** Either “mixed”, “raster”, or “vector”.

**Return type** string

### 2.3.6 KML/KML file management

Modified script (original: Linwood Creekmore III)

#### Examples

# output to geopandas dataframe (gdf) gdf = kmx2other("my-places.kmz", output="gpd")

# plot the new gdf (use %matplotlib inline in notebooks) gdf.plot()

# convert a kml-file to a shapefile success = kmx2other("my-places.kml", output="shp")

`flusstools.geotools.kml.`**`kmx2other`**(*file*, *output='df'*)

Converts a Keyhole Markup Language Zipped (KMZ) or KML file to a pandas dataframe, geopandas geodataframe, csv, geojson, or ESRI shapefile.

**Parameters**

- **`file`** (*str*) – The path to a KMZ or KML file.
- **`output`** (*str*) – Defines the output type. Valid options are: `"shapefile"`, `"shp"`, `"shapefile"`, or `"ESRI Shapefile"`.

---

**Hint:** The core function is taken from http://programmingadvent.blogspot.com/2013/06/kmzkml-file-parsing-with-python.html

---

**Returns** object

**Return type** self

Original Classes written by Linwood Creekmore III (modified for geo_utils)

With code blocks from:

- http://programmingadvent.blogspot.com/2013/06/kmzkml-file-parsing-with-python.html
- http://gis.stackexchange.com/questions/159681/geopandas-cant-save-geojson
- https://gist.github.com/mciantyre/32ff2c2d5cd9515c1ee7

**class** `flusstools.geotools.kmx_parser.`**`ModHTMLParser`**

A child of HTMLParser, tailored (modified) for kml/kmy parsing.

**`handle_data`**(*data*)

Generates mapping and series if `in_table` is `True`.

**Parameters** **`data`** (*str*) – Text lines of data divided by colons.

**`handle_starttag`**(*tag*, *attrs*)

Enables a table if a table-tag is provided.

**Parameters**

- **`tag`** (*str*) – Set to "table" for enabling usage of a table.
- **`attrs`** (*list*) – List of additional attributes (currently unused).

**class** `flusstools.geotools.kmx_parser.`**`PlacemarkHandler`**

Child of `xml.sax.handler.ContentHandler`, tailored for handling kml files.

**`characters`**(*data*)

Adds a line of data to the read-buffer.

**Parameters** **data** (*str*) –

**endElement**(*name*)
Sets the end (last) element.

**Parameters** **name** (*str*) –

**htmlizer**()
Creates an html file.

**spatializer**()
Converts string objects to spatial Python objects.

**Parameters** **row** (*df*) – List of strings for conversion

**startElement**(*name*, *attributes*)
Looks for the first Placemark element in a kml file.

**Parameters**

- **name** (*str*) – Name-tag of the element
- **attributes** (*str*) –

CHAPTER

# THREE

# FUZZYCORR

This repository contains the work developed for a Master Thesis on fuzzy map comparison methods to evaluate the performance of hydro-morphodynamic numerical models. Please read the License terms for code usage and re-distribution.

Sediment transport and hydraulic processes can be reproduced with numerical models such as SSIIMM, Hydro_AS_2D, TELEMAC and many more. The accuracy of numerical models is assessed through comparing the simulated and the observed datasets, which constitutes a model validation. With the purpose of analyzing simulated and observed bed elevation change, two methods of comparison can be applied:

1. Comparison via statistical methods such as RMSE (Root Mean Squared Error) or visual human comparison. However, local measures of similarity (or a similarity) like the RMSE are very sensible to uncertainty of location and amount, thus indicating low agreement even when overall patterns were adequately simulated.
2. Visual comparison captures global similarity, which is one of the reasons why modelers often use it for model validation. Humans are capable of finding patterns without deliberately trying, and therefore, this type of comparison provides substantial advantages over local similarity measures. Nevertheless, more research has to be done to implement automated validation tools that emulate human thinking. This is necessary because human comparison is not transparent, prone to subjective interpretations, time consuming, and hardly reproducible.

In this context, the concept of fuzzy set theory has capacities to consider similarity of spatial pattern analogous to human thinking. For instance, fuzziness of location introduces a tolerance regarding spatial uncertainty in the results of hydro-morphodynamic models. To this end, fuzzy logic enables an objective validation of such models by overcoming uncertainties in the model structure, parameters and input data.

The algorithms provided with `fuzzycorr` address the necessity in evaluating (or validating) model performance through the use of fuzzy map comparison. Future developments aim to go beyond a one-way validation towards a two-way communication between the validation algorithms and the models. The two-way communication represents a feedback loop that will eventually enable an automated calibration of numerical hydro-morphodynamic models.

## 3.1 Usage

### 3.1.1 Basics

The following code block exemplifies the usage of *fuzzycorr* to explore the fuzzy correlation between two (e.g., observed and modeled) maps (in *GeoTIFF* format):

```
from flusstools import fuzzycorr as fc
```

### 3.1.2 Example (showcase)

The best way to learn the usage is by examples. In the directory `examples`, the usage of the modules are demonstrated in a case study. Inside the folder `salzach_case`, the results from a hydro-morphodynamic numerical simulation ( i.e., simulated bed elevation change, deltaZ) are located in `raw_data`. For more details on the hydro-morphodynamic numerical refer to Beckers et al. (2020).

The following showcase scripts live in *ROOT/examples/fuzzycorr-showcase/*:

- `prepro_salzach.py`: example of the usage of the class `FuzzyPreProcessor` of the module `prepro.py`, where vector data is interpolated and rasterized.
- `classification_salzach.py`: example of the usage of the class `PreProCategorization` of the module `prepro.py`.
- `fuzzycomparison_salzach.py`: example of the usage of the class `FuzzyComparison` of the module `fuzzycomp.py`, which creates a correlation (similarity) measure between simulated and observed datasets.
- `plot_salzach.py`, `plot_class_rasters.py` and `performance_salzach`: example of the usage of the module `plotter.py`.
- `random_map`: example of generating a raster following a uniform random distribution, which uses the module `prepro.py`.

## 3.2 Structure

This package contains the following modules, which were designed in *Python 3.6*:

- *prepro.py* includes functions for reading, normalizing and rasterizing vector data. These are preprocessing steps for fuzzy map comparison (module fuzzycomp).
- *fuzzycomp.py* provides routines for fuzzy map comparison in continuous valued rasters. Refer to Hagen (2006) for more details.
- *plotter.py*: Visualization routines for output and input rasters.
- The package documentation is located in the folder *docs*.

### 3.2.1 Pre-processing: prepro.py

**Hints:**

- many class methods could be imported from geotools
- already removed: clip_raster, which is a duplicate of raster_mgmt

**class** flusstools.fuzzycorr.prepro.**CategorizationPreProcessor**(*raster*)
Structured for ... (UNCLEAR)

**Parameters** **raster** – string, path of the raster to be categorized

**categorize_raster**(*class_bins*, *map_out*, *save_ascii=True*)
Classifies the raster according to the classification bins

**Parameters**

- **map_out** – path of the project directory
- **class_bins** – list of floats
- **save_ascii** – bool

**Returns** saves the classified raster in the chosen directory

**nb_classes**(*n_classes*)
Generates class bins based on the Natural Breaks method

**Parameters** **n_classes** – integer, number of classes

**Returns** list of optimized bins

**class** flusstools.fuzzycorr.prepro.**FuzzyPreProcessor**(*df*, *attribute*, *crs*, *nodatavalue*, *res=None*, *ulc=(nan, nan)*, *lrc=(nan, nan)*)
Parent pre-processing structure for the comparison of numeric maps

**Parameters**

- **df** – pandas dataframe, can be obtained by reading the textfile as pandas dataframe
- **attribute** – string, name of the attribute to burn in the raster (ex.: deltaZ, Z)
- **crs** – string, coordinate reference system
- **nodatavalue** – float, value to indicate nodata cells
- **res** – float, resolution of the cell (cell size), is the same for x and y
- **ulc** – tuple of floats, upper left corner coordinate, optional
- **lrc** – tuple of floats, lower right corner coordinate, optional

**array2raster**(*array*, *raster_file*, *save_ascii=True*)
Saves a raster using interpolation

**Parameters**

- **raster_file** – string, path to save the rasterfile
- **save_ascii** – boolean, true to save also an ascii raster

**Returns** saves the raster with the selected filename

---

**Hint:** Function can be moved to geotools/raster_mgmt

---

**create_polygon**(*shape_polygon*, *alpha=nan*)
Creates a polygon surrounding a cloud of shapepoints

**Parameters**

- **shape_polygon** – string, path to save the shapefile
- **alpha** – float, excentricity of the alphashape (polygon) to be created

**Returns** saves the polygon (*.shp) with the selected filename

---

**Hint:** Function can be moved to geotools/shp_mgmt

---

**norm_array**(*method='linear'*)
Normalizes the raw data in equally distanced points depending on the selected resolution

**Returns** interpolated and normalized array with selected resolution

---

**Hint:** Read more at https://github.com/rosskush/skspatial

---

**plain_raster**(*shapefile*, *raster_file*, *res*)
Converts a shapefile(.shp) to a GeoTIFF raster without normalizing

**Parameters**

- **shapefile** – string, filename with path of the input shapefile (*.shp)
- **raster_file** – string, filename with path of the output raster (*.tif)
- **res** – float, resolution of the cell

**Returns** saves the raster in the default directory

**points_to_grid**()
Creates a grid of new points in the target resolution

**Returns** array of size nrow, ncol

**Hints:** Read more at http://chris35wills.github.io/gridding_data/

**random_raster**(*raster_file*, *save_ascii=True*, ***kwargs*)
Creates a raster of randomly generated values

**Keyword Arguments** **minmax** – tuple of floats, (zmin, zmax) min and max ranges for random values

**Returns** array of random values within a range of the same size and chape as the original

### 3.2.2 Fuzzy map comparison core: fuzzycomp.py

Description

**class** flusstools.fuzzycorr.fuzzycomp.**FuzzyComparison**(*raster_a*, *raster_b*, *neigh=4*, *halving_distance=2*)
Performing fuzzy map comparison :param raster_a: string, path of the raster to be compared with rasterB :param raster_b: string, path of the raster to be compared with rasterA :param neigh: integer, neighborhood being considered (number of cells from the central cell), default is 4 :param halving_distance: integer, distance (in cells) to which the membership decays to its half, default is 2

**fuzzy_numerical**(*comparison_name*, *save_dir*, *map_of_comparison=True*)
Compares a pair of raster maps using fuzzy numerical spatial comparison

**Parameters**

- **save_dir** – string, directory where to save the results
- **comparison_name** – string, name of the comparison
- **map_of_comparison** – boolean, create map of comparison in the project directory if True

**Returns** Global Fuzzy Similarity and comparison map

**fuzzy_rmse**(*comparison_name*, *save_dir*, *map_of_comparison=True*)
Compares a pair of raster maps using fuzzy root mean square error as spatial comparison

**Parameters**

- **comparison_name** – string, name of the comparison
- **save_dir** – string, directory where to save the results of the map comparison
- **map_of_comparison** – boolean, if True it creates map of of local squared errors (in the project directory)

**Returns** global fuzzy RMSE and comparison map

**get_neighbours**(*array*, *x*, *y*)
Captures the neighbours and their memberships :param array: array A or B :param x: int, cell in x :param y: int, cell in y :return: np.array (float) membership of the neighbours (without mask), np.array (float) neighbours' cells (without mask)

**save_comparison_raster**(*array_local_measures*, *directory*, *file_name*)
Create map of comparison

**save_results**(*measure*, *directory*, *name*)
Saves a results file

flusstools.fuzzycorr.fuzzycomp.**f_similarity**(*centre_cell*, *neighbours*)
Calculates the similarity function for each pair of values (fuzzy numerical method)

**Parameters**

- **centre_cell** – float, cell under analysis in map A
- **neighbours** – np.array of floats, neighbours in map B

**Returns** np.array of floats, each similarity between each of two cells

flusstools.fuzzycorr.fuzzycomp.**jaccard**(*a*, *b*)
Creates a ...

**Parameters**

- **a** (*float*) –
- **b** (*float*) –

**Returns** jac

**Return type** float

flusstools.fuzzycorr.fuzzycomp.**squared_error**(*centre_cell*, *neighbours*)
Calculates the error measure fuzzy rmse

**Parameters**

- **centre_cell** – float, cell under analysis in map A
- **neighbours** – np.array of floats, neighbours in map B

**Returns** np.array of floats, each similarity between each of two cells

### 3.2.3 Plot routines: plotter.py

Description

**class** flusstools.fuzzycorr.plotter.**RasterDataPlotter**(*path*)
Class of raster for plotting

**Parameters** **path** – string, path of the raster to be plotted

**make_hist**(*legendx*, *legendy*, *fontsize*, *output_file*, *figsize*, *set_ylim=None*, *set_xlim=None*)
Creates a histogram of numerical raster

**Parameters**

- **legendx** – string, legend of the x axis of he histogram
- **legendy** – string, legend of the y axis of he histogram

- **fontsize** – integer, size of the font
- **output_file** – string, path for the output file
- **figsize** – tuple of integers, size of the width x height of the figure
- **set_ylim** – float, set the maximum limit of the y axis
- **set_ylim** – float, set the maximum limit of the x axis

**Returns** saves the figure of the histogram

**plot_categorical_raster**(*output_file*, *labels*, *cmap*, *box=True*)
Creates a figure of a categorical raster

**Parameters**

- **output_file** – path, file path of the figure
- **labels** – list of strings, labels (i.e., titles)for the categories
- **cmap** – string, colormap to plot the raster
- **box** – boolean, if False it sets off the frame of the picture

**Returns** saves the figure of the raster

**plot_categorical_w_window**(*output_file*, *labels*, *cmap*, *xy*, *width*, *height*, *box=True*)
Creates a figure of a categorical raster with a zoomed window

**Parameters**

- **output_file** – path, file path of the figure
- **labels** – list of strings, labels (i.e., titles)for the categories
- **cmap** – string, colormap to plot the raster
- **xy** – tuple (x,y), origin of the zoomed window, the upper left corner
- **width** – integer, width (number of cells) of the zoomed window
- **height** – integer, height (number of cells) of the zoomed window

**Returns** saves the figure of the raster

**plot_continuous_raster**(*output_file*, *cmap*, *vmax=nan*, *vmin=nan*, *box=True*)
Creates a figure of a continuous valued raster

**Parameters**

- **output_file** – path, file path of the figure
- **cmap** – string, colormap to plot the raster
- **vmax** – float, optional, value maximum of the scale, this value is used in the normalization of the colormap
- **vmin** – float, optional, value minimum of the scale, this value is used in the normalization of the colormap
- **box** – boolean, if False it sets off the frame of the picture

**Returns** saves the figure of the raster

**plot_continuous_w_window**(*output_file*, *xy*, *width*, *height*, *bounds*, *cmap=None*, *list_colors=None*)
Create a figure of a raster with a zoomed window :param output_file: path, file path of the figure :param xy: tuple (x,y), origin of the zoomed window, the upper left corner :param width: integer, width (number

of cells) of the zoomed window :param height: integer, height (number of cells) of the zoomed window :param bounds: list of float, limits for each color of the colormap :param cmap: string, optional, colormap to plot the raster :param list_colors: list of colors (str), optional, as alternative to using a colormap :returns: saves the figure of the raster

flusstools.fuzzycorr.plotter.**read_raster**(*raster_path*)

Opens a raster

**Parameters** **raster_path** (*str*) – directory and name of a raster

**Returns** a numpy array of the raster

**Return type** ndarray

## 3.3 References


- Ross Kushnereit
- Chris Wills

CHAPTER

# FOUR

# LIDARTOOLS

The *laspy_X* modules are universal *Python3* scripts, which are completely open-source and can be applied on any platform (*Window*, *Linux*). However, *laspy* may crash with larger *las* files (> 1 GB), and in particular, when the available system memory is small. For these reasons, preferably use the inter-platform and open source *laspy_X* modules, but if you need to deal with large *las* files on a weak system, use the *Windows*-only *lastools*. Note that *lastools* builds on LAStools from rapidlasso. It might be possible to run *LAStools* on *Linux* with wine (not yet tested with *flusstools*).

## 4.1 LasPy

The *laspy_X* modules extract geospatial information from *las* files and convert them to ESRI shapefiles or GeoTIFF rasters. *las* is the typical file format for storing airborne lidar (Light Detection and Ranging) data. The *flusstools laspy_X* modules make use of the inter-platform and open source laspy *Python* package. The currently implemented capacities involve:

- A point shapefile with user-defined point attributes such as *intensity*, *waveform*, or *nir*.
- Digital elevation model (DEM) with user-defined resolution (pixel size).
- *GeoTIFF* rasters with user-defined resolution (pixel size) for any attribute of a *las* file (e.g., *intensity*, *waveform*, or *nir*).

---

**Computation Power and Memory Errors**

*Las* files can be very large and the *laspy_X* modules load entire *las* files in the system memory. A large *las* file (> 1 GB) may result in your system shutting down *Python* because it is *eating* more memory than available. Therefore, consider using *las* file subsets or computers with large memory. Read more about memory errors in the Troubleshooting section (see below **:ref:`memory_error`_**).

---

### 4.1.1 Usage

#### Basics

To convert a *las* file to an ESRI shapefile or GeoTIFF, load *flusstools.lidartools.laspy_main* in Python:

```
import flusstools.lidartools.laspy_main as hylas
las_file_name = "path/to/a/las-file.las"
methods = ["las2shp", "las2tif"]
hylas.process_file(las_file_name, epsg=3857, methods=methods)
```

The above code block defines a `las_file_name` variable and `methods` to be used with `flusstools.lidartools.laspy_main.process_file` (see *LasFile main*). The function accepts many more optional arguments:

Loads a las-file and convert it to another geospatial file format (keyword arguments `**opts`).

**param source_file_name** Full directory of the source file to use with methods * if method="las2*" > provide a las-file name * if method="shp2*" > provide a shapefile name

**type source_file_name** str

**param epsg** Authority code to use (try `hylas.lookup_epsg(las_file_name)` to look up the epsg online).

**type epsg** int

**keyword create_dem** default: False - set to True for creating a digital elevation model (DEM)

**kwtype create_dem** bool

**keyword extract_attributes** Attributes to extract from the las-file available in pattr (config.py)

**kwtype extract_attributes** str

**keyword methods** Enabled list strings are las2shp, las2tif, shp2tif, las2dem

**kwtype methods** *list* [*str*]

**keyword overwrite** Overwrite existing shapefiles and/or GeoTIFFs (default: `True`).

**kwtype overwrite** bool

**keyword pixel_size** Use with *2tif to set the size of pixels relative to base units (pixel_size=5 > 5-m pixels)

**kwtype pixel_size** float

**keyword shapefile_name** Name of the point shapefile to produce with las2*

**kwtype shapefile_name** str

**keyword tif_prefix** Prefix include folder path to use for GeoTiFFs (defined extract_attributes are appended to file name)

**kwtype tif_prefix** str

**keyword interpolate_gap_pixels** Fill empty pixels that are not touched by a shapefile point with interpolated values (default: `True`)

**kwtype interpolate_gap_pixels** bool

**keyword radius1** Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.

**kwtype radius1** float

**keyword radius2** Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.

**kwtype radius2** float

**keyword power** Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).

**kwtype power** float

**keyword smoothing** Smoothing parameter for interpolating pixel values (default: `0.0`).

**kwtype smoothing** float

**keyword min_points** Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).

**kwtype min_points** int

**keyword max_points** Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

**kwtype max_points** int

**returns** `True` if successful, `False` otherwise

**rtype** bool

---

**Note:** The `LasPoint` class (see *LasFile main*) can also be directly called in any script with `laspy_processor.LasPoint`. Have a look at the `laspy_processor.process_file` function (*LasFile main*) to see how an instance of the `LasPoint` class is used.

---

### Application example

The following code block converts a file called *las-example.las* first into a shapefile and then into a *GeoTIFF*. By using the attributes `"aci"`, the `scan_angle` (`a`), the `classification_flags` (`c`), and the `intensity` (`i`) are extracted from the *las* file. Find out more about applicable attributes in the *flusstools.lidartools.laspy_config.wattr* dictionary (see below **:ref:`laspy_config`_**).

```
import flusstools.lidartools.laspy_main as hylas
import os

las_file_name = os.path.abspath("") + "/data/las-example.las"
shp_file_name = os.path.abspath("") + "/data/example.shp"
epsg = 25832
methods = ["las2tif"]
attribs = "aci"
px_size = 2
tif_prefix = os.path.abspath("") + "/data/sub"

hylas.process_file(las_file_name,
                   epsg=epsg,
                   methods=methods,
                   extract_attributes=attribs,
                   pixel_size=px_size,
                   shapefile_name=shp_file_name,
                   tif_prefix=tif_prefix)
```

---

**Note:** The method `las2tif` automatically calls the `las2shp` (`flusstools.lidartools.laspy_processor.LasPoint.export2shp`) method because the GeoTIFF pixel values are extracted from the attribute table of the point shapefile. So `las2shp` is the baseline for any other operation.

---

### 4.1.2 Code Documentation

#### LasFile main

flusstools.lidartools.laspy_main.**lookup_epsg**(*file_name*)

Starts a google search to retrieve information from a file name (or other `str`) with information such as *UTM32*.

**Parameters** **file_name** (*str*) – file name or other string with words separated by "-" or "_"

##### Notes

- This function opens a google search in the default web browser.
- More information about projections, spatial reference systems, and coordinate systems

  can be obtained with the geo_utils package.

**process_file(source_file_name, epsg, **opts)**

Loads a las-file and converts it to another geospatial file format (keyword arguments `**opts`).

Note that this function documentation is currently manually implemented because of *Sphinx* having troubles to look behind decorators.

**Arguments:**

- **source_file_name (`str`): Full directory of the source file to use with methods**
  - if `method="las2*"`: provide a las-file name
  - if `method="shp2*"`: provide a shapefile name
- **epsg** (`int`): Authority code to use (try `hylas.lookup_epsg(las_file_name)` to look up the epsg online).

**Keyword Arguments (`**opts`):**

- **create_dem** (`bool`): Set to True for creating a digital elevation model (DEM - default: `False`)
- **extract_attributes** (`str`): Attributes to extract from the las-file available in `pattr` (`config.py`)
- **methods** (`list` [`str`]): Enabled list strings are `las2shp`, `las2tif`, `shp2tif`, `las2dem`.
- **overwrite** (`bool`): Overwrite existing shapefiles and/or GeoTIFFs (default: `True`).
- **pixel_size** (`float`): Use with *2tif to set the size of pixels relative to base units (`pixel_size=5` indicates 5x5-m pixels)
- **shapefile_name** (`str`): Name of the point shapefile to produce with `las2*`
- **tif_prefix** (`str`): Prefix include folder path to use for GeoTiFFs (defined extract_attributes are appended to file name)
- **interpolate_gap_pixels** (`bool`): Fill empty pixels that are not touched by a shapefile point with interpolated values (default: `True`)
- **radius1** (`float`): Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **radius2** (`float`): Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.

- **power** (`float`): Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).
- **smoothing** (`float`): Smoothing parameter for interpolating pixel values (default: `0.0`).
- **min_points** (`int`): Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).
- **max_points** (`int`): Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

**Returns:** `bool`: `True` if successful, `False` otherwise.

More information on pixel value interpolation: * `interpolate_gap_pixels=True` interpolates values at pixels that are not touched by any las point. * The pixel value interpolation uses `gdal_grid` (i.e., its Python bindings through `gdal.Grid()`). * Control the interpolation parameters with the keyword arguments `radius1`, `radius2`, `power`, `max_points`, `min_points`, and `smoothing`.

**See also:**

All variables are illustratively explained on the GDAL website.

### Las processor

**class** flusstools.lidartools.laspy_processor.**LasPoint**(*las_file_name*, *epsg=3857*, *use_attributes='aciw'*, *overwrite=True*)

Las file container to convert datasets to ESRI point shapefiles and/or GeoTIFFs.

**Parameters**

- **las_file_name** (*str*) – Directory to and name of a las file.
- **epsg** (*int*) – Authority Code - Geodetic Parameter Dataset ID (default: `3857`).
- **overwrite** (*bool*) – Overwrite existing shapefiles and/or GeoTIFFs (default: `True`).
- **use_attributes** (*str*) – Attributes (properties) to use from the las-file available in pattr (config.py). (default: `use_attributes="aciw"`).

**Variables**

- **las_file** (*laspy.file.File*) – A laspy file object
- **attributes** (*str*) – Defined with `use_attributes`
- **epsg** (*int*) – Authority code
- **gdf** (*geopandas.GeoDataFrame*) – geopandas data frame containing all points of the las file with the properties (columns) defined by `use_attributes`
- **offset** (*laspy.file.File()header.offset*) – Offset of las points (auto-read)
- **overwrite** (*bool*) – Enable or disable overwriting existing files (default: `True`)
- **scale** (*laspy.file.File()header.scale*) – Scale of las points relative to the offset (auto-read)
- **shapefile_name** (*str*) – The name and dicrectorty of a point shapefile where all las-file data is stored
- **srs** (*osr.SpatialReference*) – The geo-spatial reference imported from `epsg`

**create_dem**(*target_file_name=''*, *pixel_size=1.0*, ***kwargs*)

Creates a digital elevation model (DEM) in GeoTIFF format from the *las* file points.

**Parameters**

- **target_file_name** (*str*) – A file name including an existing directory where the dem will be created< must end on `.tif`.
- **pixel_size** (*float*) – The size of one pixel relative to the spatial reference system

**Keyword Arguments**

- **src_shp_file_name** (*str*) – Name of a shapefile from which elevation information is to be extracted (default: name of the las-point shapefile)
- **elevation_field_name** (*str*) – Name of the field from which elevation data is to be extracted (default: `"elevation"`)
- **interpolate_gap_pixels** (*bool*) – Fill empty pixels that are not touched by a shapefile point with interpolated values (default: `False`)
- **radius1** (*float*) – Define the x-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **radius2** (*float*) – Define the y-radius for interpolating pixels (default: `-1`, corresponding to infinity). Only applicable `with interpolate_gap_pixels`.
- **power** (*float*) – Power of the function for interpolating pixel values (default: `1.0`, corresponding to linear).
- **smoothing** (*float*) – Smoothing parameter for interpolating pixel values (default: `0.0`).
- **min_points** (*int*) – Minimum number of points to use for interpolation. If the interpolator cannot find at least `min_points` for a pixel, it assigns a `no_data` value to that pixel (default: `0`).
- **max_points** (*int*) – Maximum number of points to use for interpolation. The interpolator will not use more than `max_points` closest points to interpolate a pixel value (default: `0`).

---

**Hint:** This function works independently and does not require the prior creation of a shapefile.

---

**Returns** `0` if successful, otherwise `-1`

**Return type** int

**export2shp**(***kwargs*)

Converts las file points to a point shapefile.

**Keyword Arguments** **shapefile_name** (*str*) – Optional shapefile name (must end on .shp). (default: `'/this/dir/las_file_name.shp'`).

**Returns** `/path/to/shapefile.shp`, which is a point shapefile created by the function.

**Return type** str

**get_file_info**()

Prints las file information to console.

#### Analysis config

This is the hylas config file

flusstools.lidartools.laspy_config.**pattr = {'C': 'classification', 'G': 'gps_time', 'N': 'r**

dict of attributes to extract data layers (shapefile columns or multiple GeoTIFFs) from a las file.

All attributes defined in `pattr.values()` must be an attribute of a las_file object. Print all available las file attributes with:

```
print(dir(LasPoint.las_file))
```

**Type** dict

flusstools.lidartools.laspy_config.**wattr = {'C': 'Class', 'G': 'GPStime', 'N': 'NumberRet',**

dict with column headers (shapefile attribute table) and GeoTIFF file names to use for parsing attributes.

**Type** dict

### 4.1.3 Troubleshooting

#### Memory errors

**MemoryError**

**Cause**: *las* files may have a size of several GiB, which may quickly cause a `MemoryError` (e.g., `MemoryError: Unable to allocate 9.1 GiB for an array with shape ...`). In particular, the *Linux* kernel will not attempt to run actions that exceed the commit-able memory.

**Solution: Enable memory over-committing:**

- **Check the current over-commit mode in *Terminal*:** `cat /proc/sys/vm/overcommit_memory`
- If `0` is the answer, the system calculates array dimensions and the required memory (e.g., an array with dimensions `(266838515, 12, 49)` requires a memory of `266838515 * 12 *49 / 1024.0**3` = `146` GiB, which is unlikely to fit in the memory).
- **To enable over-committing, set the commit mode to `1`:** `echo 1 | sudo tee /proc/sys/vm/overcommit_memory`

**Alternative Solution**: Use *LasTools* (see below), which has better capacity to deal with system memory limitations, but works on *Windows* only (not yet tested: implementation of *LasTools* in *Linux* with wine).

## 4.2 LasTools (Windows only)

*lastools* is forked from GCS_scripts by Kenny Larrieu. The original code is designed for *Python2* and the commercial `arcpy` library. The tweaked codes of *las4windows* run with Python 3.8 and work without `arcpy`. This repository only uses the GUI for lidar processing with LASTools.

Because *LASTools* is proprietary, its executables can hardly be run on Linux or other UNIX-based systems (not yet tested: implementation of *LasTools* in *Linux* with wine). This is why *LasTools* is a *Windows*-only application.

**Use the GUI**

Launch `flusstools.lidartools.lastools_GUI.create_gui()` to open a graphical user interface that walks you through the *lastools* scripts implemented in *flusstools*, and calls relevant functions by a simple mouse click.

### 4.2.1 Additional requirements

*LASTools* is used for LiDAR Data Processing and can be downloaded here.

### 4.2.2 Usage

The main function to start processing a *las* or *laz* file with *lastools* is `process_lidar`, which can be called as follows:

```
import flusstools.lidartools.lastools_core as lastools

lastools.process_lidar(
    lastoolsdir="C:/dir/to/LAStools/bin",
    lidardir="C:/LiDAR/file/directory",  # las or laz file
    ground_poly="C:/dir/to/Ground-area-shp-file (optional)",  # limit the analysis␣
↪region
    cores=4,  # numbers of cores to use
    units_code="Meters",  # alternative: "Feet"
    keep_orig_pts=False,  # Keep original ground/veg points (True or False)
    coarse_step="",  # numeric as string (do not use None)
    coarse_bulge="",  # numeric as string (do not use None)
    coarse_spike="",  # numeric as string (do not use None)
    coarse_down_spike="",  # numeric as string (do not use None)
    coarse_offset="",  # numeric as string (do not use None)
    fine_step="",  # numeric as string (do not use None)
    fine_bulge="",  # numeric as string (do not use None)
    fine_spike="",  # numeric as string (do not use None)
    fine_down_spike="",  # numeric as string (do not use None)
    fine_offset=""  # numeric as string (do not use None)
)
```

Alternatively, *lastools* can be started as a graphical user interface as follows (from *Windows Prompt*):

```
cd C:\dir\to\flusstools\lidartools
python LiDAR_processing_GUI
```

### 4.2.3 Code Documentation

#### LiDAR processing

**Things to consider adding:** choice of las or laz output set default values for lasground_new params clip structures step lasclassify params to identify buildings use veg polygon (if given) instead of inverse ground polygon to clip veg points

**class** `flusstools.lidartools.lastools_core.`**DF**(**args: Any*, ***kwargs: Any*)

Extended pandas DataFrame class with an additional title attribute

flusstools.lidartools.lastools_core.**ar1_acorr**(*series*, *maxlags=''*)
Returns lag, autocorrelation, and confidence interval using geometric autocorrelation for AR1 fit of series

flusstools.lidartools.lastools_core.**cmd**(*command*)
Executes command prompt command

flusstools.lidartools.lastools_core.**cox_acorr**(*series*, *maxlags=''*)

**Parameters**

- **series** – (list)
- **maxlags** – (str)

**Returns** two lists (lags and autocorrelation), using Cox variant 3 of ACF

flusstools.lidartools.lastools_core.**ft**(*x*, *y*)
Returns the fourier transform magnitude of the x,y data

flusstools.lidartools.lastools_core.**lof_text**(*pwd*, *src*)
creates a .txt file in pwd (LAStools bin) containing a list of .las/.laz filenames from src directory

flusstools.lidartools.lastools_core.**pd**(*filename*)
returns point density from lasinfo output .txt file

flusstools.lidartools.lastools_core.**pts**(*filename*, *lastoolsdir*)
returns number of points in las file

flusstools.lidartools.lastools_core.**r_confidence_interval**(*r*, *n*, *alpha=0.05*)
Retrieves the confidence interval at the 1-alpha level for correlation of r with n observations when alpha=0.05, it returns the range of possible population correlations at the 95% confidence level so if 0 is not within the bounds, then the correlation is statistically significant at the 95% level

**Parameters**

- **r** – correlation (float)
- **n** – number of observations (int)
- **alpha** – confidence level (float)

**Returns** Confidence interval (low and high) as sequence (list or tuple) of floats.

flusstools.lidartools.lastools_core.**white_noise_acf_ci**(*series*, *maxlags=''*)
Returns the 95% confidence interval for white noise ACF

### File functions

Description

flusstools.lidartools.lastools_fun.**browse**(*root*, *entry*, *select='file'*, *ftypes=[('All files', '*')]*)
GUI button command opens browser window and adds selected file/folder to entry

flusstools.lidartools.lastools_fun.**get_largest**(*directory*)
returns name of largest file in directory

flusstools.lidartools.lastools_fun.**get_las_files**(*directory*)
returns list of all .las/.laz files in directory (at top level)

flusstools.lidartools.lastools_fun.**split_list**(*list2split*, *break_pts*)
returns list l split up into sublists at break point indices

flusstools.lidartools.lastools_fun.**split_reaches**(*list_of_reaches*, *new_reach_pts*)
splits l into sections where new_reach_pts contains the starting indices for each slice

CHAPTER

FIVE

# CONTRIBUTING

## 5.1 Become a contributor

Most team members joined in the framework of their Bachelor or Master's Thesis with innovative contributions. So if you are a student and you want to contribute to *flusstools*, why not in the scope of an innovative thesis? Check out our currently open Bachelor and Master Thesis topics.

Obviously you do not have to be a student to join us - please use Sebastian Schwindt s informal contact form - quick response (most of the time) for sure.

## 5.2 How to document

This package uses *Sphinx* readthedocs and the documentation regenerates automatically after pushing changes to the repositories `main` branch.

To set styles, configure or add extensions to the documentation use `ROOT/.readthedocs.yml` and `ROOT/docs/conf.py`.

Functions and classes are automatically parsed for docstrings and implemented in the documentation. `hylas` docs use google style docstring formats - please familiarize with the style format and strictly apply in all commits.

To modify this documentation file, edit `ROOT/docs/index.rst` (uses reStructuredText format).

In the class or function docstrings use the following section headers:

- `Args` (alias of `Parameters`)
- `Arguments` (alias of `Parameters`)
- `Attention`
- `Attributes`
- `Caution`
- `Danger`
- `Error`
- `Example`
- `Examples`
- `Hint`
- `Important`
- `Keyword Args` (alias of `Keyword Arguments`)

- Keyword Arguments
- Methods
- Note
- Notes
- Other Parameters
- Parameters
- Return (alias of Returns)
- Returns
- Raise (alias of Raises)
- Raises
- References
- See Also
- Tip
- Todo
- Warning
- Warnings (alias of Warning)
- Warn (alias of Warns)
- Warns
- Yield (alias of Yields)
- Yields

For local builds of the documentation, the following packages are required:

```
sudo apt-get install build-essential
sudo apt-get install python-dev python-pip python-setuptools
sudo apt-get install libxml2-dev libxslt1-dev zlib1g-dev
apt-cache search libffi
sudo apt-get install -y libffi-dev
sudo apt-get install python3-dev default-libmysqlclient-dev
sudo apt-get install python3-dev
sudo apt-get install redis-server
```

To generate a local html version of the hylas documentation, cd into the docs directory and type:

```
make html
```

Learn more about *Sphinx* documentation and the automatic generation of *Python* code docs through docstrings in the tutorial provided at github.com/sschwindt/docs-with-sphinx.

## 5.3 Implement new stuff

All contributors, please respect the *Zen of Python* (`import this`).

How to add new package or library imports:

- Add it to the global import management file (*ROOT/import_mgmt.py*) within an *try-except-ImportError* statement (read more).
- If you need to import a library or package that is not yet listed in the *ROOT/environments.yml* and *ROOT/requirements.txt* files, please make sure to add the new library or package in both files.
- Add the new library or package to the `autodoc_mock_imports` *list* in *ROOT/docs/conf.py*.
- Update the version number according to the CONTRIBUTING standards.

Please use *PEP 8* for any code (read more on hydro-informatics.github.io/hypy_pystyle) and try to keep the number of lines per script below 150 (it's hard or even apparently impossible sometimes - just try please).

---

**Important:** Only push debugged code to the main branch - Thank you!

---

CHAPTER

SIX

# DISCLAIMER AND LICENSE

## 6.1 Disclaimer (general)

No warranty is expressed or implied regarding the usefulness or completeness of the information provided for *flusstools* and its documentation. References to commercial products do not imply endorsement by the Authors of *flusstools*. The concepts, materials, and methods used in the codes and described in the docs are for informational purposes only. The Authors have made substantial effort to ensure the accuracy of the code and the docs and the Authors shall not be held liable, nor their employers or funding sponsors, for calculations and/or decisions made on the basis of application of *flusstools*. The information is provided "as is" and anyone who chooses to use the information is responsible for her or his own choices as to what to do with the code, docs, and data and the individual is responsible for the results that follow from their decisions.

## 6.2 BSD 3-Clause License



More information and examples are available in the docs of every *flusstools* module.

# PYTHON MODULE INDEX

## f

# INDEX

## R



## S



## V



## W