CompareCC_Py4dGeo

Max Tiedl

Nov 30, 2022

CONTENTS:

CHAPTER

ONE

COMPARE_PY4DGEO_CC

1.1 src package

1.1.1 Submodules

1.1.2 src.compare module

class `src.compare.`**`Compare`**(*re_pts*, *re_normals*, *re_dist*, *re_lod*, *cl_pts*, *cl_normals*, *cl_dist*, *cl_lod*)

Bases: `object`

A class for comparing the outputs of the m3c2-algorithm between CloudCompare and Py4dGeo

`re`

Stores all the information of the reference point cloud (Cloud compare) -> coordinates, distances, lodetection, spread1, spread2 and normal coordinates

Type
dict

`cl`

Stores all the information of the point cloud (Py4dGeo) -> coordinates, distances, lodetection, spread1, spread2 and normal coordinates

Type
dict

`size`

The number of points in each cloud

Type
int

`diffs`

The differences between between each related argument of both clouds -> gets calculated in calc_differences()

Type
dict

`aspects`

The aspects between related normal vectors in both clouds.

Type
numpy.array

`slopes`

The slopes between related normal vectors in both clouds.

Type
numpy.array

`calculate_differences()`

Calculate all the differences between related arguments of this class

Parameters
self (Compare) – the object itself

Returns
The differences between each related argument of both point clouds

Return type
dict

`mapDiff`(*path1*, *path2*, *proj='2d'*, *advanced=False*, *ps=10*)

Handle the plots of distance and lodetection. Decide if the plots are single-plotted or the distance/lodetection plots from CC and Py4dGeo are also shown.

Parameters

- **self** (Compare) – The object itself.
- **path1** (*str*) – The output path for the distance plot.
- **path2** (*str*) – The output path for the lodetection plot.
- **proj** (*str*) – Specifies whether the plot is plotted in 2d or 3d.
- **advanced** (*bool*) – Specifies whether the differences gets plotted in comparison to the CC and Py4dGeo Distances/LODetection or just alone.
- **ps** (*int*) – Sets the point size.

`plotNormDiff`(*path*)

Draw the normal differences between both point clouds on a polar plot by using matplotlib.

Parameters

- **self** (Compare) – The object itself.
- **path** (*str*) – The output path for the normal plot.

`plotNormHist`(*path*)

Plot the normal differences between both point clouds in a histogram.

Parameters

- **self** (Compare) – The object itself.
- **path** (*str*) – The output path for the normal plot.

`sampleDiff`(*re_samples*, *cl_samples*, *path*, *plot=False*)

Plot point density differences between both point clouds.

Parameters

- **self** (Compare) – The object itself.
- **path** (*str*) – The output path for the spread plot.

spreadDiff(*re_spread*, *cl_spread*, *path*, *plot=False*)

Plot standard deviation differences between both point clouds.

Parameters

- **self** (Compare) – The object itself.
- **path** (*str*) – The output path for the spread plot.

writeCloud(*path*, *cc_mode=True*)

Handle writing to different filetypes(ascii and las/laz), so theres no need to change the function when using a different file extension.

Parameters

- **self** (Py4d_M3C2) – The object itself.
- **cc_mode** (*bool*) – Specifies if the header is written with CC vocabulary or py4dgeo vocabulary.

writeDiff(*path*)

Write all the differences between both point clouds to a csv file.

Parameters

- **self** (Compare) – The object itself.
- **path** (*str*) – The output path for the csv file.

writeStatistics(*path*)

Write statistics (mean, median, standard-deviation) of the differences between both point clouds to a csv file:

Parameters

- **self** (Compare) – The object itself.
- **path** (*str*) – The output path for the csv file.

1.1.3 src.file_handle module

src.file_handle.**read_las**(*pointcloudfile*, *get_attributes=False*, *useevery=1*)

Read a pointcloud from a las/laz file.

Parameters

- **pointcloudfile** (*str*) – specification of input file
- **get_attributes** (*bool*) – if True, will return all attributes in file, otherwise will only return XYZ (default is False)
- **useevery** (*int*) – value specifies every n-th point to use from input, i.e. simple subsampling (default is 1, i.e. returning every point)

Returns

3D array of points (x,y,z) of length number of points in input file (or subsampled by ‘useevery’)

Return type

numpy.ndarray

src.file_handle.**read_xyz**(*pointcloudfile*, *get_attributes=False*)

Read a pointcloud from an ascii file.

`src.file_handle.`**`write_las`**(*points*, *path*, *attributes={}*)

Write a point cloud to a las/laz file

Parameters

- **points** (*numpy.array*) – 3D array of points to be written to output file
- **path** (*str*) – specification of output file
- **attributes** (*dict*) – dictionary of attributes (key: name of attribute; value: 1D array of attribute values in order of points in 'outpoints'); if not specified, dictionary is empty and nothing is added

`src.file_handle.`**`write_xyz`**(*points*, *path*, *attributes={}*)

Write a point cloud to a las/laz file

Parameters

- **points** (*numpy.array*) – 3D array of points to be written to output file
- **path** (*str*) – specification of output file
- **attributes** (*dict*) – dictionary of attributes (key: name of attribute; value: 1D array of attribute values in order of points in 'outpoints'); if not specified, dictionary is empty and nothing is added

1.1.4 src.main module

`src.main.`**`checkParams`**()

Check if console arguments are provided. The arguments are handled by the ArgumentParser class.

Returns

- str – Path to the first point cloud
- str – Path to the second point cloud
- str – Path to the corepoint file
- str – Path to the output directory
- str – Path to the CC parameter file
- str – Defines if the lodetection/distances plot gets mapped in 3d or 2d
- bool – Defines if the lodetection/distances plot gets mapped in advanced mode or not
- str – The point size for the lodetection/distance plots
- str – The file format for the pointcloud files
- str – The path to the CC binary
- bool – If the py4dgeo calculations should use CC normals
- bool – Skip the calculations in py4dgeo/CC and instead provide pointcloud files
- bool – Repeat the difference calculations but skip the py4dgeo/CC calculations

Return type

tuple

src.main.**reorder**(*cloud*)

Rearrange certain parameters.

Parameters

cloud (*numpy.ndarray*) – All the parameters of a the point cloud.

Returns

- numpy.ndarray – The normals of the cloud.
- numpy.ndarray – The spread information.
- numpy.ndarray – The num_sample information.

Return type

tuple

1.1.5 src.map_diff module

class src.map_diff.**Map_Diff**(*mapped_vals*, *coords*, *title=''*, *unit='m'*, *point_size=1*, *cmap='YlGnBu'*)

Bases: object

This class can be used to plot certain values as colors onto given coordinates either in a 3d or a 2d plot, by using matplotlib.

Parameters

- **mapped_vals** (*numpy.ndarray*) – The values to be plotted as colors.
- **coords** (*numpy.ndarray*) – The points to be plotted.
- **title** (*str*) – The title of the plot.
- **size** (*int*) – The number of coordinates.
- **unit** (*str*) – The unit used for the axes.

compare(*vals*, *crds*, *ttls*, *output=False*, *proj='2d'*, *show=False*)

Show a plot, that makes comparison easier by giving the oportunity to also plot different points and data next to the the initial plot. Subplots are used.

Parameters

- **self** (Compare) – The object itself.
- **vals** (*list*) – A list of arrays, whereby each array contains one set of values that gets plotted as colors. More than one set of values is possibly, but then also more coords must be provided.
- **crds** (*list*) – A list of coordinate arrays. Each coordinate array will be plotted in a different subplot.
- **ttls** (*list*) – A list of titles for each subplot.
- **output** (*str*) – The path for the output file. If not set the plot wont be saved.
- **show** (*bool*) – Specifies if the plot is shown or not.
- **proj** (*str*) – Specifies if the plot is 2d or 3d.

mapDiff(*output=False*, *show=False*, *proj='2d'*)

Create a 2d/3d plot that shows a point cloud and maps given values as colors onto them.

Parameters

- **self** (Compare) – The object itself.
- **output** (*str*) – The path for the output file. If not set the plot wont be saved.
- **show** (*bool*) – Specifies if the plot is shown or not.
- **proj** (*str*) – Specifies if the plot is 2d or 3d.

plot(*crds*, *vals*, *ttl*, *ax*, *proj='2d'*)

Draw given coordinates in a Subplot.

Parameters

- **self** (Compare) – The object itself.
- **vals** (*numpy.ndarray*) – The values to be plotted as colors
- **crds** (*numpy.ndarray*) – The points to be plotted.
- **ttl** (*str*) – The title of the subplot.
- **ax** (*matplotlib.axes._subplots.AxesSubplot*) – The Axes.
- **max** (*float*) – The maximum of the mapped values.
- **min** (*float*) – The minimum of the mapped values.
- **proj** (*str*) – Specifies if the plot is 2d or 3d.

1.1.6 src.py4d_m3c2 module

class src.py4d_m3c2.**Py4d_M3C2**(*path1*, *path2*, *corepoint_path=False*, *output_path=False*, *params=False*, *cc_normals=None*, *compr=1*)

Bases: `object`

A class for calculating distances between point clouds by implementing the m3c2-algorithm from the py4dgeo library.

Py4d_M3C2 also provides some extra functionality, such as general read/write functions for ascii(xyz/txt) and las/laz files. It is also able to read parameters from a CloudCompare param file. And for a better impression it also can plot the distances and lodetection by using the matplotlib package.

epoch1

An epoch object of the first point cloud.

Type

py4dgeo.epoch

epoch2

An epoch object of theh second cloud.

Type

py4dgeo.epoch

output_path

The name and path for the output file.

Type

str

`params`

Either the path to a CC params file or a dictionary of params

Type
str/dict

`corepoints`

Either the pointcloud from a separate corepoint file or a subsampled version of the first point cloud.

`read`(**path*, *other_epoch=None*, ***parse_opts*)

Handle reading epochs from different file types(ascii and las/laz), so theres no need to change the function when using a different file extension.

Parameters

- **self** (Py4d_M3C2) – The object itself.
- ***path** (*str*) – The path to a point cloud file. Can also handle multiple files.

`read_cc_normals`(*path*)

Read CloudCompare normals from a given file

Parameters

- **self** (Py4d_M3C2) – The object itself.
- **path** (*str*) – Path to a computed CloudCompare file.

Returns
Contains the normals.

Return type
numpy.array

`read_cc_params`()

Read the required parameters from a given file out and store them in a dictionary.

Parameters
self (Py4d_M3C2) – The object itself.

Returns
A dictionary containing the required parameters for the m3c2-algorithm.

Return type
dict

`run`()

Main function for calculating the distances. Implements the m3c2-algorithm from the py4dgeo library.

Parameters
self (Py4d_M3C2) – The object itself.

Returns
The calculated m3c2 distances.

Return type
dict

`write`(*cc_mode=True*)

Handle writing to different filetypes(ascii and las/laz), so theres no need to change the function when using a different file extension.

Parameters

- **self** (Py4d_M3C2) – The object itself.
- **cc_mode** (*bool*) – Specifies if the header is written with CC vocabulary or py4dgeo vocabulary.

class src.py4d_m3c2.**Vertical_M3C2**(*normal_radii: Optional[List[float]] = None*, *orientation_vector: ndarray = array([0, 0, 1])*, *corepoint_normals: Optional[ndarray] = None*, *cloud_for_normals: Optional[Epoch] = None*, ***kwargs*)

Bases: M3C2

directions()

The normal direction(s) to use for this algorithm.

1.1.7 src.vec_calc module

src.vec_calc.**getAngle**(*v1*, *v2*)

Calculate the angle between two vectors.

Parameters

- **v1** (*numpy.array*) – The first vector.
- **v2** (*numpy.array*) – The second vector.

Returns

The calculated angle.

Return type

float

src.vec_calc.**getAspect**(*normal*)

Calculate the aspect of a given vector.

Parameters

normal – The vector whose aspect should be returned.

Returns

The calculated aspect.

Return type

float

src.vec_calc.**getSlope**(*normal*)

Calculate the slope of a given vector.

Parameters

normal – The vector whose slope should be returned.

Returns

The calculated slope.

Return type

float

src.vec_calc.**rotate**(*v1*, *v2*, *norm*, *angl*)

Take two vectors and rotate the first one around a given line, the second one gets rotated in the same manner, so that both vectors keep their angle to each other.

Parameters

- **v1** (*numpy.array*) – The first vector.

- **v2** (*numpy.array*) – The second vector.
- **norm** (*numpy.array*) – A vector that represents the direction of a line. vec1/vec2 will be rotated around this line.
- **angl** (*numpy.array*) – A given angle that defines about how many degrees both vectors gets rotated

Returns

- numpy.array – The rotated first vector.
- numpy.array – The rotated second vector.

Return type

tuple

src.vec_calc.**transform**(*v1*, *v2*)

Take two vectors as input and rotate the second vector, so that it has the same angle to the z-unitvector as the first vector to the second one.

Parameters

- **v1** (*numpy.array*) – The first vector.
- **v2** (*numpy.array*) – The second vector.

Returns

The rotated second vector.

Return type

numpy.array

1.1.8 Module contents

PYTHON MODULE INDEX

INDEX