FlussTools

Release latest

FlussTeam

May 27, 2026

CONTENTS

The analysis, research, and science-based design of hydrological 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 of processing field data, numerical model outputs, or geospatial data. "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. Because we have a strong commitment to transparent open-source applications, we created *flusstools* and we welcome new team members (for example, to add or amend a module) at any time - read more in the *Contributing* section.

FlussTools has the following sub-packages:

- *bedanalyst* - for plotting and numeric analysis of riverbed characteristic to identify, for instance, clogging (developers: Beatriz Negreiros, and Ricardo Barros).
- *geotools* - versatile functions for processing spatial data for fluvial ecosystem analyses based on gdal and other open source libraries (developers: Kilian Mouris, Beatriz Negreiros, and Sebastian Schwindt). The functions are explained with the geospatial Python tutorials on hydro-informatics.com and the HydroMorphodynamics YouTube channel.
- *fuzzycorr* - a map comparison toolkit that builds on fuzzy sets to assess the accuracy of (numerical) river models (principal developer: Beatriz Negreiros).

How to cite FlussTools

If our codes helped you to accomplish your work, we won't ask you for a coffee, but to cite and spread the utility of our code - Thank you!

```
@software{flussteam2026flusstools,
          author       = {Sebastian Schwindt and
                          Beatriz Negreiros and
                          Ricardo Barros and
                          Niklas Henning and
                          Kilian Mouris},
          title        = {FlussTools},
          year         = 2026,
          publisher    = {GitHub \& Center for Open Science (OSF)},
          version      = {v2.0.1},
          doi          = {10.17605/OSF.IO/G7K52},
          url          = {https://doi.org/10.17605/OSF.IO/G7K52}
        }
```

The documentation is also as available as style-adapted PDF.

CHAPTER

ONE

INSTALLATION

The recommended way to install *flusstools* is inside a **conda/mamba environment** - and this is now the best practice on **every platform** (*Linux*, *Windows*, and *macOS*) alike. The reason is GDAL: *flusstools* builds on the GDAL geospatial library, which is **not** available as a pre-built *pip* wheel. The conda-forge channel ships ready-to-use GDAL binaries, so a conda/mamba environment resolves GDAL (and the rest of the geospatial stack) without any compiler or system-library setup, identically across operating systems.

We recommend mamba - a fast drop-in replacement for `conda` - but every command below works the same if you replace `mamba` with `conda`.

1.1 Quick install

Create an environment with the GDAL binaries from conda-forge, then install *flusstools* into it with *pip*:

```
mamba create -n flussenv -c conda-forge python=3.11 gdal
mamba activate flussenv
pip install flusstools
```

GDAL is provided by conda-forge, and *pip* resolves the remaining dependencies (`numpy`, `geopandas`, `rasterio`, ...) from wheels - nothing has to be compiled.

1.2 Reproducible environment

To recreate the exact, version-pinned environment used to develop and test *flusstools*, build it from the environment.yml that ships with the package:

```
curl -O https://raw.githubusercontent.com/Ecohydraulics/flusstools-pckg/main/environment.
↪yml
mamba env create -f environment.yml
mamba activate flussenv
pip install flusstools
```

1.3 Installing with pip only

A plain `pip install flusstools` (into a regular *virtualenv*) works **only if a matching system GDAL is already present** - i.e. `gdal-config` is on your `PATH` and its version matches the `gdal` Python bindings. Because GDAL publishes no PyPI wheels, *pip* otherwise tries to compile GDAL from source and the install fails. This is why the conda/mamba route above is the platform-independent best practice. For installation within an existing environment, update *pip* first (`python -m pip install --upgrade pip`).

More detailed, step-by-step installation instructions (including how to set up conda/mamba from scratch) are provided with the hydro-informatics eBook (at https://hydro-informatics.com).

CHAPTER

TWO

BASIC USAGE

2.1 Import

Import `flusstools`:

```
import flusstools as ft
```

Or one of its modules:

```
from flusstools import geotools
```

New to Python? Take a look at the Python tutorial for water resources engineering and research at hydro-informatics.com

2.2 Example

```
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)
```

2.3 Requirements (Dependencies)

FlussTools depends on a stack of geospatial libraries - most importantly GDAL, which has no *pip* wheel. As mentioned above (*Installation*), a conda/mamba environment (built from environment.yml) resolves these binaries cleanly and is the recommended setup; a pure-*pip* virtual environment with requirements.txt works only where a system GDAL is available. The full set of third-party dependencies is:

Ext. libs.		
alphashape	pyproj	rasterio
earthpy	mapclassify	rasterstats
gdal	matplotlib	tk
geojson	numpy	scipy
geopandas	pandas	shapely
h5py	openpyxl	tabulate
networkx	pyshp	scikit-fuzzy

CHAPTER

THREE

BEDANALYST

Algorithms for analyzing riverbed clogging through visualization functions, geospatial interpolation, and a novel fuzzy degree of clogging

The *BedAnalyst* (`flusstools.bedanalyst`) modules provide *Python3* functions for substrate sample analysis and a fuzzy logic assessment of so-called riverbed clogging.

3.1 Usage

3.1.1 Import

Import `bedanalyst` from flusstools:

```
from flusstools import bedanalyst as bea
```

3.1.2 Example (code block)

```
from flusstools import bedanalyst as bea
clogging_pars = bea.fuzzy_analyze("/sample-data/sample.csv")
```

3.1.3 Example (showcase)

A showcase is provided in the flusstools example degree of clogging.

3.2 Code structure

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

The modules `cd_profiles`, `nABP_degree_clogging`, and `interp_z2shp` are independent from the `degree_clogging` module.

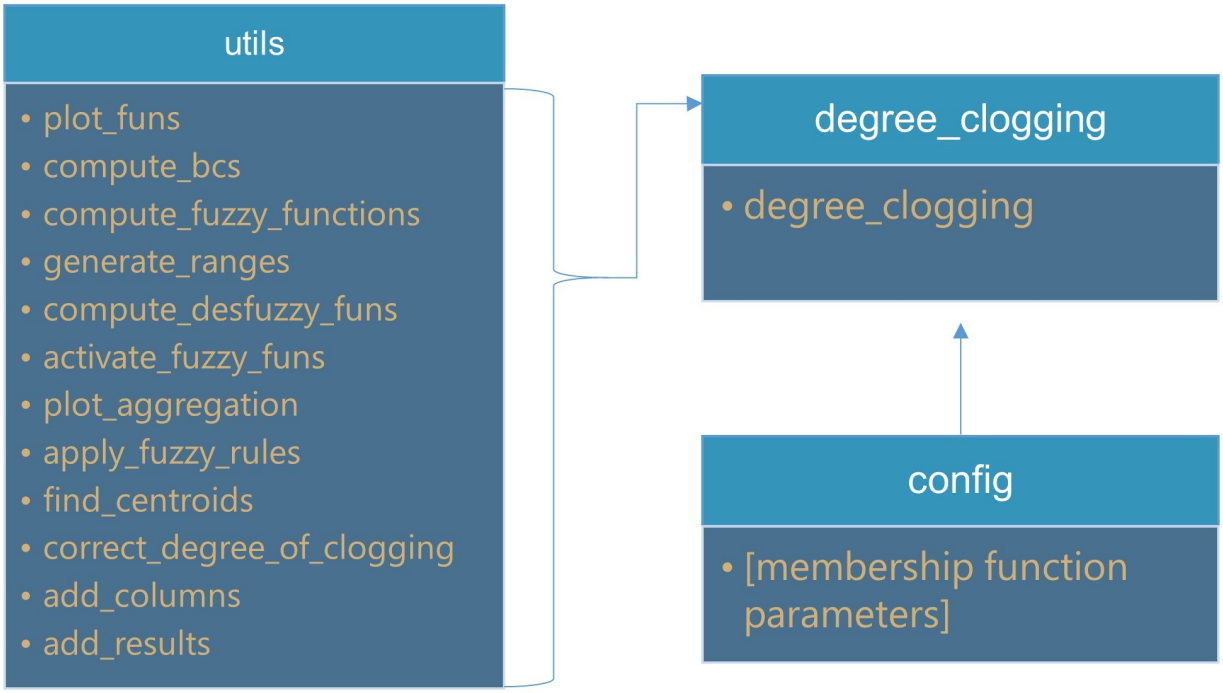
3.3 Script and function docs

3.3.1 Package Head: `bedanalyst`

This module contains all the imported packages (dependencies) and user inputs for running the main script.

The first input is the dictionary containing two points for the fuzzy and defuzzification functions. Two points define a function and the have the following format:

(parameter_values, corresponding_membership_values)



For example, for the parameter fine sediment share, the point with the lowest value in the y-axis is written as follows:

(fs_c_llow, fs_mu_c_llow)

where fs stands for fine sediment share, c indicates the membership function associated to clogging, mu indicates the membership value, and llow indicates the relative position of the point in the curve.

The second input is the local address of the csv file containing the parameters of the samples.

The third input is composed by two booleans that indicates the algorithm to print the fuzzy functions and the aggregation functions inside the folder output.

3.3.2 Another algorithm `degree_clogging`

This module calls the necessary functions to perform the computation of degree of clogging.

`flusstools.bedanalyst.degree_clogging.`**`degree_clogging`**(*df_samples*, *output_csv_path*, *plot=(False, False)*)

Function for computing degree of clogging using the input riverbed parameter along the riverbed depth:

- Fine Sediment Saare/Fraction (fsf/fss): [%]
- Hydraulic Conductivity (kf): [m/s]
- Porosity (n): [-]
- Interstitial Dissolved Oxygen Content (IDOC): [mg/L]
- Bridging criterion according to Huston & Fox (2015; referred as 'ratio' here): [-]

Parameters

- **`df_samples`** (*`pandas.DataFrame`*) – df containing the columns 'id' (sample id), 'fss', 'kf', 'n', 'idoc' and 'ratio'

- **output_csv_path** (*str*) – path to output csv containing the results of the fuzzy inference and final computed degree of clogging
- **plot** (*list*) – List of two boolean objects indicating if the plots for the aggregation function and the fuzzy
- **required** (*membership functions of the above mentione dparameters is required. Should be True if*)
- **respectively.**

Returns
None

3.3.3 Another algorithm interp_z2shp

flusstools.bedanalyst.interp_z2shp.**interp_z2shp**(*df*, *lonlat*, *crs*, *sample_column*, *interp_at_z_stamps*, *new_attr_names*, *meas_at_cols*, *path_shp*)

Interpolates vertical riverbed measurements (e.g., kf, IDOS) to desired z (vertical) stamps, enters them as attributes for creating a point shapefile

Parameters

- **df** (*pandas.DataFrame*) – df with rows indicating samples and columns indicating parameters
- **lonlat** (*tuple of str*) – name of the columns containing longitude and latitude, respectively (x, y)
- **crs** (*str*) – of type 'epsg:xxxx or xxxxx', coordinate system of the input longitude and latitude values
- **sample_column** (*str*) – name of the column in the df which contains the sample names
- **interp_at_z_stamps** (*numpy.array of floats*) – contains the z (vertical) stamps where the measurement should be interpolated
- **new_attr_names** (*list of str*) – names of the attributes referring to the selected new z stamps.
- **meas_at_cols** (*tuple of str*) – contains the column names as a tuple (z_stamp, measurement) of the df that have the vertical spatial stamp of the measurement and the value measured at the corresponding z stamp.
- **path_shp** (*str*) – path to save the shapefile

Returns
geopandas.GeoDataFrame

3.3.4 Another algorithm utils

This module contains the functions called by main.

flusstools.bedanalyst.utils.**activate_fuzzy_funs**(*probe*, *dc_param_range*, *dc_fuzzy_funs*)

Function to compute the membership values of the fuzzy functions for the parameters of a real sample.

Parameters

- **probe** (*tuple*) – float values of the parameters of a sample in the following order (F.S, kf, n, IDOC, ratio)
- **dc_param_range** (*dict*) – arrays of float values of the six parameters defined into utils.generate_ranges()

- **dc_fuzzy_funs** (*dict*) – arrays with the membership values of the fuzzy functions

Returns
fuzzy membership float values corresponding to the parameter values of the real sample

Return type
dc_fuzzy_activated (dict)

flusstools.bedanalyst.utils.**add_columns**(*df_samples*)

Function to add columns of csv output

Parameters
df_samples (*Dataframe*) – dataframe with the parameters of the samples

Returns
same input Dataframe but with new columns

Return type
df_samples (Dataframe)

flusstools.bedanalyst.utils.**add_results**(*df*, *step*, *degree_of_clogging_corrected*, *degree_of_clogging*, *dc_mu_desfuzzy_values*, *dc_af*)

Function to add computed result to output-Dataframe

Parameters

- **df** (*Dataframe*) – input Dataframe
- **step** (*int*) – nth sample
- **degree_of_clogging_corrected** (*float*) – dregree of clogging corrected to interval [0, 1]
- **degree_of_clogging** (*float*) – degree of clogging in the
- **[centroid_of_no_clogging** (*interval*)
- **centroid_of_strong_clogging]**
- **dc_mu_desfuzzy_values** (*dict*) – three float values of the sample-activated defuzzification functions
- **dc_af** (*dict*) – membership float values of the activated fuzzy functions
- **step** – nth sample

Returns
output Dataframe

Return type
df (Dataframe)

flusstools.bedanalyst.utils.**apply_fuzzy_rules**(*dc_af=None*, *dc_desfuzzy_funs=None*, *centroid=False*, *mu_list=None*)

Function to choose membership value of defuzzification functions based on the fuzzy rules.

Parameters

- **dc_af** (*dict*) – membership values of fuzzy functions of a sample
- **dc_desfuzzy_funs** (*dict*) – arrays with the membership values of the defuzzification functions
- **centroid** (*boolean*) – True to correct the limits of degree of clogging to [0,1]

- **mu_list** (*list*) – corrected membership function values of the activated defuzzification functions

Returns
array of membership values that define area under the curve of the defuzzification functions dc_mu_defuzzy_values (dict): three float values of the sample-activated defuzzification functions

Return type
dc_mu_defuzzy (dict)

flusstools.bedanalyst.utils.**compute_bcs**(*dc_limits*)

Function to compute bs and cs constants that define the sigmoid fuzzy membership functions (y = 1 / (1. + exp[-c * (x - b)]))

Parameters
dc_limits (*dict*) – thresholds of the membership functions defined in config.py

Returns
b values of the membership functions dc_c (dict): c values of the membership functions

Return type
dc_b (dict)

flusstools.bedanalyst.utils.**compute_desfuzzy_funs**(*dc_param_range*, *dc_b*, *dc_c*)

Function to compute defuzzification membership functions. The functions for high and low degree of clogging are Sigmoids and for medium degree of clogging is Gaussian.

Parameters
dc_param_range – arrays of float values of the six parameters defined into utils.generate_ranges()

Returns
arrays with the membership values of the defuzzification functions

Return type
(dict)

flusstools.bedanalyst.utils.**compute_fuzzy_functions**(*dc*, *dc_b*, *dc_c*)

Function to compute bs and cs constants that define the sigmoid fuzzy membership functions (y = 1 / (1. + exp[-c * (x - b)]))

Parameters

- **dc_limits** (*dict*) – thresholds of the membership functions defined in config.py
- **dc_b** (*dict*) – b values of the membership functions
- **dc_c** (*dict*) – c values of the membership functions

Returns
arrays with the membership values of the fuzzy functions

Return type
(dict)

flusstools.bedanalyst.utils.**correct_degree_of_clogging**(*dc_defuzzy_centroids*, *degree_of_clogging*)

Function to correct degree of clogging from the interval [centroid_of_no_clogging, centroid_of_strong_clogging] to [0, 1]

Parameters

- **degree_of_clogging** (*float*) – original degree of clogging

- **dc_defuzzy_centroids** (*dict*) – two float values that represent the centroids of sc and nc defuzzi-functions

Returns
degree of clogging in the interval [0, 1]

Return type
degree_of_clogging_corrected (float)

flusstools.bedanalyst.utils.**find_centroids**(*dc_desfuzzy_funs*, *dc_param_range*)

Function to find centroids of the non clogging and strong clogging defuzzification functions

Parameters

- **dc_param_range** (*dict*) – arrays of float values of the six parameters defined into utils.generate_ranges()
- **dc_desfuzzy_funs** (*dict*) – arrays with the membership values of the defuzzification functions

Returns
two float values that represent the centroids of sc and nc defuzzi-functions

Return type
dc_defuzzy_centroids (dict)

flusstools.bedanalyst.utils.**generate_ranges**()

Function to define de ranges of the parameters

Parameters
None

Returns
array with the ranges of the parameters

Return type
(dict)

flusstools.bedanalyst.utils.**plot_aggregation**(*dc_param_range*, *dc_mu_desfuzzy*, *dc_desfuzzy_funs*, *activation*, *degree_of_clogging*, *aggregated*, *step*)

Function to compute the membership values of the fuzzy functions for the parameters of a real sample.

Parameters

- **dc_param_range** (*dict*) – arrays of float values of the six parameters defined into utils.generate_ranges()
- **dc_mu_desfuzzy** (*dict*) – defuzzified membership float values corresponding to the parameter values of the real sample
- **dc_desfuzzy_funs** (*dict*) – arrays with the membership values of the defuzzification functions
- **activation** (*float*) – degree of clogging that equals to the center of mass of tha sum of the areas under
- **functions** (*the activated defuzzification*)
- **degree_of_clogging** (*float*) – degree of clogging correct between 0 and 1 in utils.correct_degree_of_clogging()
- **aggregated** (*array*) – membership float values that define the summed area below the
- **functions.** (*activated defuzzification*)

- **step** (*int*) – integer that represents the nth sample

Returns
None

flusstools.bedanalyst.utils.**plot_funs**(*dc_param_range*, *dc_fuzzy_funs*, *dc_disfuzzy_funs*)

Function to plot the membership functions of the parameters and defuzzyfication membership functions into one plot.

Parameters

- **dc_param_range** (*dict*) – arrays of float values of the six parameters defined into utils.generate_ranges()
- **dc_fuzzy_funs** (*dict*) – arrays of membership fuzzy functions values of the given ranges of the six parameters
- **dc_disfuzzy_funs** (*dict*) – arrays of membership defuzzy functions values of the given ranges of the six parameters

Returns
None

CHAPTER

FOUR

GEOTOOLS

Geospatial Functions for Hydraulics and Morphodynamics

The *GeoTools* (`flusstools.geotools`) modules provide *Python3* functions for many sorts of river-related analyses with geospatial data (e.g., for working with numerical model input and output). The package is intended as support material for the hydro-informatics eBook.

4.1 Usage

4.1.1 Import

Import `geotools` from flusstools:

```
from flusstools import geotools as geo
```

4.1.2 Example (code block)

```
from flusstools import geotools as geo
raster, array, geo_transform = geo.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)
```

4.1.3 Example (showcase)

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

4.2 Code structure

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

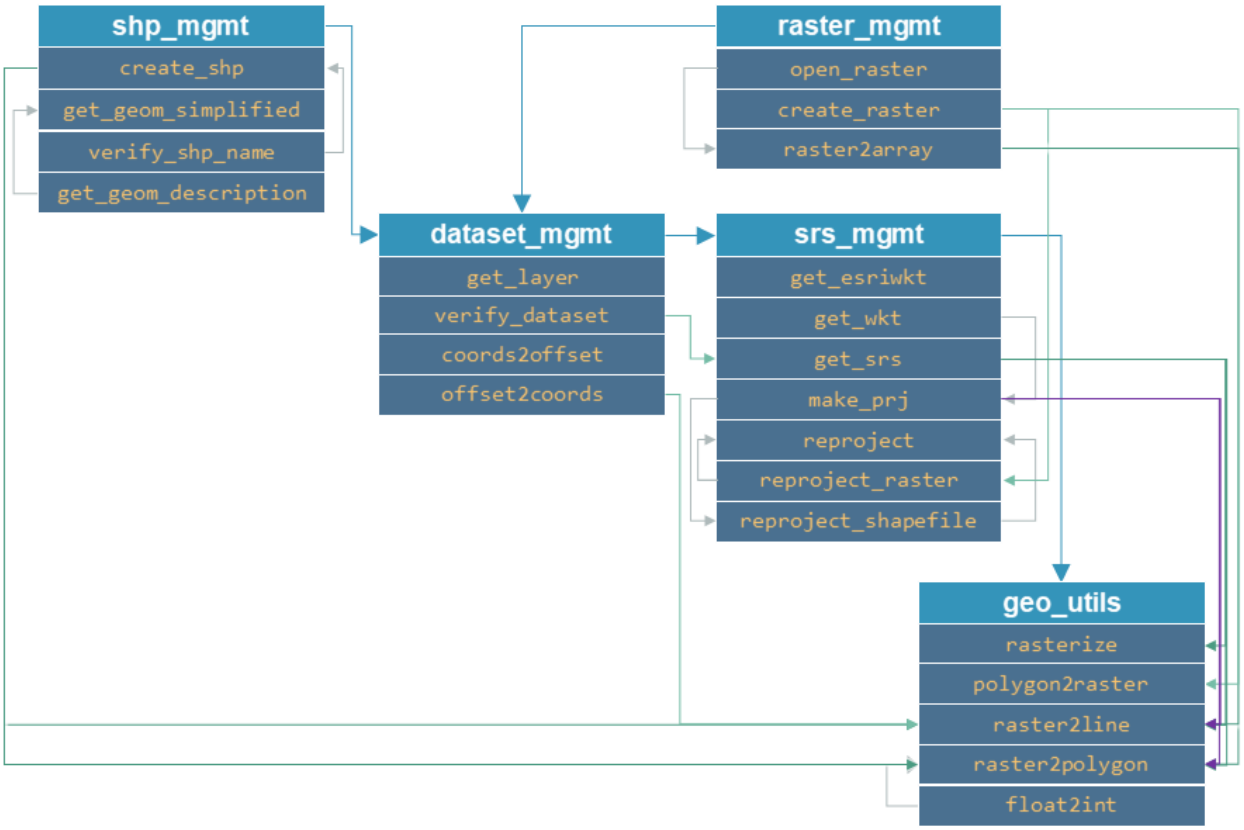


Fig. 4.1: *Diagram of the code structure (needs to be updated).*

4.3 Script and function docs

4.3.1 Package Head: `geotools`

`geotools` is a package for creating, modifying, and transforming geospatial datasets.

`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*, *max_distance_method='simplified'*)

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** (`int` or `float`) – Pixel values to connect.
- **max_distance_method** (*str*) – change to (pixel) `"width"` or `"height"` to force lines to exactly follow pixels (no triangulation).

Returns
Writes a new shapefile to disk.

Return type
None

`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** (*bool*) – If `True`, an "area" field will be added, where the area in the shapefiles unit system is calculated (default: `False`)

Returns
Python object of the provided `out_shp_fn`.

Return type
osgeo.ogr.DataSource

flusstools.geotools.geotools.**rasterize**(*in_shp_file_name*, *out_raster_file_name*, *pixel_size=10*, *no_data_value=-9999*, *rdtype=osgeo.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

4.3.2 Raster Management `raster_mgmt`

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=osgeo.gdal.GDT_Float32*, *geo_info=False*, *rotation_angle=None*, *shear_pixels=True*, *options=None*)

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) – 2d-array (no bands) or list (bands) of 2-darrays of values to rasterize. If a list of 2d-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 `rotation_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 a numpy `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
three-elements of [`osgeo.DataSet` of the raster, `numpy.ndarray` of the raster `band_number` (input) where no-data values are replaced with `np.nan`, `osgeo.GeoTransform` of the original raster]

Return type
list

`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.

Return type
None

`flusstools.geotools.raster_mgmt.`**`xy_raster_shift`**(*file_name*, *x_shift*, *y_shift*, *bands=1*, *rdtype=osgeo.gdal.GDT_Float32*, *nan_val=-9999.0*, *compress=True*, *options=None*, *compress_config=None*)

Creates new geotiff raster with shifts in x and y direction. If enabled compresses it also compresses file.

Args: file_name (string): File path of GeoTiff x_shift (float OR int): Shift origin in x direction. *Check that correct units are used. Example: wgs 84 is in degrees y_shift (float OR int): Shift origin in y direction. *Check

that correct units are used. Example: wgs 84 is in degrees bands (int): Number of bands default is 1, however check raster to see how many are required. Example: RGBA=4 rdtype: gdal.GDALDataType raster data type (default: gdal.GDT_Float32 (32 bit floating point). 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`). compress (Bool): If True creates compressed version of the GeoTiff options (list): Raster creation options - default is ['PROFILE=GeoTIFF']. Add 'PHOTOMETRIC=RGB' to create an RGB image raster. compress_config: (list) Compress creation options - default is ["COMPRESS=LZW", "TILED=YES"] LZW=Lempel-Ziv-Welch-Algorithm See gdal.Translate for more options

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

Return type
int

Hint: For drone rasters try bands=4 (rgba) rdtype=gdal.GDT_Byte nan_val=0 options=['PROFILE=GeoTIFF','PHOTOMETRIC=RGB'])

Bugs: Issues displaying logging

4.3.3 Shapefile Management `shp_mgmt`

`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 (Python object)

Return type
osgeo.ogr.DataSource

> **Hint**
>
> Use the `layer_name` and `layer_type` kwargs along with each other. Keeping these parameters default is deprecated.

`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=numpy.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 `shorten_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

4.3.4 Projection Management `srs_mgmt`

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 occurs, the default epsg=``4326`` is used).

Return type
str

Example

Call this function with `get_esriwkt(4326)` to get a return, such as `'GEOGCS["GCS_WGS_1984",DATUM[...],...]`.

`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`.

Return type
None

`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.

Return type
None

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.

Return type
None

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.

Return type
None

4.3.5 Dataset Management (`dataset_mgmt`)

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 is 1).

Returns
{"type": raster or vector or "None", layer": 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 “unknown”, “raster”, or “vector”.

Return type
str

4.3.6 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

- **(str)** (*output*)
- **(str)**

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

Returns
str

Return type
Success message (use `print(kmx2other(...))` to see what the function did.)

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

Flavored 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.

Returns
Assigns `ModHTMLParser.mapping` and `ModHTMLParser.series` attributes

Return type
None

`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).

Returns
Verifies if the `tag` argument contains the string `"table"`

Return type
None

`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*)

Returns
None

end_element(*name*)

Sets the end (last) element.

Parameters
name (*str*)

Returns
None

htmlizer()

Creates an html file.

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

Returns
An instance of the ModHTMLParser() class

Return type
htmlparser.series

spatializer()

Converts string objects to spatial Python objects.

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

Returns
None

start_element(*name*)

Looks for the first Placemark element in a kml file.

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

Returns
None

4.3.7 Shortest Path Finder

This module is inspired by Michael Diener - read more at
https://github.com/mdiener21/python-geospatial-analysis-cookbook/tree/master/ch08

Example use: `create_shortest_path(shp_file_name, start_node_id, end_node_id)`

flusstools.geotools.shortest_path.**build_graph_from_lines**(*line_shp_name*)

Builds an undirected graph from the line features of a shapefile.

Replaces the removed `networkx.read_shp`: every line feature becomes an edge between its first and last vertex. Edges carry a `distance` weight (summed segment length) and a `Json` attribute holding the full vertex coordinates, as expected by *get_path()*.

Parameters
line_shp_name (*str*) – Input line shapefile name.

Returns
Undirected graph of the line network.

Return type
networkx.Graph

flusstools.geotools.shortest_path.**create_shortest_path**(*line_shp_name*, *start_node_id*, *end_node_id*)

Calculates the shortest path from a network of lines.

Parameters

- **line_shp_name** (*str*) – Input shapefile name
- **start_node_id** (*int*) – Start node ID
- **end_node_id** (*int*) – End node ID

Returns
Creates a graph of nodes (coordinate pairs) connecting a start node with an end node in the defined `line_shp_name`.

Return type
None

flusstools.geotools.shortest_path.**get_full_path**(*path*, *nx_list_subgraph*)

Creates a numpy array of the line result.

Parameters

- **path** (*str*) – Result of `nx.shortest_path`
- **nx_list_subgraph** (*list*) – See `create_shortest path` function

Returns
Coordinate pairs along a path.

Return type
ndarray

flusstools.geotools.shortest_path.**get_path**(*n0*, *n1*, *nx_list_subgraph*)

Get path between nodes **n0** and `n1`.

Parameters

- **n0** (*int*) – Node 1
- **n1** (*int*) – Node 2
- **nx_list_subgraph** (*list*) – (see create shortest path)

Returns
An array of point coordinates along the line linking these two nodes.

Return type
ndarray

flusstools.geotools.shortest_path.**write_geojson**(*outfilename*, *indata*)

Creates a new GeoJSON file

Args>
outfilename (str): Name for the output file indata (array): Array to write tyo the geojson file.

Returns
Creates a new GeoJSON file.

CHAPTER

FIVE

FUZZYCORR

FuzzyCorr was developed along with novel fuzzy map comparison methods to evaluate the performance of hydro-morphodynamic numerical models. The procedure and math behind the package are described in Negreiros et al. 2021 (open-access paper) .

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.

5.1 Usage

5.1.1 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
```

5.1.2 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`.

5.2 Structure

This package contains the following modules:

- *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*.

5.2.1 Pre-processing: prepro.py

Pre-processing structures for the fuzzy map comparison.

Some logic here overlaps with `flusstools.geotools` and is reused from there instead of being duplicated:

- `plain_raster` delegates to `geotools.geotools.rasterize`.
- `clip_raster` was removed earlier (duplicate of the `geotools` raster clip).

The remaining candidates are *not* drop-in duplicates and stay local:

- `array2raster` (rasterio-based, bound to this instance's CRS/extent/resolution) only partially overlaps `geotools.raster_mgmt.create_raster`.
- `create_polygon` (alphashape over this instance's in-memory points) only partially overlaps `geotools.shp_mgmt.polygon_from_shapepoints`, which reads a shapefile from disk instead.

class flusstools.fuzzycorr.prepro.**CategorizationPreProcessor**(*raster*)

Structured for … (Description to be implemented by Bea)

Parameters

(str) (*raster*) – 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=(numpy.nan, numpy.nan)*, *lrc=(numpy.nan, numpy.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
- **(str)** (*crs*) – name of the attribute to burn in the raster (ex.: deltaZ, Z)
- **(str)** – coordinate reference system
- **(float)** (*res*) – value to indicate nodata cells
- **(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

- **(str)** (*raster_file*) – path to save the rasterfile
- **(bool)** (*save_ascii*) – true to save also an ascii raster

Returns None
Saves the raster with the selected filename

> **Hint**
>
> Function will be moved to `geotools/raster_mgmt` in a future release (operated by Bea)

create_polygon(*shape_polygon*, *alpha=numpy.nan*)

Creates a polygon surrounding a cloud of shapepoints

Parameters

- **(str)** (*shape_polygon*) – path to save the shapefile
- **(float)** (*alpha*) – 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.

Delegates to *flusstools.geotools.geotools.rasterize()*, burning this pre-processor's `attribute` into the raster pixels (no duplicated GDAL rasterization code is kept here).

Parameters

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

Returns int
`0` on success (`None` on failure); saves the raster to `raster_file`.

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 numpy.ndarray
array of random values within a range of the same size and shape as the original

5.2.2 Fuzzy map comparison core: fuzzycomp.py

Head structure for fuzzy map comparisons

Usage: `fuzzy_comparison = FuzzyComparison()`

Descriptions will be updated by Bea

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.**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

5.2.3 Plot routines: plotter.py

Plotting routines and classes for fuzzy comparison maps

class flusstools.fuzzycorr.plotter.**RasterDataPlotter**(*path*)

Class of raster for plotting

Parameters
(str) (*path*) – 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

- **(str)** (*output_file*) – legend of the x axis of he histogram
- **(str)** – legend of the y axis of he histogram
- **(int)** (*fontsize*) – size of the font
- **(str)** – path for the output file
- **(tuple)** (*figsize*) – of integers, size of the width x height of the figure
- **(float)** (*set_ylim*) – set the maximum limit of the y axis
- **(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
- **(list)** (*labels*) – of strings, labels (i.e., titles)for the categories
- **(str)** (*cmap*) – 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

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

Returns

saves the figure of the raster

plot_continuous_raster(*output_file*, *cmap*, *vmax=numpy.nan*, *vmin=numpy.nan*, *box=True*)

Creates a figure of a continuous valued raster

Parameters

- **output_file** – path, file path of the figure
- **(str)** (*cmap*) – colormap to plot the raster
- **(float)** (*vmin*) – optional, value maximum of the scale, this value is used in the normalization of the colormap
- **(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 (int): width (number of cells) of the zoomed window :param height (int): height (number of cells) of the zoomed window :param bounds (list): of float, limits for each color of the colormap :param cmap (str): optional, colormap to plot the raster :param list_colors (list): of colors (str), optional, as alternative to using a colormap :returns None: saves the figure of the raster

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

Opens a raster using rasterio

Parameters

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

Returns

a numpy array of the raster

Return type

ndarray

5.3 References

- Ross Kushnereit
- Chris Wills

CHAPTER

SIX

CONTRIBUTING

6.1 Become a contributor

Many team members joined while working on their Bachelor's or Master's thesis. If you are a student, why not contribute to *flusstools* within an innovative thesis? Have a look at the open Bachelor and Master Thesis topics.

You do not have to be a student to join - just reach out to Sebastian Schwindt.

6.2 How the project is organized

flusstools lives in **two** GitHub repositories:

- flusstools-pckg - the Python **code**, released to PyPI.
- flusstools-docs - **this documentation** (the `.rst` text files).

The documentation installs *flusstools* straight from PyPI and builds the function/class reference **automatically from the docstrings in the code**. So you never copy code into the docs repo - you publish a release of the code, and edit the text here.

6.3 Set up a development environment

GDAL only installs reliably with conda/mamba (see *Installation*). Clone the code repo, create the environment, and install *flusstools* in editable mode:

```
git clone https://github.com/Ecohydraulics/flusstools-pckg.git
cd flusstools-pckg
mamba env create -f environment.yml
mamba activate flussenv
pip install -e . --no-deps
```

`--no-deps` is used because all dependencies already come from `environment.yml`. "Editable" (`-e`) means your code changes take effect immediately, without reinstalling.

6.4 Write code

A few rules keep the package clean and the docs working:

- **Imports:** every module imports exactly what it uses, at the top of the file - there is no central import file. To use a function from another *flusstools* module, import it directly from where it is defined, e.g. `from ..geotools.geotools import rasterize`.

- **Docstrings are the docs.** Write a Google-style docstring for every public function and class; it is exactly what shows up in this documentation. The sections you normally need:
  - `Args:` - describe each parameter
  - `Returns:` - describe what comes back
  - `Raises:` - errors it may raise *(optional)*
  - `Example:` - a short usage snippet *(optional, but appreciated)*
- **Make a function public:** if users should be able to call your new function or class, add its name to the `__all__` list in that subpackage's `__init__.py`.
- **Added a new library?** Declare it in three places: `pyproject.toml` (`dependencies`), `environment.yml`, and - using its *import* name (e.g. `skfuzzy`, not `scikit-fuzzy`) - the `autodoc_mock_imports` list in `flusstools-docs/docs/conf.py`.
- Keep it readable, and only push code that actually runs.

6.5 Release a new version

Once your code works and is merged, publish it to PyPI so that users - and these docs - receive it. The full recipe is in CONTRIBUTING.md; the short version:

1. Bump `version` in `flusstools-pckg/pyproject.toml` (e.g. `2.0.1` to `2.0.2`).
2. Build and upload (needs a PyPI API token):

```
mamba run -n flussenv python -m build
TWINE_USERNAME=__token__ python -m twine upload dist/*
```

3. Tag the release on GitHub:

```
git tag -a v2.0.2 -m "FlussTools 2.0.2"
git push --tags
```

6.6 Update this documentation

Edit (or add) the matching `.rst` file in `flusstools-docs/docs/` and push to `main` - Read the Docs rebuilds automatically:

```
git clone https://github.com/Ecohydraulics/flusstools-docs.git
cd flusstools-docs
# edit docs/<module>.rst
git commit -am "docs: describe <something>"
git push
```

To document a **new module**, add a `.. automodule:: flusstools.<subpackage>.<module>` block to the matching `.rst` file (copy the pattern from `geotools.rst`) and list the page in the `toctree` of `index.rst`.

Important

Only push code that you have run successfully - thank you!

CHAPTER

SEVEN

DISCLAIMER AND LICENSE

7.1 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.

7.2 BSD 3-Clause License

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

PYTHON MODULE INDEX

f

INDEX

G

H

I

K

M

N

O

P

R

S

V

W

X