hiegeo

Alessandro Comunian

Oct 02, 2019

CONTENTS:

Welcome to the *hiegeo*'s documentation, a Python module to model geology taking into account for hierarchy and chronological order among the geological structures.

CHAPTER

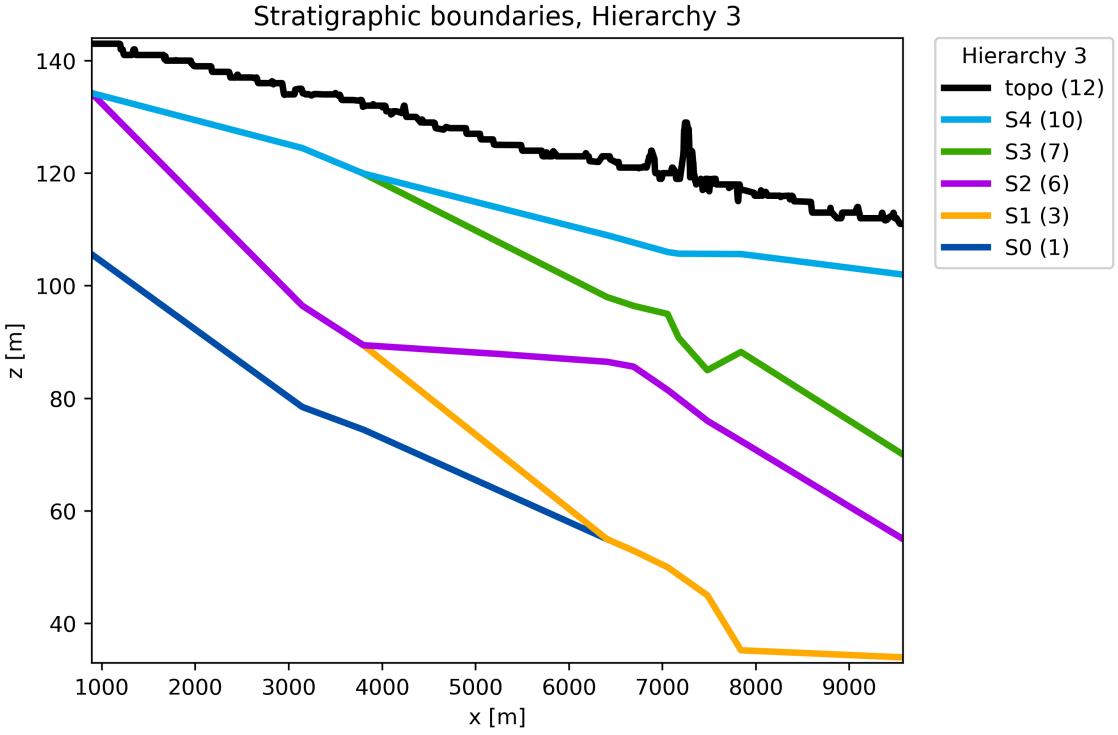
ONE

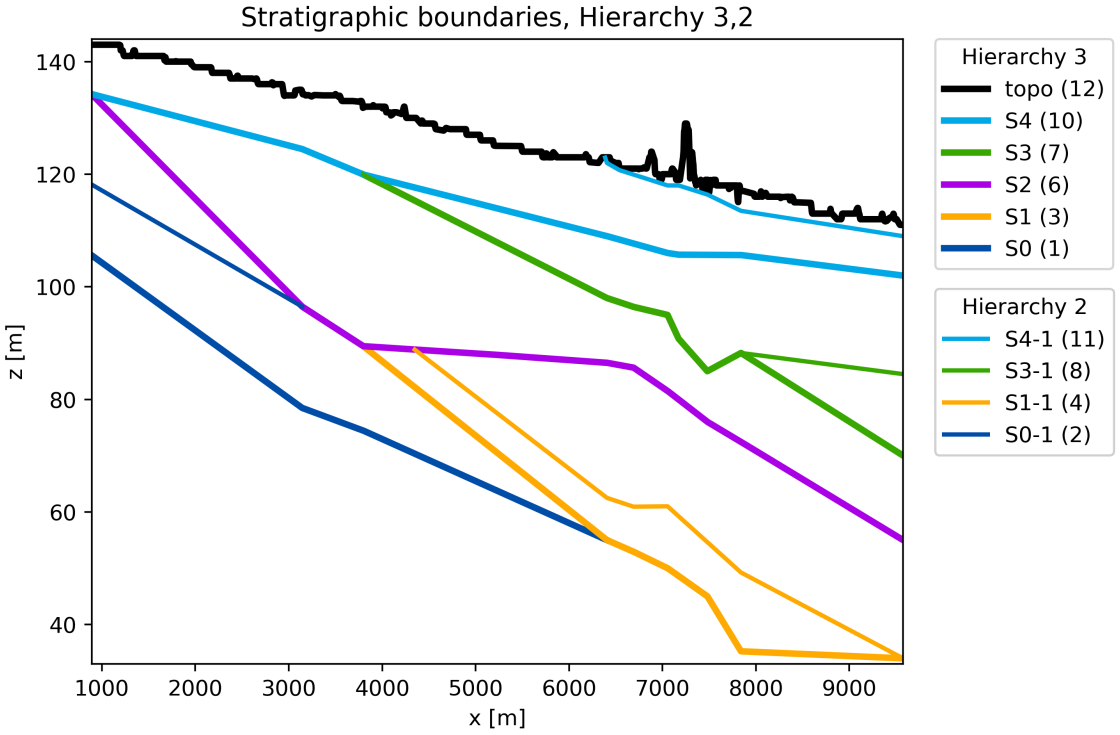
PURPOSE

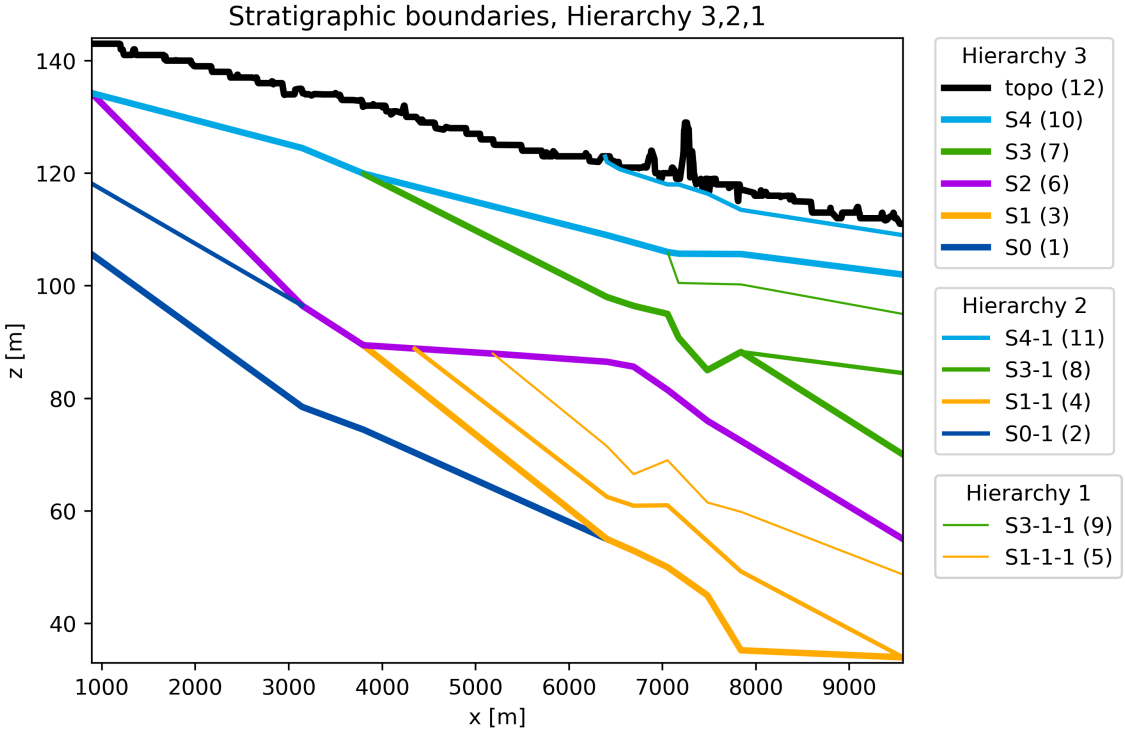
The purpose of the *hiegeo* Python module is to propose a novel approach to geological modeling, that takes into account for the hierarchy and the chronological order of the geological structures to be represented.

See the manuscript *hiegeo: a novel Python module to model stratigraphic alluvial architectures, constrained by stratigraphic hierarchy and relative chronology* by Chiara Zuffetti, Alessandro Comunian, Riccardo Bersezio, and Philippe Renard for more details.

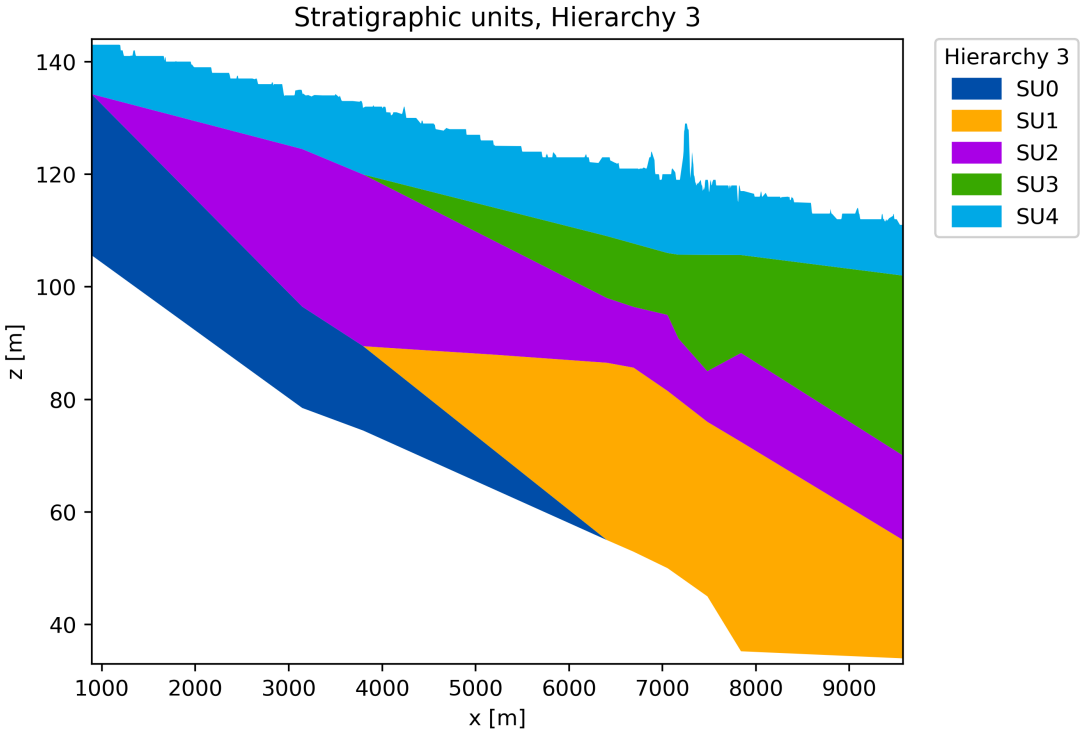
Hereinafter you can see one of the possible output provided by *hiegeo*, for example a representation of the geology in terms of stratigraphic boundaries (SBs) including only the 3rd level of hierarchy, the 3rd and the 2nd, and 3rd, 2nd and 1st:

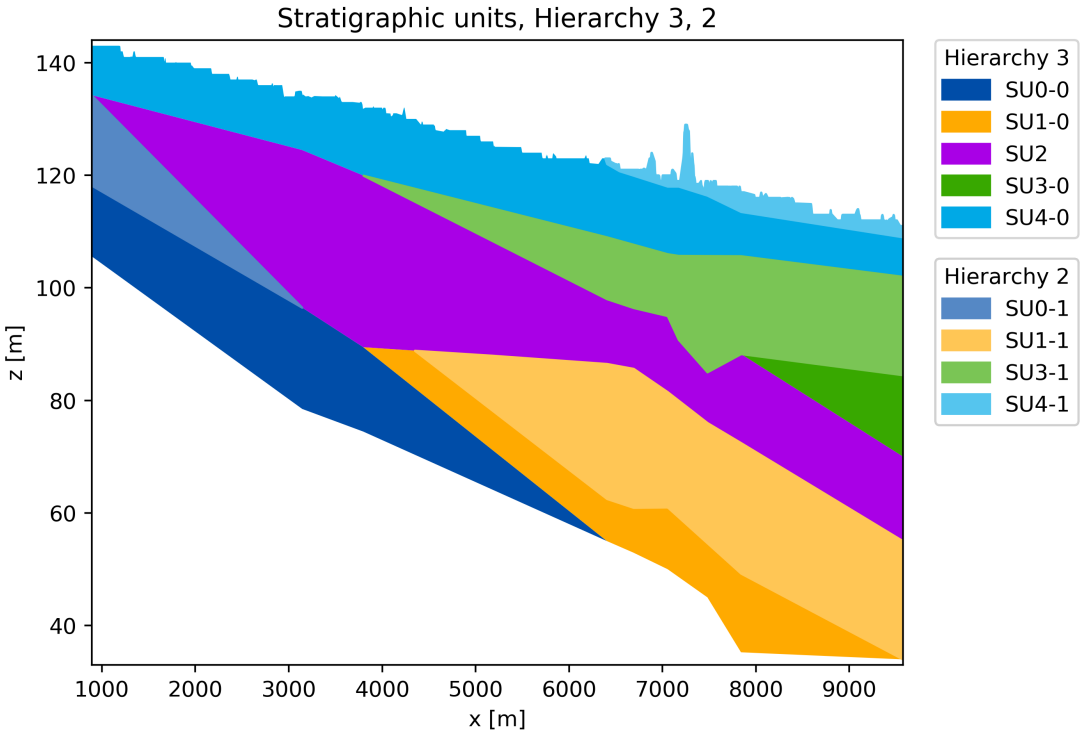


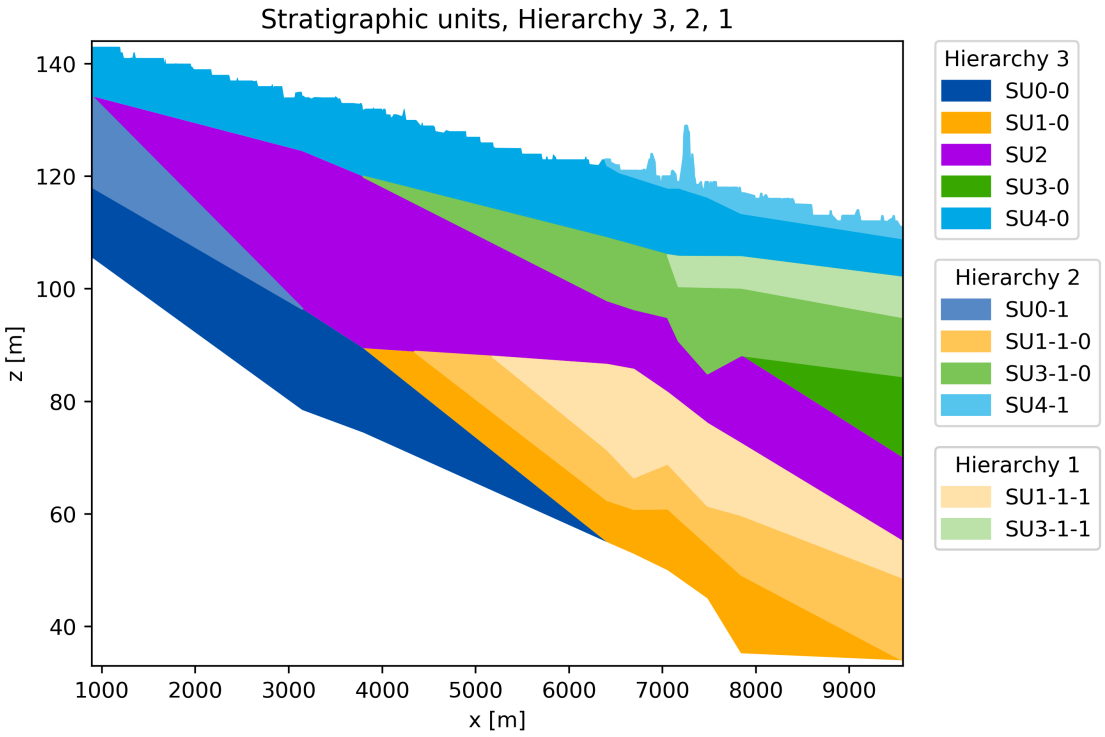




The same representation at different hierarchical levels can be obtained in terms of Stratigraphic Units (SUs):


Stratigraphic units, Hierarchy 3
z [m]
x [m]
140
120
100
80
60
40
1000
2000
3000
4000
5000
6000
7000
8000
9000
Hierarchy 3
SU0
SU1
SU2
SU3
SU4


Stratigraphic units, Hierarchy 3, 2
z [m]
x [m]
140
120
100
80
60
40
1000
2000
3000
4000
5000
6000
7000
8000
9000
Hierarchy 3
SU0-0
SU1-0
SU2
SU3-0
SU4-0
Hierarchy 2
SU0-1
SU1-1
SU3-1
SU4-1



In addition, you can also have a representation of the geological hierarchy as a tree structure:

topo
S4
└── S4-1
S3
└── S3-1
    └── S3-1-1
S2
S1
└── S1-1
    └── S1-1-1
S0
└── S0-1

CHAPTER

TWO

INSTALLATION

Installing and using *hiegeo* should be relatively easy, and thanks to Python's flexibility, that should be feasible on many operative systems (OSs), including MS Windows, Mac OS X and Linux.

2.1 Requirements

To use the *hiegeo* module you need a standard Python3.X installation, together with the main mathematical and plotting libraries (`numpy`, `pandas`, `matplotlib` etc) and the module `anytree`, which is required to handle the hierarchical structures.

2.1.1 Python

Probably, unless you are working with Linux and you can use some package manager like *Synaptic*, the easy way to have Python up and running on your system is to use *Anaconda* (https://www.anaconda.com/). Therefore, download and install the Python (version 3.X) of Anaconda which is suitable for your OS by following the instruction provided on the Anaconda web site.

2.1.2 The *anytree* module

Some capabilities of *hiegeo* are provided by the Python *anytree* module (https://pypi.org/project/anytree/). Once installed Anaconda, you can open the *Anaconda prompt* and install it by typing:

```
pip install anytree
```

To verify if the installation worked properly, you can open a Python shell and check that the line `import anytree` works and does not provide any error output.

2.1.3 The *hiegeo* module

If you keep the provided module file (`hiegeo.py`) in the same directory of the calling script, then everything should work as it is.

Alternatively, if you prefer to have more flexibility, you can put the file `hiegeo.py` in a directory that should be included in your `PYTHONPATH` environmental variable. If you are using the editor *Spyderlib*, included in Anaconda, to do that you can use the `Python Path Manager`; in that case you will need to **restart** Spyderlib.

2.2 Verification

In the end, to verify if `hiegeo` was properly installed and available to your Python script, you can call it from the Python shell with

```
include hiegeo
```

to double check is any error rises.

CHAPTER

THREE

USAGE

This section describe an example usage of *hiegeo*.

3.1 Input files

For properly run *hiegeo*, you need two files:

1) A *.JSON* file, that contains info about the discretization grid and the rendering of the plots.
2) A data file (*.CSV* format) that include the actual data set, with information about the contact points, chronology and hierarchy.

3.1.1 JSON file

An example *JSON* file is provided in the file `hiegeo_test.json`. It contains three main sections:

colors This section contains the definition (HEX format) of the color that should be given to each SB. The color should be provided for the highest hierarchy SBs only.

data_file The name of the CSV data file containing the dataset itself.

grid Info about the grid used to discretize the domain. The notation should be straightforward.

3.1.2 CSV file

This file contains all the required information to create and plot SBs and SUs.

Its header should look like:

,gis_id,chronology,hierarchy,sb_name,su_name,x,z
0,1,10,3,S4,LCN,897.1953,134.202585
1,2,10,3,S4,LCN,3142.7824,124.47562
2,3,10,3,S4,LCN,3797.4044,119.94562
...

Apart from the very first column, that contains some index that are used by the Python library `pandas` (but here they could be set to whatever), the other columns content should be:

gis_id This is simply a legacy column that contains the ID of the points extracted from the GIS software. You can set whatever value here as these are not used by *hiegeo*.

chronology This is a very important column since it contains information about the chronology of the corresponding SB. The value should be an integer, that grows from the oldest to the youngest SB.

hierarchy Another very important column, contains the hierarchy (sometimes called "rank") of the SB. High level of hierarchy means high importance in sedimentary terms. In the provided example we have integer values ranging from 1 to 3.

sb_name This is the name to be given to the SB. It should be a string

su_name This is the name of the SU. Actually, this column is not implemented and the SU is named according to and internally consistent nomenclature.

x The coordinate along the x axis of the (contact) points

y The coordinate along the y axis of the (contact) points

3.2 Example scripts

Here two example scripts are provided. One file (*hiegeo_test-simple.py*) contains a full working example to read data, plot with a basic layout SBs and SUs, and provide a hierarchical representation of the geology with a tree structure. The other file, instead (*hiegeo_test-full.py*) provides a more complete example where plots are made for three different levels of hierarchical representation, with advanced plot legend and includes the creation of a GSLIB output file.

Both script should be sufficiently documented to allow running them without additional information.

CHAPTER

FOUR

MODULE DOCUMENTATION

This file `hiegeo.py`

Purpose hie-geo: "hierarchical geo-modeling"

A module containing the functions required to read information from a geological data-set (contact point) and plot in a hierarchical fashion the unit sections.

Usage See `hiegeo_test-simple.py` and `hiegeo_test-full.py` for an example usage.

Version

1.9, 2019-10-02 : Some clean up an adding License information.

1.8, 2019-07-31 : Version updated with some clean up, before important changes in "rank" and "order" definitions...

Some conventions

SB: Stratigraphic Bound

SU: Stratigraphic Unit

Hierarchy: Minumum value stands for lower hierarchical importance; higher values are for more important SBs.

chronology: This is the temporal order of the SBs. Lower values corresponds to older SBs, while higher values correspond to more recent SBs.

Authors Alessandro Comunian

Note:

1) At the moment, the points of intersection among surfaces are a required input data. Also, it is expected that when two SBs are intersecting at one point, the point is defined in the dataset for both the SBs.

ALTRI DUBBI:

"Correlation points" or "contact points"?

class hiegeo.**SBound**(*name*, *data=None*, *hierarchy=None*, *chronology=None*, *xd=None*, *color=None*, *parent=None*)

A class useful to contain all the properties of a Stratigraphic Boundary (SB).

broken_line()

This is useful to create, for each point of the provided discretization along *x* axis, (*self.xd*), a broken line. A *numpy.nan* value is set where the provided "raw" coordinates does not allow to complete the discretization.

get_ancestors(*anc_list=[]*)

Provide a list of ancestors of the current SB.

Parameters:

anc_list: list A partially filled input list can be provided, to allow appending additional ancestors.

get_obj_above(*hierarchy*)

Get all the SBs defined in the script with the given hierarchy, ordeder by "chronology".

Parameters:

hierarchy: integer The hierarchy of the SBs to be plotted.

Returns: A list containing all the defined SBs with the given hierarchy, ordered by "chronology".

plot(*lw=None*, *chronology=None*)

Plot the SB.

Parameters:

lw: integer This is the line width

chronology: integer The relative chronology of the SB.

Note: The values where `zd==np.nan` are not plotted.

---

plot_ax(*ax*, *lw=None*, *color=None*)

Plot the SB when a matplotlib axis is provided.

Parameters:

lw: integer (optional) The line width. Using directly the hierarchies (for example, with hierarchies from 1 to 3) seems to provide a figure which is OK

color: color code (optional) The line color.

Note: The values where *zd==np.nan* are not plotted.

---

plot_fill(*ax*, *hierarchy*, *alpha*)

"fill_between" plot of a SU given the bottom SB.

Parameters:

ax: matplotlib axis The object where to make the plot

hierarchy: integer Hierarchy.

alpha: float Transparency level.

Note: Using *alpha* as parameter could create problems when one tries to properly represent overlapping regions. (and this, in principle, should happen quite frequently when dealing with hierarchical models...)

print_hie()
Print out some info about a SB, in a hierarchical fashion.

class hiegeo.**SUnit**(*name=None*, *bot=None*, *top=None*, *sbs=None*, *id=None*)
A class useful to contain all the properties related to a SU.

Note: Is is expected that in the same script all the SBounds were already created extracting the information from the correlation/contact points file.

Warning: The development of this Class is still work in progress, and only basic functionalities are provided for the moment.

plot()
Plot a SU

hiegeo.**check_data_ingrid**(*data*, *x*, *z*)
Check if data are contained in the discretization grid.

Parameters:

data: (pandas DataFrame) The input data set. See the documentation for more details.

x, y: numpy arrays These are the discretization grid coordinates.

Note: Actually, the warning message in general is not a "nasty" one, since it only states that the definining points of some SBs were not available on all the vertical coordinates, and therefore the domain was restricted.

hiegeo.**create_sb_by_hierarchy**(*data*, *hierarchy*, *xd*, *color=None*)
Given a data set containing the correlation points and a hierarchy, create a list of SB objects.

Parameters:

data: DataFrame The info about the correlation points

hierarchy: integer The hierarchy that should be created

xd: numpy array Discretized *x* coordinate

Returns: A list containing the SB objects, ordered by "chronology"

hiegeo.**create_sb_from_data**(*data*, *xd*, *color=None*)
Create all the SBs defined in a dataset

Parameters:

data: DataFrame Info about the correlation points

Returns: A dictionary containing, for each hierarchy, a list of SBs.

hiegeo.**get_SBparent_by_name**(*data*, *sb_name*)
Given a data set, find the "parent" SB of a SB with a given name.

Parameters:

data: data set (see package documentation for details) The data set where to look for parents.

sb_name: string The name of the SB of which looking for a "parent"

Return: The parent object SB

hiegeo.**get_SU_by_bot**(*bot*)
Find the SU object corresponding to a given bottom SB.

hiegeo.**get_allmin**(*sbs*)
Given a list of objects, return the min of the *z* coordinate.

Note: Warnings about all columns containing a NaN in *nanmin* is suppressed as it should not be harmful.

---

hiegeo.**get_bot_sb**(*sb*, *sbs*)
Get a list of SB objects that are potential candidates to provide the bottom for the given SB.

Parameters: sb: SBound sbs: list of SBound objects

hiegeo.**get_intersect**(*name*, *data*)
Get a dataframe of points of SBs that intersect with the SB "name". If there is no intersection, the output dataframe will be empty.

Parameters:

name: Name of the current SB we are working with

data: Dataset containing the point correlations

Returns: A daframe containing the intersection points.

hiegeo.**get_max_chronology**(*data*)
Get the maximum chronology defined in the dataset.

Parameters:

data: DataFrame Info about the correlation points

Return: The value of the max chronology

hiegeo.**get_newer_sb_same_hie**(*sb*)
Get all the defined SBs with the same hierarchy but of greater chronology.

Parameters:

sb: SBound The interested SB.

hiegeo.**get_parent_by_name**(*data*, *sb_name*)
Given a data set, find the "parent" SB of a SB with a given name.

Parameters:

data: data set (see package documentation for details) The data set where to look for parents.

sb_name: string The name of the SB of which looking for a "parent"

Return: Name of the parent SB

hiegeo.**get_sb_chronology**(*data*, *sb*)
Get the chronology of the current SB.

Parameters:

data: DataFrame The point correlation info contained in the DBF file

sb: string Name of the SB.

hiegeo.**get_sb_hierarchy**(*data*, *sb*)
Get the hierarchy of the current SB.

Parameters:

data: DataFrame The point correlation info contained in the DBF file

sb: string Name of the SB.

hiegeo.**get_sbobj**()
Get all the SBs defined within the script, ordeder by "chronology".

Returns: A list containing all the defined SBs ordered by "chronology".

hiegeo.**get_sbobj_by_hie**(*hierarchy*, *mode='eq'*, *reverse=False*)
Get all the SBs defined in the script with the given hierarchy, ordeder by "chronology".

Parameters:

hierarchy: integer The hierarchy of the SBs to be plotted.

mode: in ("eq" or "ge") Depending on this optional value, one can get all the SBs of the same hierarchy (mode="eq") or all the objects with a >= hierarchy.

Returns: A list containing all the defined SBs with the given hierarchy, ordered by "chronology".

hiegeo.**get_sbobj_by_name**(*sb_name*)
Get all the SBs defined within the script, ordeder by "chronology".

Returns: A list containing all the defined SBs ordered by "chronology".

hiegeo.**get_top_sb**(*sb*, *sbs*)
Get a list of SB objects that are potential candidates to provide the bottom for the given SB

Parameters: sb: SBound sbs: list of SBound objects

hiegeo.**get_unique_chronology**(*data*)
Extract from a dataset al the defined chronologies.

Parameters:

data: DataFrame Info about the correlation points

Return: A list containing the defined chronologies.

hiegeo.**get_unique_hierarchy**(*data*)
Extract from a dataset al the defined hierarchies.

Parameters:

data: DataFrame Info about the correlation points

Return: A list containing the defined hierarchies.

hiegeo.**get_unique_sb_name**(*data*)
Get an ordered list containing the unique names of the SBs.

Parameters:

data: input DataFrame See the documentation for more details about the data format.

Returns: A list containing the ordered ids

hiegeo.**migrate**(*df*, *xmesh*, *zmesh*)
Migrate the "sparse" coordinates of some points contained in the "x" and "z" of a DataFrame into the points or a regularly spaced grid.

Parameters:

df: pandas DataFrame The dataframe containing the coordinates "x" and "z" to be migrated.

xmesh: "x" output from meshgrid A matrix containing the *x* coordinates of a structured grid.

zmesh: "z" output from meshgrid A matrix containing the *z* coordinates of a structured grid.

hiegeo.**plot_sb_ax**(*ax*, *hies*, *palette=None*)
Plot all the defined objects for all the provided hierarchies.

Parameters:

ax: The axis object

hies: List of integers. Hierarchies to be plotted.

palette: Dictionary A dictionary containing the codes corresponding to each SB.

hiegeo.**plot_sb_by_hie**(*hierarchy*)
Plot all the SB of a given hierarchy.

Parameters:

hierarchy: integer The hierarchy of the SBs to be plotted

hiegeo.**plot_sb_by_hie_ax**(*ax*, *hierarchy*, *palette=None*)
Plot all the SB of a given hierarchy.

Parameters:

ax: axis object The axis object where the plot should be made

hierarchy: integer The hierarchy of the SBs to be plotted

palette: dictionary A dictionary containing the correspondence between SB name and plotting color.

hiegeo.**plot_sb_fig**(*fig*, *hies*, *palette=None*)
Plot all the defined objects for all the provided hierarchies.

Parameters:

fig: The axis object

hies: List of integers. Hierarchies to be plotted.

palette: Dictionary A dictionary containing the codes corresponding to each SB.

hiegeo.**plot_su_ax**(*ax*, *hierarchy*, *hie_alpha*)
Plot a SU provided a matplotlib axis.

Parameters:

ax: matplotlib axis The axis where to plot the SU.

hierarchy: integer The min hierarchy to be plotted.

hie_alpha: float Transparency value

hiegeo.**print_data_info**(*data*)
Print some info about the data.

hiegeo.**print_start**()
Print a header with the module name and the version.

CHAPTER

FIVE

NOTATION

Hereinafter some acronyms that will be used throughout the document:

SB Stratigraphic boundaries.

SU Stratigraphic units.

CHAPTER

SIX

INDICES AND TABLES

- genindex

PYTHON MODULE INDEX

h

INDEX