GemGIS

Alexander Juestel

Dec 22, 2023

GETTING STARTED

GemGIS is a Python-based, open-source geographic information processing library. It is capable of preprocessing spatial data such as vector data (shape files, geojson files, geopackages,...), raster data (tif, png,...), data obtained from online services (WCS, WMS, WFS) or XML/KML files (soon). Preprocessed data can be stored in a dedicated Data Class to be passed to the geomodeling package GemPy in order to accelerate the model building process. Postprocessing of model results will allow export from GemPy to geoinformation systems such as QGIS and ArcGIS or to Google Earth for further use.

GemGIS uses and combines the full functionality of GeoPandas, rasterio, OWSLib, Pandas, Shapely, PyVista and NumPy to simplify, accelerate and automate the workflows used to preprocess spatial data for geomodeling.

About GemGIS | *Installation* | *Tutorials* | *Examples*

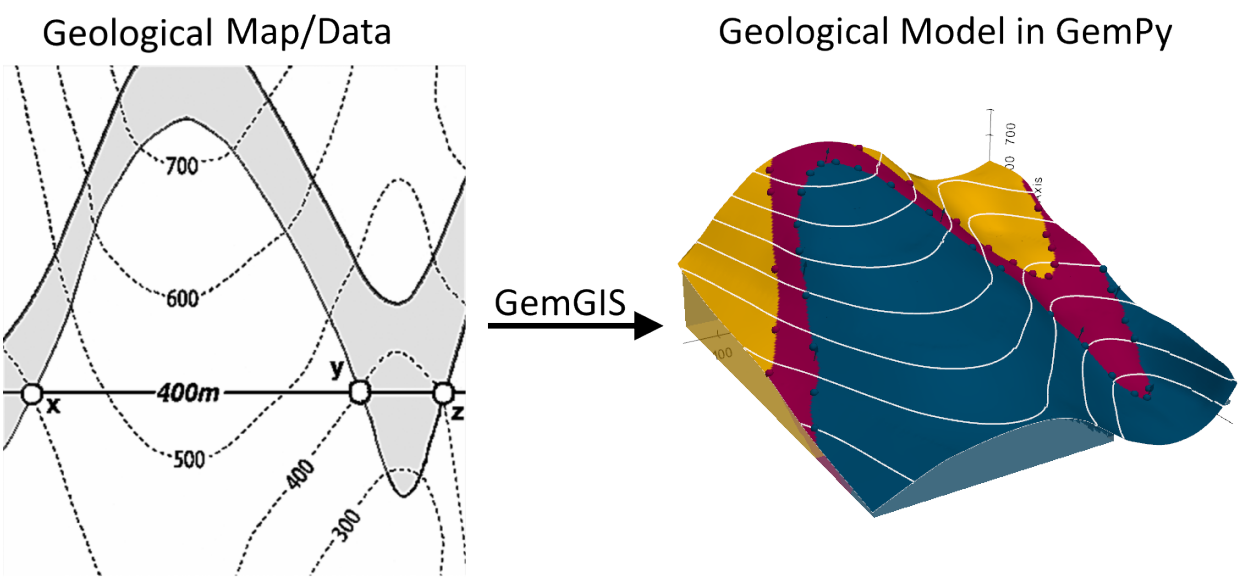
CHAPTER

ONE

ABOUT

GemGIS is a Python-based, open-source geographic information processing library. It is capable of preprocessing spatial data such as vector data (shape files, geojson files, geopackages,...), raster data (tif, png,...), data obtained from online services (WCS, WMS, WFS) or XML/KML files (soon). Preprocessed data can be stored in a dedicated Data Class to be passed to the geomodeling package GemPy in order to accelerate the model building process. Postprocessing of model results will allow export from GemPy to geoinformation systems such as QGIS and ArcGIS or to Google Earth for further use.

GemGIS uses and combines the full functionality of GeoPandas, rasterio, OWSLib, Pandas, Shapely, PyVista and NumPy to simplify, accelerate and automate the workflows used to preprocess spatial data for geomodeling.



Deployment	
GitHub	
Binder	
License	
Documentation	
Github Workflow	
Issue Tracking	
Pull Requests	

1.1 Content

This documentation page consists of a `Getting Started` section with information about *Authors, Citation, License*, how the *Installation (Tested 2023-09-01)* of GemGIS works, which Data Types are supported and which packages are included in GemGIS and most importantly for new users *Tutorials and Basic Usage* and Examples.

The `API Reference` section provides information about the different functions, that are implemented in GemGIS. This includes the GemGIS Data Object, Working with Vector Data, Working with Raster Data, Working with Online Services, different additional Utility Tools, Visualization and Plotting, different other methods not so frequently used or for specific cases under Miscellaneous and last but not least Postprocessing of GemPy Models.

Each set of functions is collected in a different module. The functions of each module can be accessed as followed:

```
import gemgis as gg

data = gg.vector.function_name(...)

data = gg.raster.function_name(...)

data = gg.visualization.function_name(...)

data = gg.web.function_name(...)

data = gg.utils.function_name(...)

data = gg.misc.functions_name(...)
```

1.2 Video Tutorials

There will be tutorial videos posted soon on YouTube where we will interactively explain the story behind GemGIS, walk through the installation process, introduce the different utilized packages and introduce the functionality of the different functions implemented in GemGIS.

1.3 Support

For general questions about the project, its applications, or about software usage, please create an issue in the cgre-aachen/gemgis repository. The community will then collectively address your questions. The developers of GemGIS can also be reached on the Software Underground Slack Workspace.

1.4 Citing GemGIS

If you are using GemGIS for your scientific research, please remember to cite our work. The citation is provided in the *Authors, Citation, License* section.

CHAPTER

TWO

AUTHORS, CITATION, LICENSE

GemGIS is an open-source project with a community of contributors ranging from Brazil to Europe and to Australia. Most of the core contributors of GemGIS are listed below but please make sure to also check out the Github Contributors Graph.

The development of the package was initiated during the Transform 2020 by the Department for Computational Geoscience and Reservoir Engineering at RWTH Aachen University, Germany and supported by most of the authors listed below.

2.1 Authors

The following list (sorted by name) shows the authors with substantial contributions to the conception or design of the software. The authors also provided new code or revised existing code and documentation.

- Arthur Endlein Correira, (@endarthur)
- Alexander Jüstel (@AlexanderJuestel)
- Marius Pischke
- Miguel de la Varga (@Leguark)
- Jan David Wagner (@JanWagner1312)
- Florian Wellmann (@flohorovic)

Revising documentation: * Jan Niederau (@Japhiolite) * Richard Scott (@RichardScottOZ)

2.2 FAIR Principle

The developers of GemGIS want to make the API, the tutorials and examples meet and adhere to the FAIR data principles (e.g. FAIR Principles).

Findable With each release, the data stored in the GemGIS repositories are uploaded to Zenodo where a persistent identifier is provided for each release. The data for the latest release of GemGIS can be found at https://zenodo.org/record/7602809#.Y-H9_XbMKUk and for the GemGIS data repository at https://zenodo.org/record/7494025#.Y-IL5nbMKUk. It is referred to Zenodo as the Github repositories do not strictly fulfill the criteria of having a globally unique and persistent identifier assigned to the (meta)data. However, all code and data can currently be found at https://github.com/cgre-aachen/gemgis and https://github.com/cgre-aachen/gemgis_data.

Accessible The files stored in the respective Zenodo repositories can be downloaded without registration as ZIP file. In addition, the data can be downloaded from the aforementioned Github repositories without registration as ZIP files

or via git. The functionality of GemGIS can be easily accessed through installing the software using conda-forge or pip. Please see also the *Installation Instructions <getting_started/installation>* provided.

Interoperable No commercial software is needed to read or alter the data provided in the repositories. Files containing code can be opened with any text editor, vector and raster data can be opened with open-source software such as QGIS or the respective Python libraries such as GeoPandas or Rasterio. Mesh data can also be opened using text editors or Python packages such as PyVista or open-source software like Blender. We mostly use file formats that are common to the geospatial community (.shp, .tif, ZMAP-Grids, etc.) and that are not proprietary.

Reusable The provision of tutorials, examples and in fact this documentation makes the data provided in the repositories reusable under the license provided below.

2.3 Citation

If you use GemGIS for any published work, please cite it using the reference below:

```
@article{Jüstel2022,
 doi = {10.21105/joss.03709},
 url = {https://doi.org/10.21105/joss.03709},
 year = {2022}, publisher = {The Open Journal},
 volume = {7},
 number = {73},
 pages = {3709},
 author = {Alexander Jüstel and Arthur Endlein Correira and Marius Pischke and Miguel de␣
↪la Varga and Florian Wellmann},
 title = {GemGIS - Spatial Data Processing for Geomodeling},
 journal = {Journal of Open Source Software}
 }
```

2.4 License

GNU LESSER GENERAL PUBLIC LICENSE Version 3, 29 June 2007

Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/> Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

This version of the GNU Lesser General Public License incorporates the terms and conditions of version 3 of the GNU General Public License, supplemented by the additional permissions listed below.

0. Additional Definitions.

As used herein, “this License” refers to version 3 of the GNU Lesser General Public License, and the “GNU GPL” refers to version 3 of the GNU General Public License.

“The Library” refers to a covered work governed by this License, other than an Application or a Combined Work as defined below.

An “Application” is any work that makes use of an interface provided by the Library, but which is not otherwise based on the Library. Defining a subclass of a class defined by the Library is deemed a mode of using an interface provided by the Library.

A “Combined Work” is a work produced by combining or linking an Application with the Library. The particular version of the Library with which the Combined Work was made is also called the “Linked Version”.

The "Minimal Corresponding Source" for a Combined Work means the Corresponding Source for the Combined Work, excluding any source code for portions of the Combined Work that, considered in isolation, are based on the Application, and not on the Linked Version.

The "Corresponding Application Code" for a Combined Work means the object code and/or source code for the Application, including any data and utility programs needed for reproducing the Combined Work from the Application, but excluding the System Libraries of the Combined Work.

1. Exception to Section 3 of the GNU GPL.

You may convey a covered work under sections 3 and 4 of this License without being bound by section 3 of the GNU GPL.

2. Conveying Modified Versions.

If you modify a copy of the Library, and, in your modifications, a facility refers to a function or data to be supplied by an Application that uses the facility (other than as an argument passed when the facility is invoked), then you may convey a copy of the modified version:

a) under this License, provided that you make a good faith effort to ensure that, in the event an Application does not supply the function or data, the facility still operates, and performs whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that the Library is used in it and that the Library and its use are covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license document.

c) For a Combined Work that displays copyright notices during execution, include the copyright notice for the Library among these notices, as well as a reference directing the user to the copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this License, and the Corresponding Application Code in a form suitable for, and under terms that permit, the user to recombine or relink the Application with a modified version of the Linked Version to produce a modified Combined Work, in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.

1) Use a suitable shared library mechanism for linking with the Library. A suitable mechanism is one that (a) uses at run time a copy of the Library already present on the user's computer system, and (b) will operate properly with a modified version of the Library that is interface-compatible with the Linked Version.

e) Provide Installation Information, but only if you would otherwise be required to provide such information under section 6 of the GNU GPL, and only to the extent that such information is necessary to install and execute a modified

version of the Combined Work produced by recombining or relinking the Application with a modified version of the Linked Version. (If you use option 4d0, the Installation Information must accompany the Minimal Corresponding Source and Corresponding Application Code. If you use option 4d1, you must provide the Installation Information in the manner specified by section 6 of the GNU GPL for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.

CHAPTER

THREE

INSTALLATION (TESTED 2023-09-01)

GemGIS is supported on Python version 3.10. Previous versions are officially not supported.

It is recommended to consider using Anaconda as a virtual environment and package manager for Python. The following installation instructions work with Anaconda.

3.1 Installing Anaconda

The latest Anaconda distribution can be installed from the Anaconda Website.

The latest Anaconda cheat sheet can be downloaded from the Anaconda Documentation.

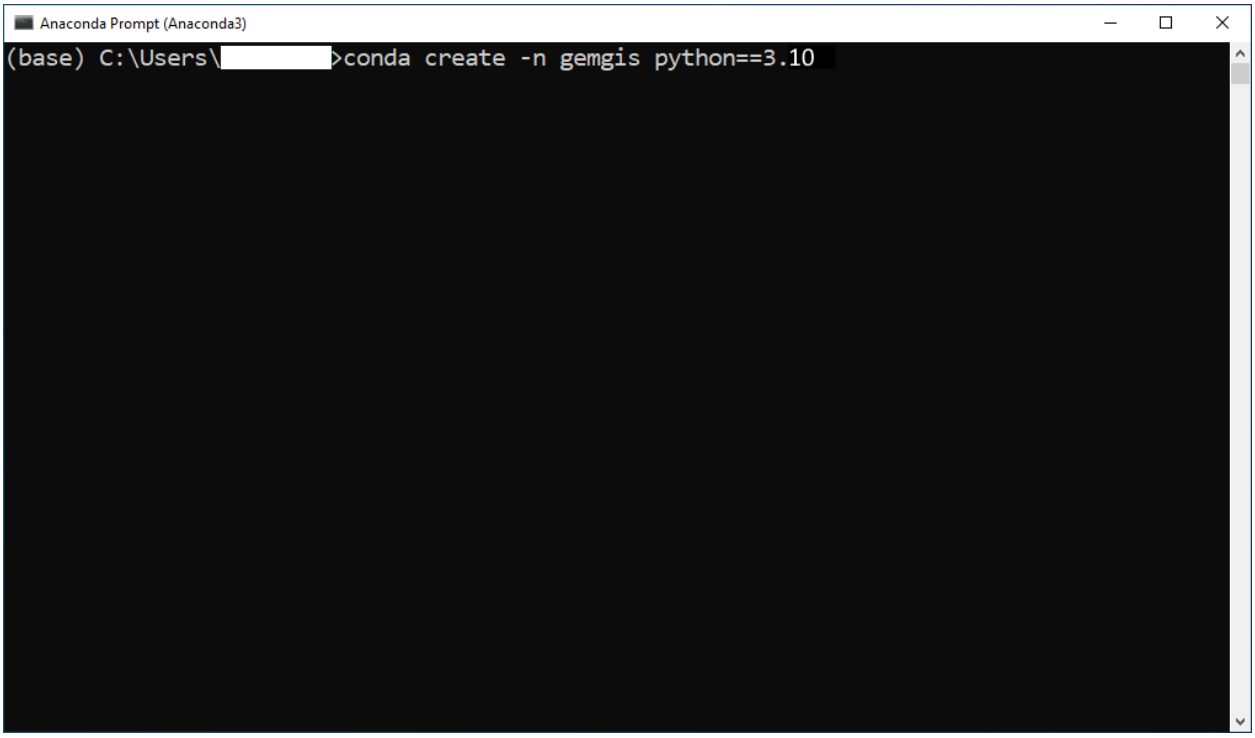
3.2 Creating virtual Environment

It is recommended to create a new virtual environment when using GemGIS to avoid conflicts with already existing projects. This step and all following steps will be performed within the `Anaconda Prompt (Anaconda3)`.

Creating a new environment in Anaconda with fixed Python version:

```
conda create -n gemgis python==3.10
```

Click `y` if you are asked to proceed.

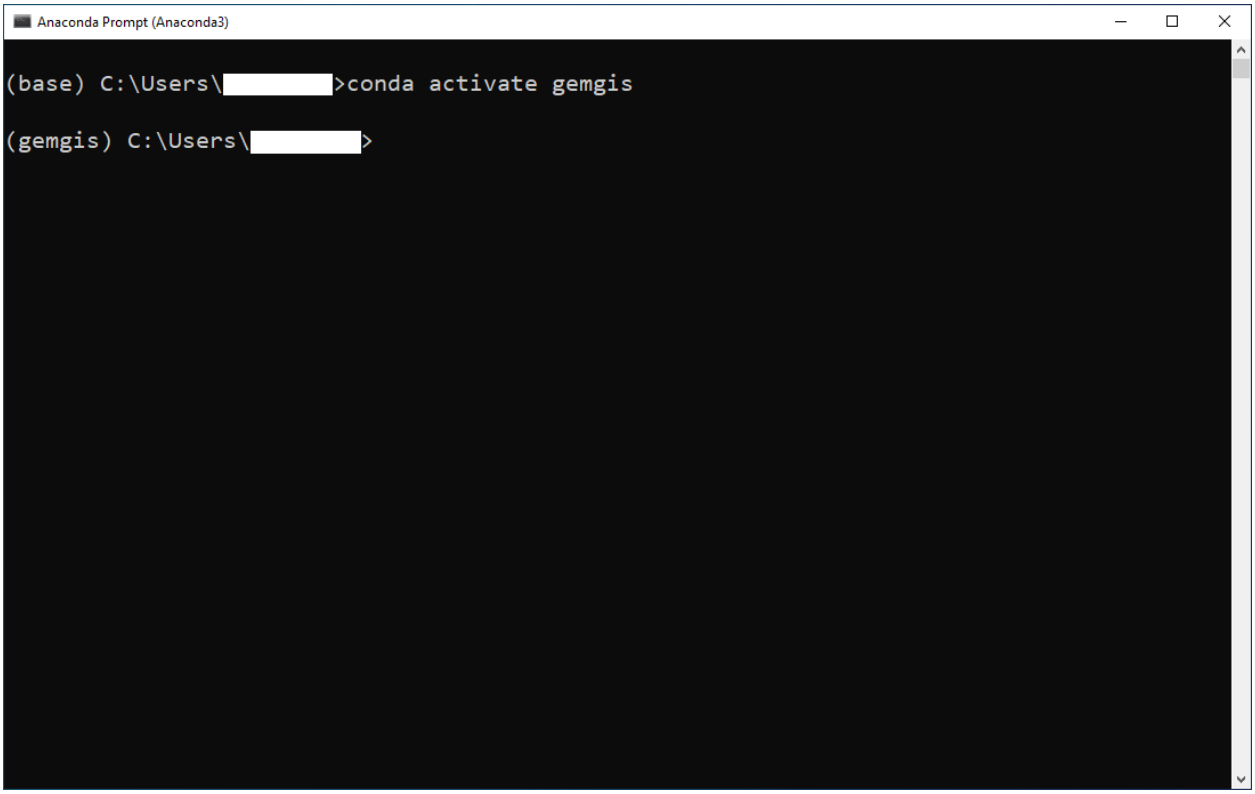


3.3 Activate virtual Environment

Activate the virtual environment:

```
conda activate gemgis
```

The gemgis environment now replaced the base environment which is indicated by `gemgis` in front of your path.



3.4 Setting channel priorities to avoid installation errors

As the installation of GeoPandas may result in an `OSError`, it is recommended to set the channel priorities when installing packages from conda.

Known error:

```
OSError: could not find or load spatialindex_c-64.dll
```

Solution:

```
conda config --env --add channels conda-forge
conda config --env --set channel_priority strict
```

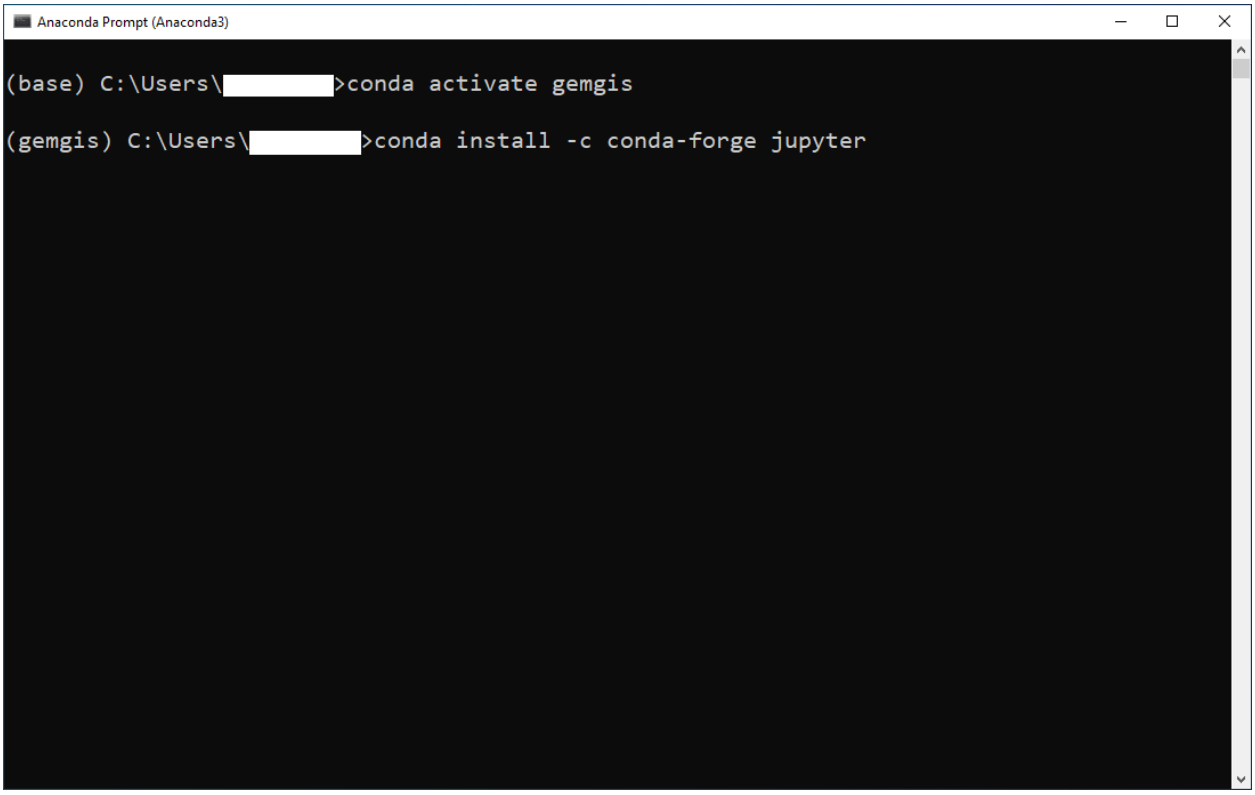
After configuring the channels, you can proceed with the installation of the packages and dependencies.

3.5 Installing Packages and Dependencies

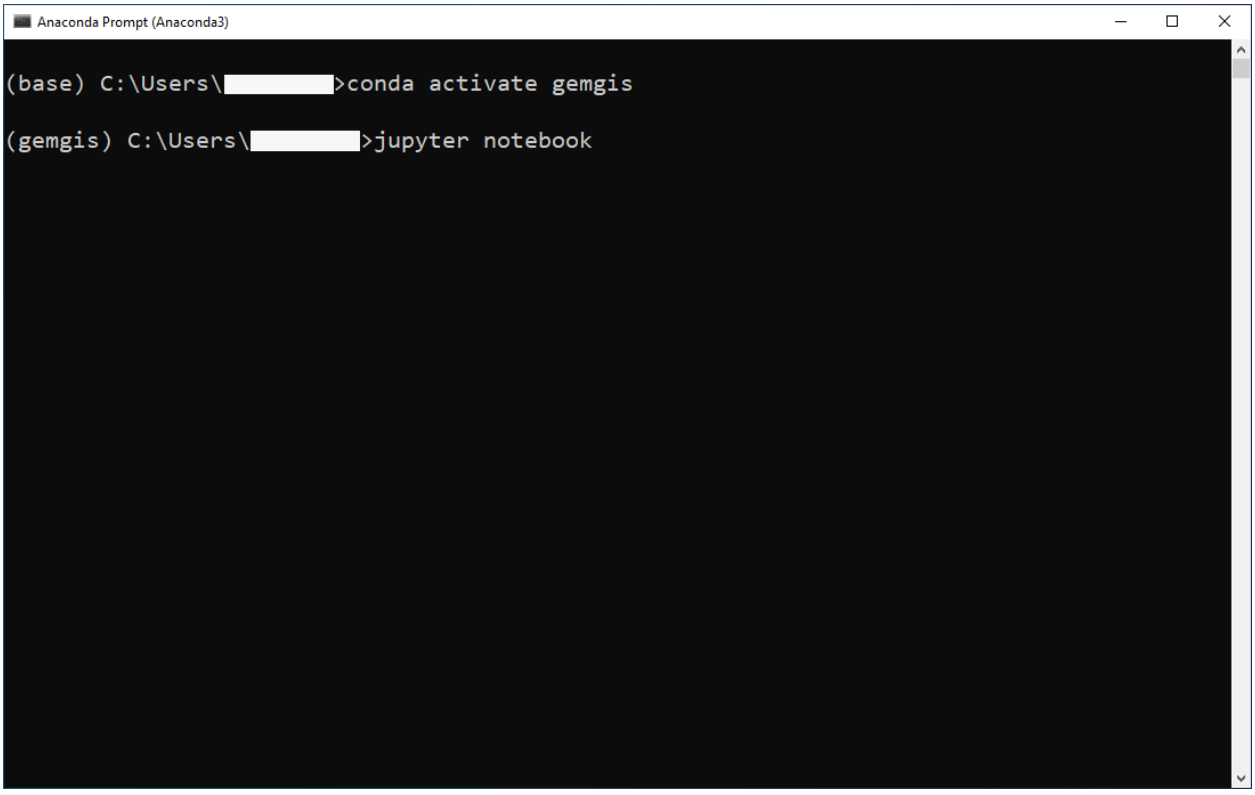
Several packages need to be installed in order to use GemGIS. It is recommended to use `Jupyter Notebooks` when working with GemGIS. Packages are installed using the `conda-forge` channel.

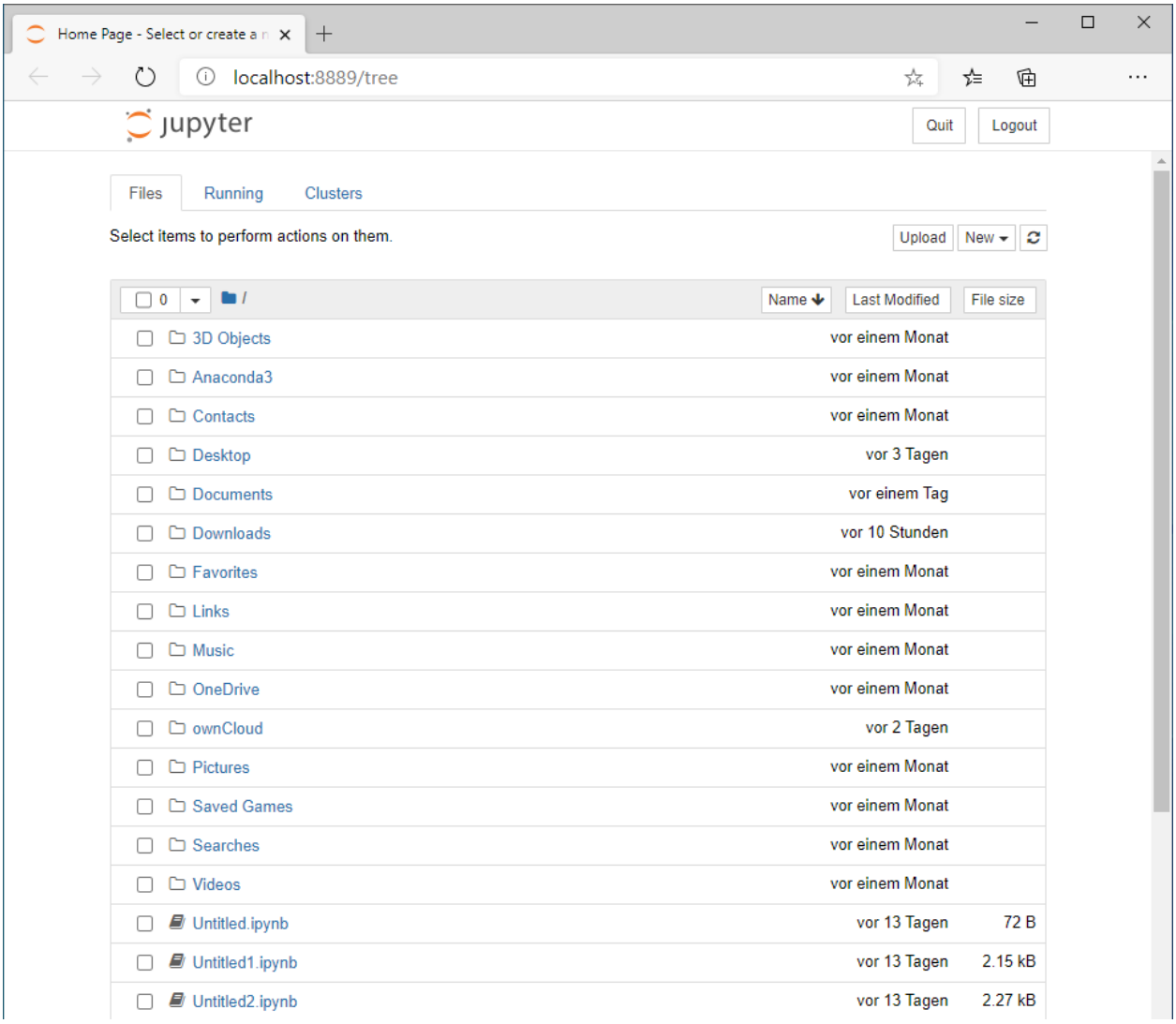
Install Jupyter Notebooks:

```
conda install -c conda-forge jupyter
```



You can start a new kernel by executing `jupyter notebook` in your Anaconda Prompt. A new kernel will then open in your browser.

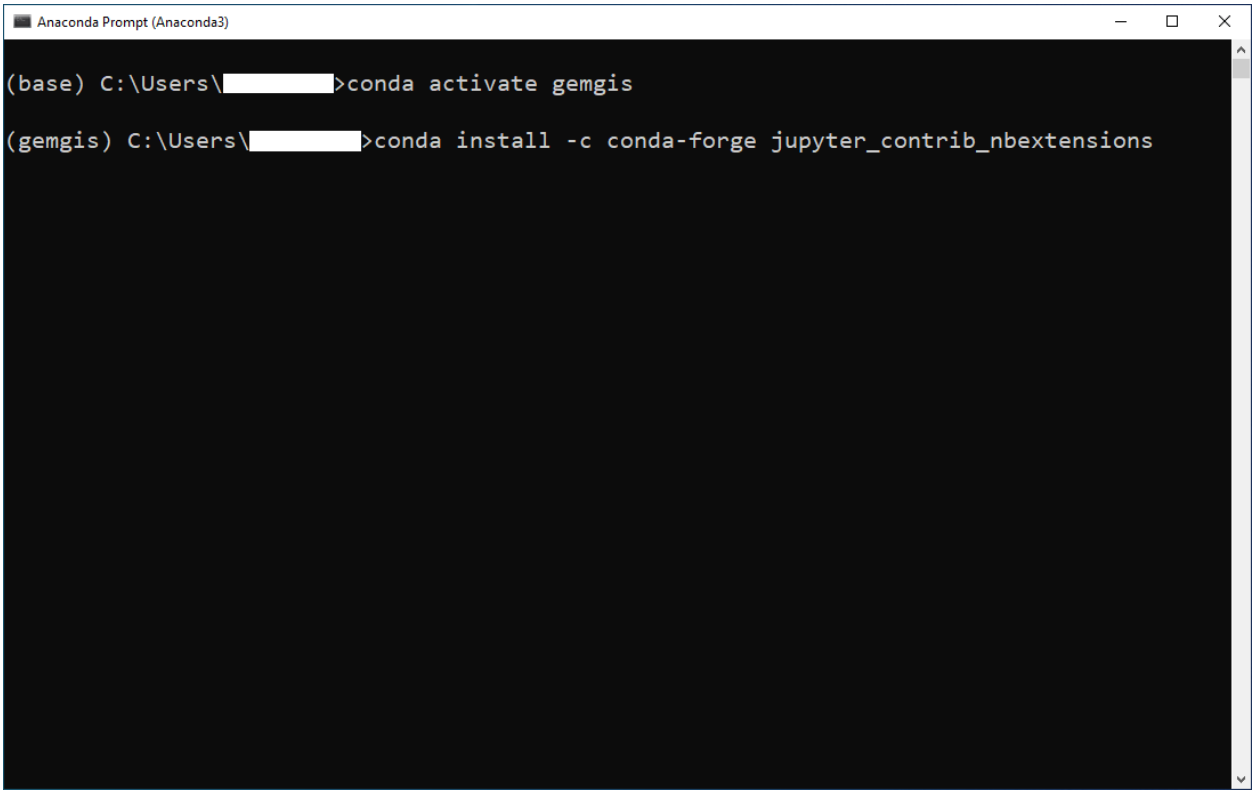




The `jupyter_contrib_nbextensions` package contains a collection of community-contributed unofficial extensions that add useful functionality to the Jupyter notebook:

```
conda install -c conda-forge jupyter_contrib_nbextensions
```

After installing the package, close and reopen your Anaconda prompt, activate your environment and open `jupyter notebook`. The Jupyter Notebook extensions are not needed in order to use GemGIS but add the additional functionalities to your notebooks.



3.6 Installing GemGIS via conda-forge

GemGIS and all its dependencies can be installed via conda-forge:

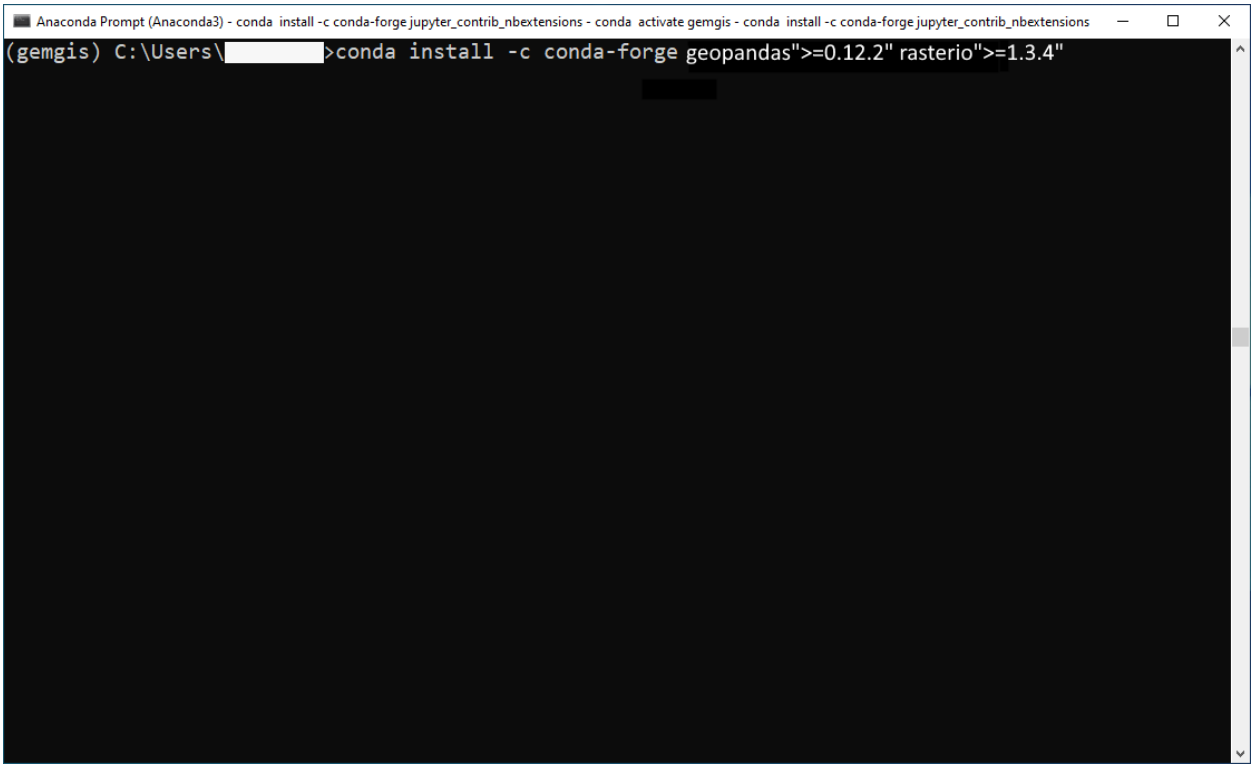
```
conda install -c conda-forge gemgis
```

3.7 Installing GemGIS's dependencies manually

Two of the main packages that GemGIS is dependent on are rasterio and GeoPandas. It is recommended to install these packages separately as they both depend on the GDAL translator library for raster and vector geospatial data. In addition, many smaller libraries like shaply or fiona will also be installed properly.

Install the latest versions of GeoPandas and Rasterio (as of 2023-09-01). Please mind the quotation marks that are necessary when specifying the version numbers.:

```
conda install -c conda-forge geopandas">=0.13.2" rasterio">=1.3.8"
```



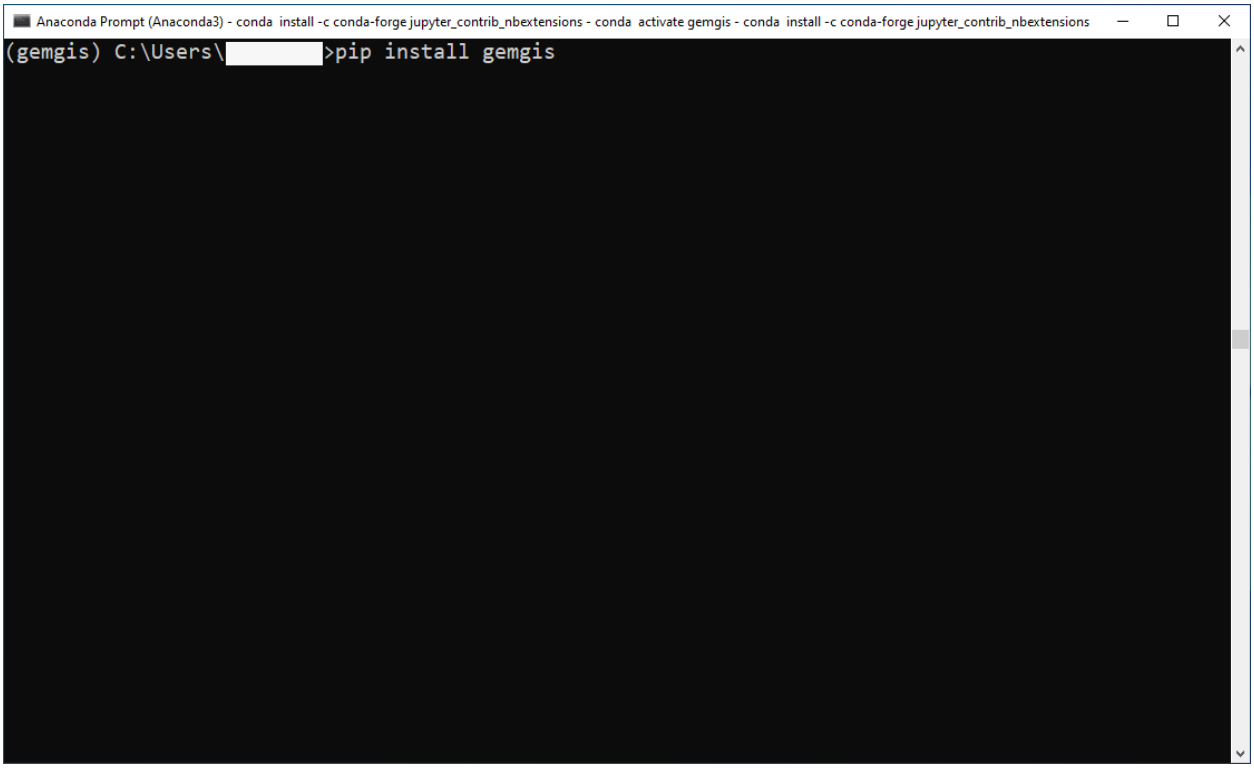
In order to visualize and work with meshes, the PyVista package is being installed:

```
conda install -c conda-forge pyvista">=0.42."
```

3.8 Installing GemGIS via PyPi

The latest stable version of GemGIS can be downloaded from PyPi:

```
pip install gemgis
```

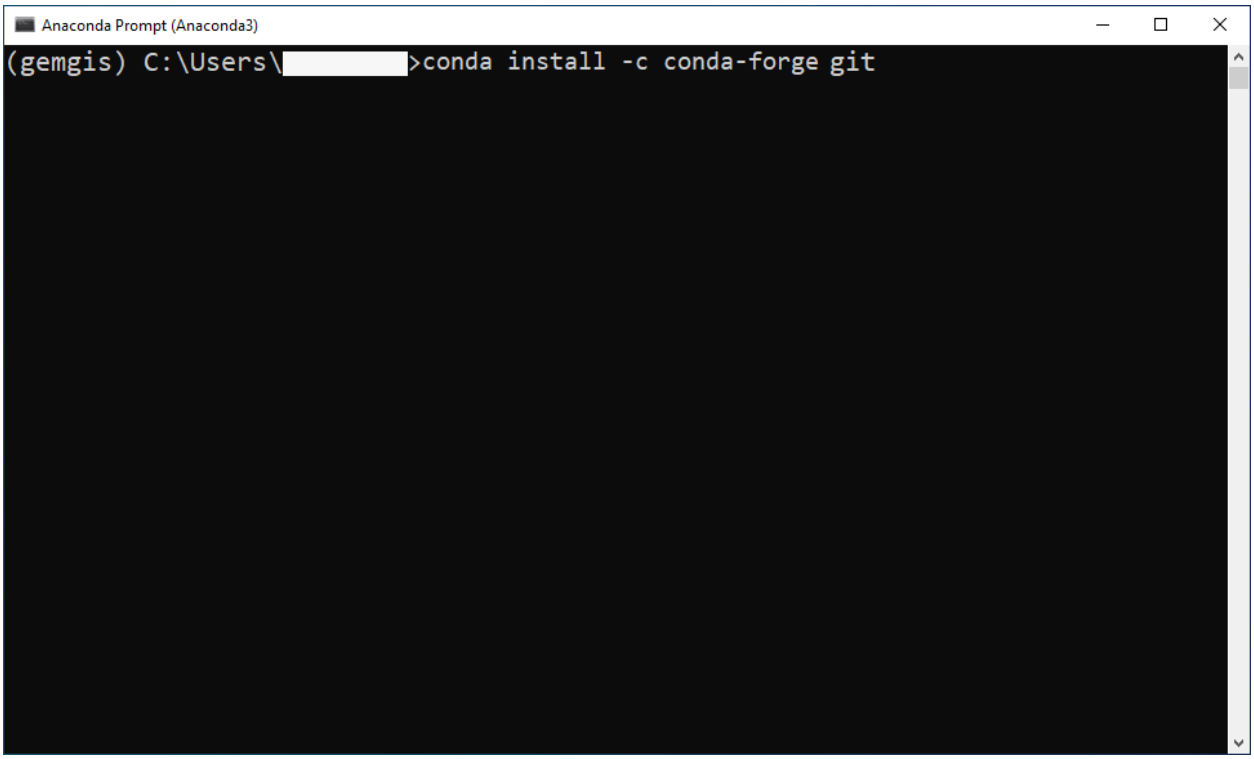


3.9 Installing GemGIS from the Repository

Alternatively, GemGIS can also be cloned from the GemGIS Github repository. You can either download the package from the repository or download it with `git`. Git can either be installed in the current conda environment or by downloading a third party programme like Git Bash.

Installing git in the current conda environment:

```
conda install -c conda-forge git
```



A folder where GemGIS is being stored needs to be selected when cloning/installing GemGIS from the Github repository. When opening the Anaconda Prompt the first time, the default path should be within the folder of the current user. Create an empty folder with your file explorer at a destination where you want to save GemGIS. In this case, GemGIS will be stored in a folder called `gemgis` within the `Documents` folder.

Navigate to the GemGIS folder:

```
cd Documents/gemgis
```

Use the following command to go up one level in your folder structure:

```
cd ..
```

Now, a new git repository needs to be initiated within the gemgis folder:

```
git init
```

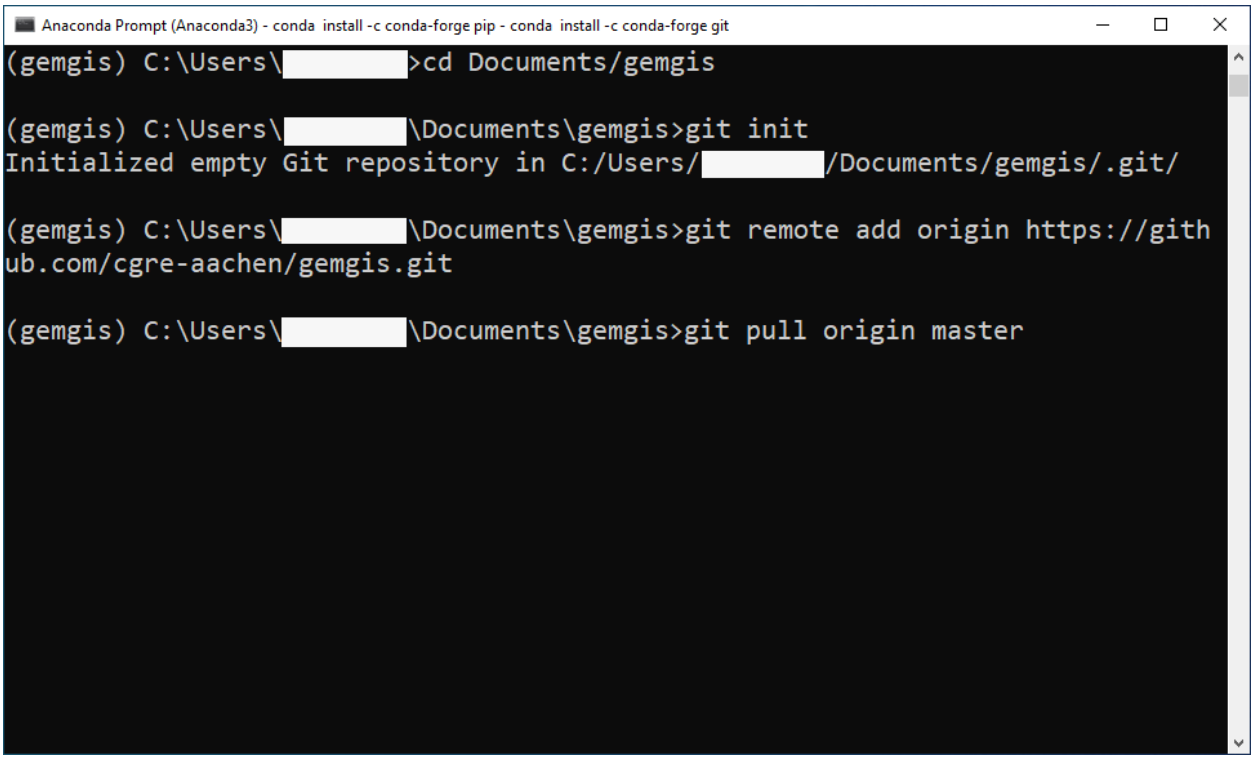
Linking the remote repository to the local repository:

```
git remote add origin https://github.com/cgre-aachen/gemgis.git
```

Download the latest version of GemGIS from the main branch:

```
git pull origin main
```

You should now have the latest GemGIS files in your gemgis folder.



3.10 Installing GemPy (Optional)

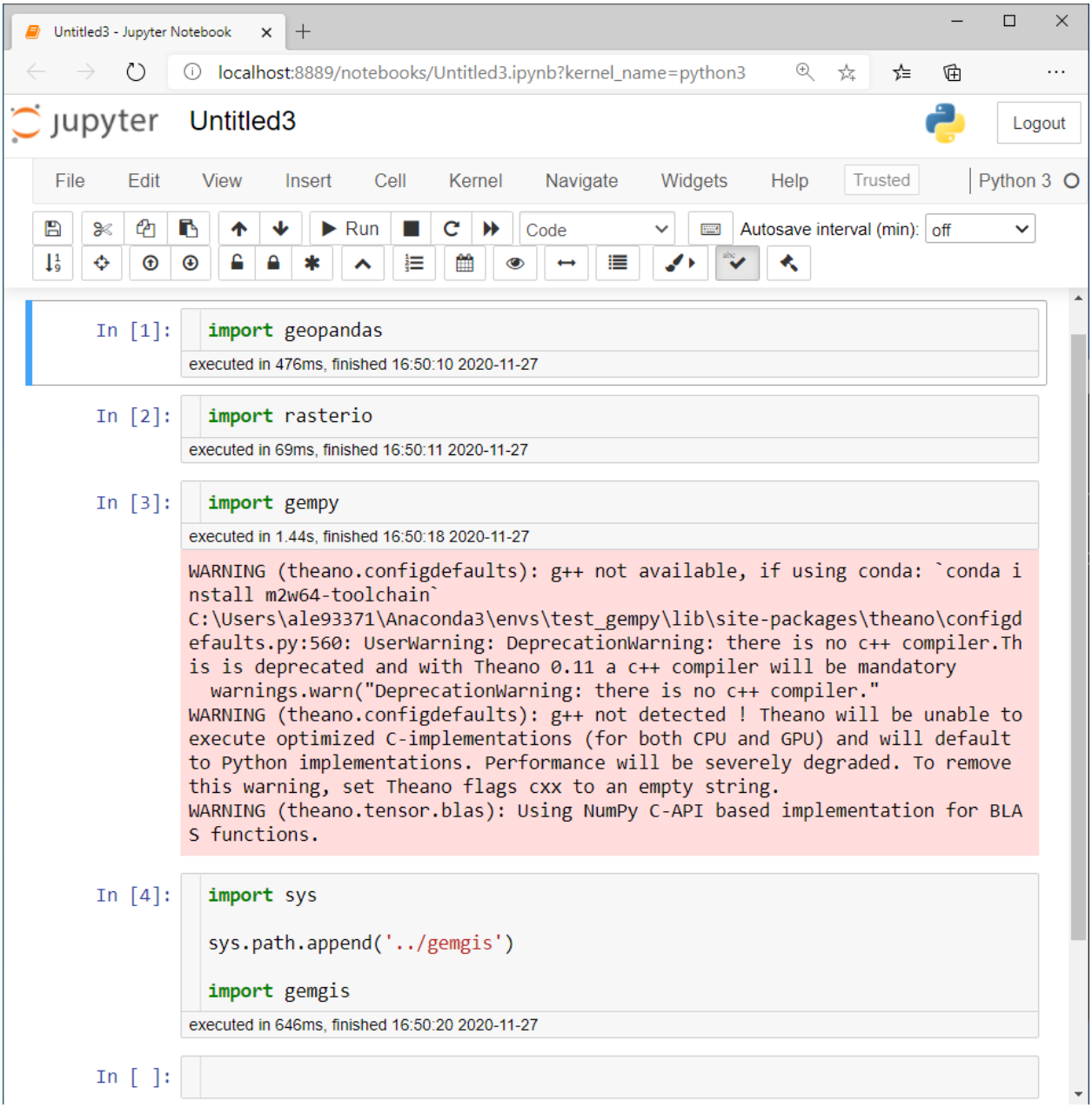
The current modeling package that is supported by GemGIS is GemPy. GemPy can easily be installed with `pip`/PyPi:

```
pip install gempy
```

3.11 Checking the Installation

Before starting to work with GemGIS, it is recommended to check that all packages have been installed successfully.

- Open a new Anaconda prompt
- Activate your gemgis environment: `conda activate gemgis`
- Start a new Jupyter kernel: `jupyter notebook`
- Either navigate to one of the provided notebooks that come with GemGIS if you cloned the repository or create a new notebook
- In the notebook, run `import rasterio`
- Run `import geopandas as gpd`
- Run `import gempy as gp`
- Run `import gemgis as gg` if you installed it via pip
- If you cloned the repository, you have to `import sys`, append the path to the local repository using `sys.path.append('..your/path/to/the/repo/gemgis')` before `import gemgis as gg`
- Install missing dependencies via the Anaconda Prompt using `pip install package name`.


Untitled3 - Jupyter Notebook
localhost:8889/notebooks/Untitled3.ipynb?kernel_name=python3
jupyter Untitled3
Logout
File Edit View Insert Cell Kernel Navigate Widgets Help Trusted Python 3
Run Code Autosave interval (min): off
In [1]: import geopandas
executed in 476ms, finished 16:50:10 2020-11-27
In [2]: import rasterio
executed in 69ms, finished 16:50:11 2020-11-27
In [3]: import gempy
executed in 1.44s, finished 16:50:18 2020-11-27
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and with Theano 0.11 a c++ compiler will be mandatory
warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute optimized C-implementations (for both CPU and GPU) and will default to Python implementations. Performance will be severely degraded. To remove this warning, set Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
In [4]: import sys
sys.path.append('../gemgis')
import gemgis
executed in 646ms, finished 16:50:20 2020-11-27
In [ ]:

CHAPTER

FOUR

CONTRIBUTING

Thank you for considering to contribute to GemGIS!

As GemGIS is an open-source and a community-driven project, we need people like you to make the package even better, simpler to use for beginners, and to extend its functionality for more advanced users. There are several ways to contribute to GemGIS:

- Submitting bug report or requests for new features
- Adding new or expanding existing tutorials and examples
- Fixing typos and improving documentation
- Adding new functionality to the package

4.1 How to get started with GemGIS?

- Work through the examples and tutorials or work with your own data.
- If you run into a problem and you think it is not related to GemGIS directly but rather to the installation or your data or something just does not work, have a look at our Github Discussion to ask your question. Maybe someone already had the same issue before and the answer is already available.
- If you run into a problem and you think it is related to GemGIS directly please open a new issue on Github. Our template will guide you through the necessary information.
- If you work with a dataset and you think something is missing in GemGIS please open a new feature request on Github. Our template will guide you through the necessary information. We will then try to implement the requested feature or support you in contributing yourself to the package.
- If you already have code for a new feature/tutorial/example, we would appreciate it if you open a pull request and contribute this way. Our team will then support you in terms of testing your code, suggesting improvements and adding tutorials or examples for your implemented feature.
- If you find typos or other error in our README or the documentation open a new issue or fix the error right away by opening a pull request.

4.2 How to get in touch with the GemGIS developers?

You can reach the developers of GemGIS either by opening a new issue on Github (feature request or bug report), open a pull request to contribute directly to the package, ask a question in the Github Discussions forum or via the Slack workspace of the Software Underground.

4.3 How to contribute code?

Is this your first contribution?

Please take a look at these resources to learn about git and pull requests but do not hesitate to ask questions if you get stuck:

- How to Contribute to Open Source.
- Aaron Meurer's tutorial on the git workflow.
- How to Contribute to an Open Source Project on GitHub.

4.3.1 General guidelines

We follow the git pull request workflow to make changes to our codebase. Every change made goes through a pull request, even our own, so that our continuous integration services (Github Actions) have a change to check that the code is up to standards and passes all our tests. This way, the *main* branch is always stable.

4.3.2 General guidelines for pull requests (PRs)

- **Open an issue first** describing what you want to do. If there is already an issue that matches your PR, leave a comment there instead to let us know what you plan to do.
- Each pull request should consist of a **small** and logical collection of changes.
- Larger changes should be broken down into smaller components and integrated separately.
- Bug fixes should be submitted in separate PRs.
- Describe what your PR changes and *why* this is a good thing. Be as specific as you can. The PR description is how we keep track of the changes made to the project over time.
- Do not commit changes to files that are irrelevant to your feature or bugfix (eg: *.gitignore*, IDE project files, etc).
- Write descriptive commit messages. Chris Beams has written a guide on how to write good commit messages.
- Be willing to accept criticism and work on improving your code; we don't want to break other users' code, so care must be taken not to introduce bugs.
- Be aware that the pull request review process is not immediate, and is generally proportional to the size of the pull request.

4.3.3 Setting up your environment

We highly recommend using Anaconda and the *conda* package manager to install and manage your Python packages. It will make your life a lot easier!

The repository includes a conda environment file *environment_dev.yml* with the specification for all development requirements to build and test the project. Once you have forked and clone the repository to your local machine, you use this file to create an isolated environment on which you can work.

4.3.4 Docstrings

All docstrings should follow the numpy style guide. All functions/classes/methods should have docstrings with a full description of all arguments and return values.

To play nicely with Jupyter and IPython, **keep docstrings limited to 79 characters** per line. We don't have a good way of enforcing this automatically yet, so please do your best.

4.3.5 Testing your code

Automated testing helps ensure that our code is as free of bugs as it can be. It also lets us know immediately if a change we make breaks any other part of the code.

All of our test code and data are stored in the *tests* subpackage. We use the pytest framework to run the test suite.

Please write tests for your code so that we can be sure that it won't break any of the existing functionality. Tests also help us be confident that we won't break your code in the future.

If you're **new to testing**, see existing test files for examples of things to do. **Don't let the tests keep you from submitting your contribution!** If you're not sure how to do this or are having trouble, submit your pull request anyway. We will help you create the tests and sort out any kind of problem during code review.

4.3.6 Documentation

Most documentation sources are in the *docs* folder. We use sphinx to build the web pages from these sources. To build the HTML files:

```
cd docs
make all
```

This will build the HTML files in *docs/_build/html*. Open *docs/_build/html/index.html* in your browser to view the pages.

The API reference is manually assembled in *docs/api_reference/index.rst*. The *autodoc* sphinx extension will automatically create pages for each function/class/module listed there.

4.3.7 Code Review

After you've submitted a pull request, you should expect to hear at least a comment within a couple of days. We may suggest some changes or improvements or alternatives.

Some things that will increase the chance that your pull request is accepted quickly:

- Write a good and detailed description of what the PR does.
- Write tests for the code you wrote/modified.
- Readable code is better than clever code (even with comments).
- Write documentation for your code (docstrings) and leave comments explaining the *reason* behind non-obvious things.
- Include an example of new features in the gallery or tutorials.
- Follow the PEP8 style guide for code and the numpy guide for documentation.

Pull requests will automatically have tests run by Github Actions. Github will show the status of these checks on the pull request. Try to get them all passing (green). If you have any trouble, leave a comment in the PR or get in touch with us

4.3.8 Attribution

This contributing document is largely based upon the work by the Fatiando a Terra project.

CHAPTER

FIVE

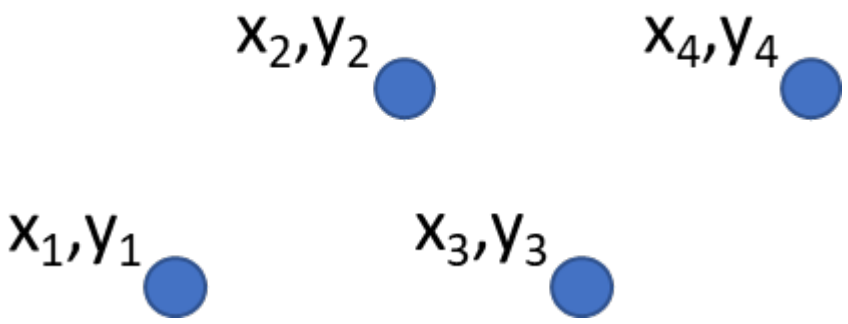
WHAT IS WHAT?

5.1 What is vector data?

Vector data provides a way to represent real world features within a GIS environment. A feature is anything you can see on the landscape. Imagine you are standing on the top of a hill. Looking down, you can see houses, roads, trees, rivers, and so on. Each one of these things would be a feature when we represent them in a GIS application. Vector features have attributes, which consist of text or numerical information that describe the features.

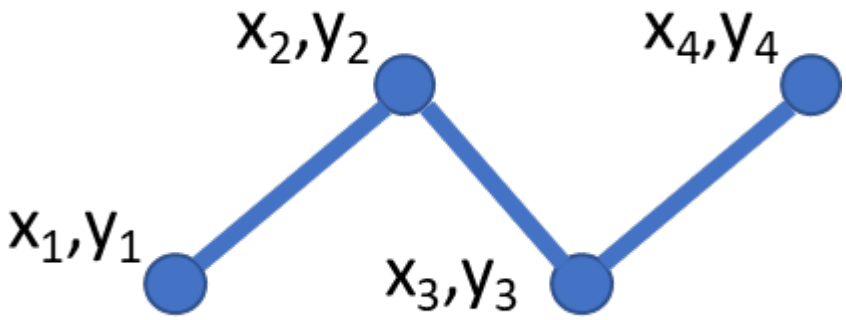
5.1.1 Point Data

A point feature has an X, Y and optionally, Z value. The X and Y values will depend on the Coordinate Reference System (CRS) being used. For now, let us simply say, that a CRS is a way to accurately describe where a particular place is on the earth's surface. One of the most common reference systems is Longitude and Latitude. Lines of Longitude run from the North Pole to the South Pole. Lines of Latitude run from the East to West. You can describe precisely where you are at any place on the earth by giving someone your Longitude (X) and Latitude (Y). Since we know the earth is not flat, it is often useful to add a Z value to a point feature. This describes how high above sea level you are.



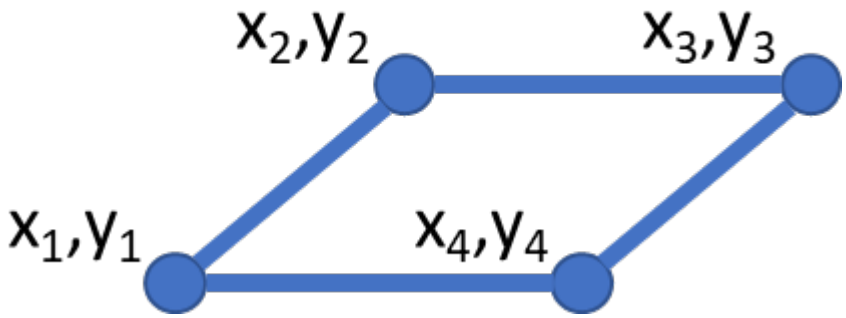
5.1.2 Line Data

A line or poly-/multiline is used to show the geometry of linear features such as roads, rivers, contours, footpaths, flight paths and so on. Sometimes, we have special rules for polylines in addition to their basic geometry. For example, contour lines may touch (e.g. at a cliff face) but should never cross over each other. Similarly, polylines used to store a road network should be connected at intersections. In some GIS applications, you can set these special rules for a feature type (e.g. roads) and the GIS will ensure that these polylines always comply to these rules.



5.1.3 Polygon Data

Polygon features are enclosed areas like dams, islands, country boundaries and so on. Like polyline features, polygons are created from a series of vertices that are connected with a continuous line. However, because a polygon always describes an enclosed area, the first and last vertices should always be at the same place! Polygons often have shared geometry boundaries that are in common with a neighbouring polygon. Many GIS applications have the capability to ensure that the boundaries of neighbouring polygons exactly coincide.



5.1.4 Working with vector data in GemGIS

Vector data is one of the fundamental data types used in `GemGIS`. Vector data is handled by the `Shapely` package for single geometries and by the `PyGEOS` package for arrays of geometries and vectorized operations performed on these arrays.

Different `BaseGeometries` are defined:

- Points/Multi-Points
- Lines/Multi-Lines
- Polygons/Multi-Polygons
- Geometry Collections

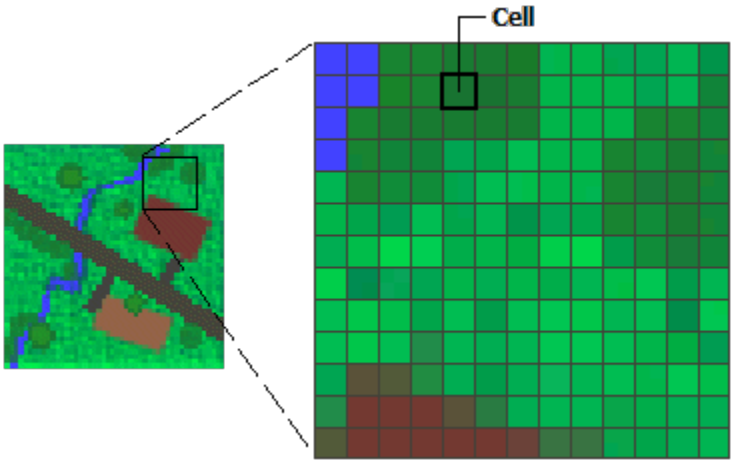
The coordinates of the vertices of each element are not associated with a coordinate reference system (CRS) at this point. This is added when collecting multiple objects in a `GeoPandas` `GeoDataFrame`. This `Pandas` `DataFrame`-like object consists of a `geometry` column holding the different geometric objects and data columns equal to the attribute fields of shape files. A CRS is usually predefined when loading shape files consisting of geometric objects with `GeoPandas` or can be assigned when creating a new `GeoDataFrame`. It is then possible to perform coordinate transformations on

`GeoDataFrames` holding a CRS attribute. Different datasets with the same coordinate reference system can the plotted to visualize the spatial distribution of the data.

	formation	geometry	X	Y
0	Sand1	POINT (0.256 264.862)	0.26	264.86
1	Sand1	POINT (10.593 276.734)	10.59	276.73
2	Sand1	POINT (17.135 289.090)	17.13	289.09
3	Sand1	POINT (19.150 293.313)	19.15	293.31
4	Sand1	POINT (27.795 310.572)	27.80	310.57

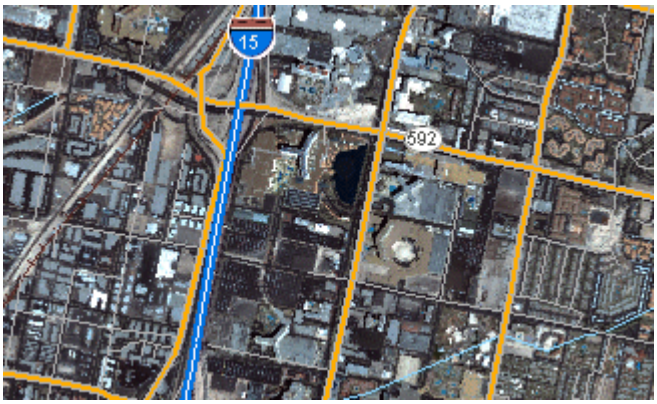
5.2 What is raster data?

In its simplest form, a raster consists of a matrix of cells (or pixels) organized into rows and columns (or a grid) where each cell contains a value representing information, such as the elevation value. Rasters are digital aerial photographs, imagery from satellites, digital pictures, or even scanned maps.



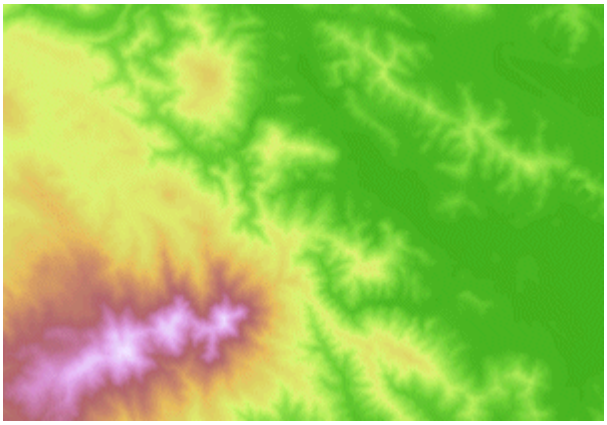
5.2.1 Rasters as base maps

A common use of raster data in a GIS is as a background display for other feature layers. Three main sources of raster basemaps are orthophotos from aerial photography, satellite imagery, and scanned maps.



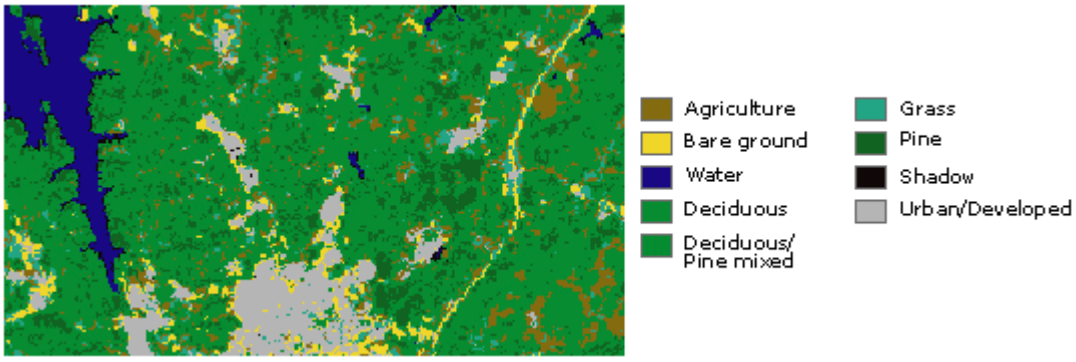
5.2.2 Rasters as surface maps

Rasters are well suited for representing data that changes continuously across a landscape (surface). Elevation values measured from the earth's surface are the most common application of surface maps, but other values, such as rainfall, temperature, concentration, and population density, can also define surfaces that can be spatially analyzed.



5.2.3 Rasters as thematic maps

Rasters representing thematic data can be derived from analyzing other data. A common analysis application is classifying a satellite image by land-cover categories. For example, you can process data through a geoprocessing model to create a raster dataset that maps suitability for a specific activity.



Source: https://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/what-is-raster-data.htm

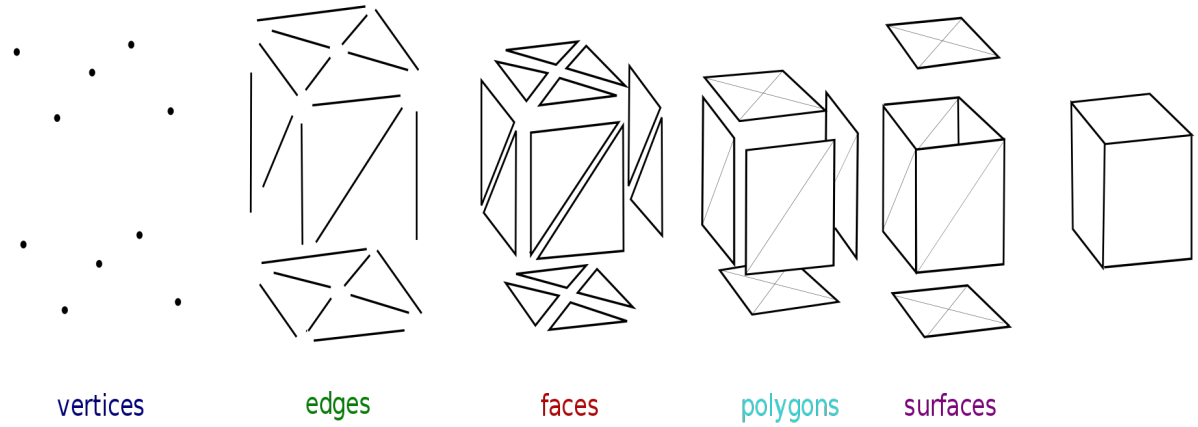
5.2.4 Working with raster data in GemGIS

Raster data is one of the fundamental data types used in `GemGIS`. Raster data is handled by the `Rasterio` package and the data itself by `NumPy`.

5.3 What is a mesh?

A mesh is a collection of vertices, edges and faces that defines the shape of a polyhedral object.

- **Vertices**: Representing the x, y and z coordinates/position of corner points of the mesh
- **Edges**: Representing the straight/linear connection between two vertices
- **Faces**: Representing a closed set of edges. Three edges/vertices make up a triangle, four or more make up a polygon. Both polygons and triangles are coplanar, indicating that are located in the same plane.
- **Surfaces**: Representing a collection of faces making up a non-coplanar surface.



Source: https://en.wikipedia.org/wiki/Polygon_mesh

5.3.1 Working with mesh data in GemGIS

Mesh data is one of the fundamental data types used in `GemGIS`. Mesh data is handled by the `PyVista` package.

`PyVista` can work with `PolyData` objects and grids (`StructuredGrid`, `UnstructuredGrid`) consisting of cells and points. Data arrays can be associated with the object holding information such as elevation or depth values, labels or in case of `MultiBlock` data the single blocks.

5.4 What are projections?

A map projection is a tool to flatten to flatten a globe's surface into a 2D map. This requires a to transform spherical longitude and latitude coordinates from the surface on the globe into X and Y coordinates on a map. The surface of the globe is being distorted when projecting it from 3D to 2D.

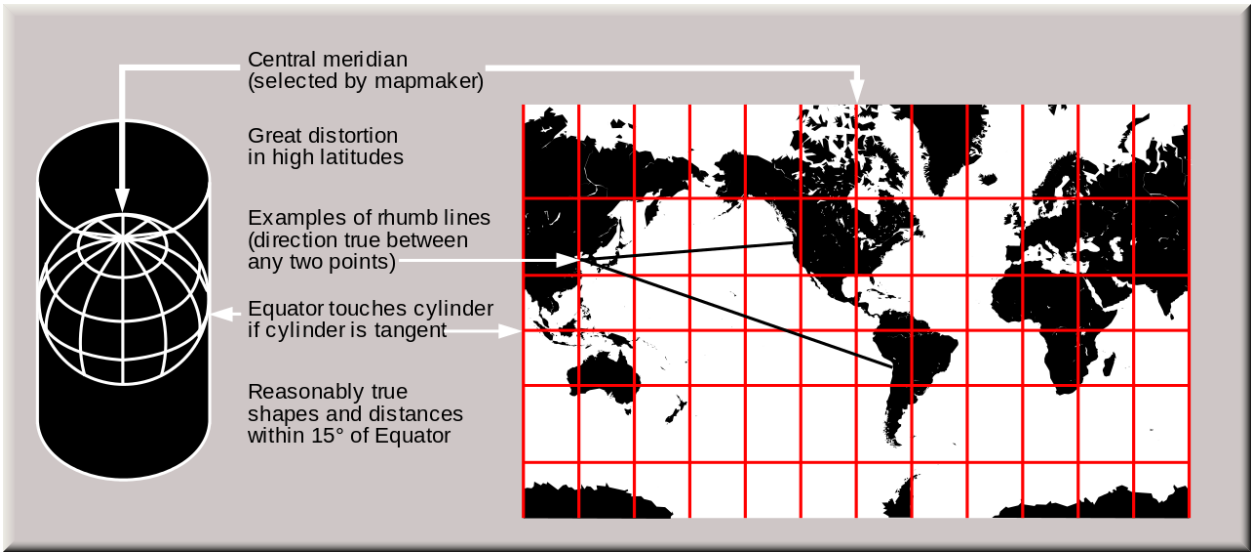
Different map projections can be classified depending on the type of projection surface onto which the globe is conceptually projected or according to the properties the model preserves.

5.4.1 Projections by Surface

Projections by different surfaces describe how the globe is projected onto a certain projection surface. These surfaces are cylindrical, conic and plane and will be introduced in the following.

Cylindrical Projections

A normal cylindrical projection is any projection in which meridians are mapped to equally spaced vertical lines and circles of latitude (parallels) are mapped to horizontal lines. The mapping of meridians to vertical lines can be visualized by imagining a cylinder whose axis coincides with the Earth's axis of rotation. This cylinder is wrapped around the Earth, projected onto, and then unrolled.



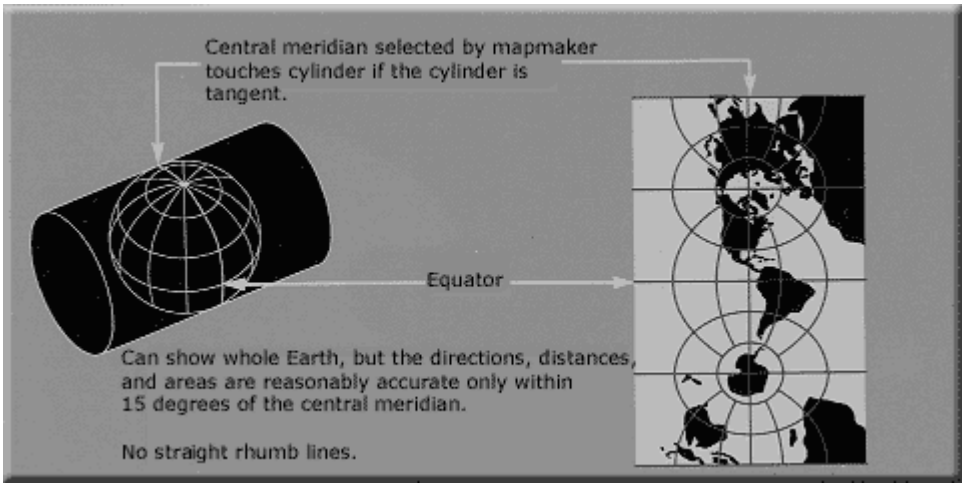
Credit: U.S. Geological Survey

Mercator Projection

The widely known Mercator projection is an example for a cylindrical projection and has become the standard map for navigation. It is representing north as up and south as down while preserving local directions and shapes. The map is conformal (see Classification by Properties of the Model the Projections preserve for more details). As a side effect, the Mercator projection inflates the size of objects away from the equator. This inflation is very small near the equator but accelerates with increasing latitude to become infinite at the poles. So, for example, landmasses such as Greenland and Antarctica appear far larger than they actually are relative to landmasses near the equator, such as Central Africa.

Transverse Mercator Projection

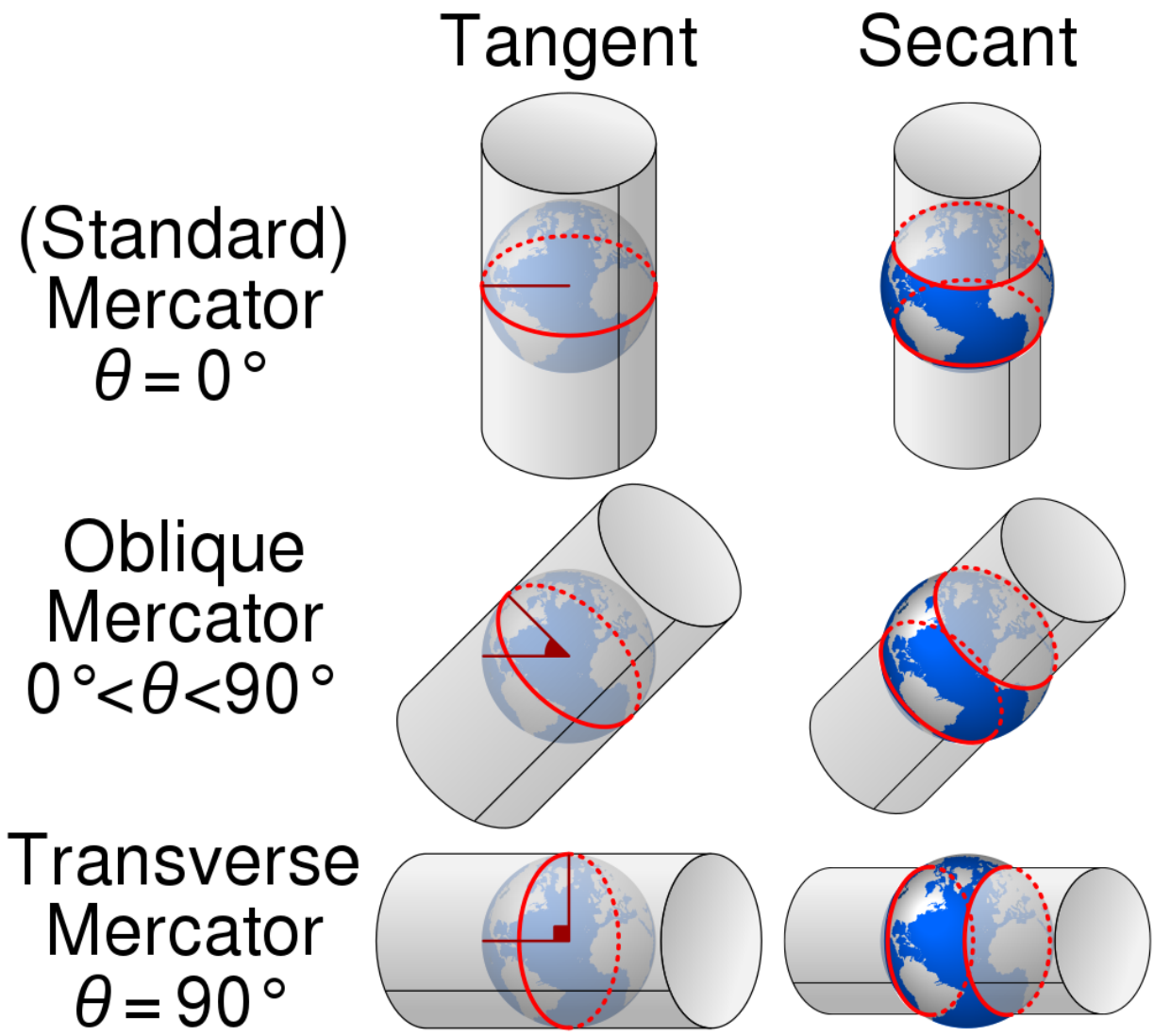
The transverse Mercator map projection is an adaptation of the standard Mercator projection. The transverse version is widely used in national and international mapping systems around the world, including the Universal Transverse Mercator. The transverse Mercator projection is the transverse aspect of the standard (or Normal) Mercator projection. They share the same underlying mathematical construction and consequently the transverse Mercator inherits many traits from the normal Mercator.



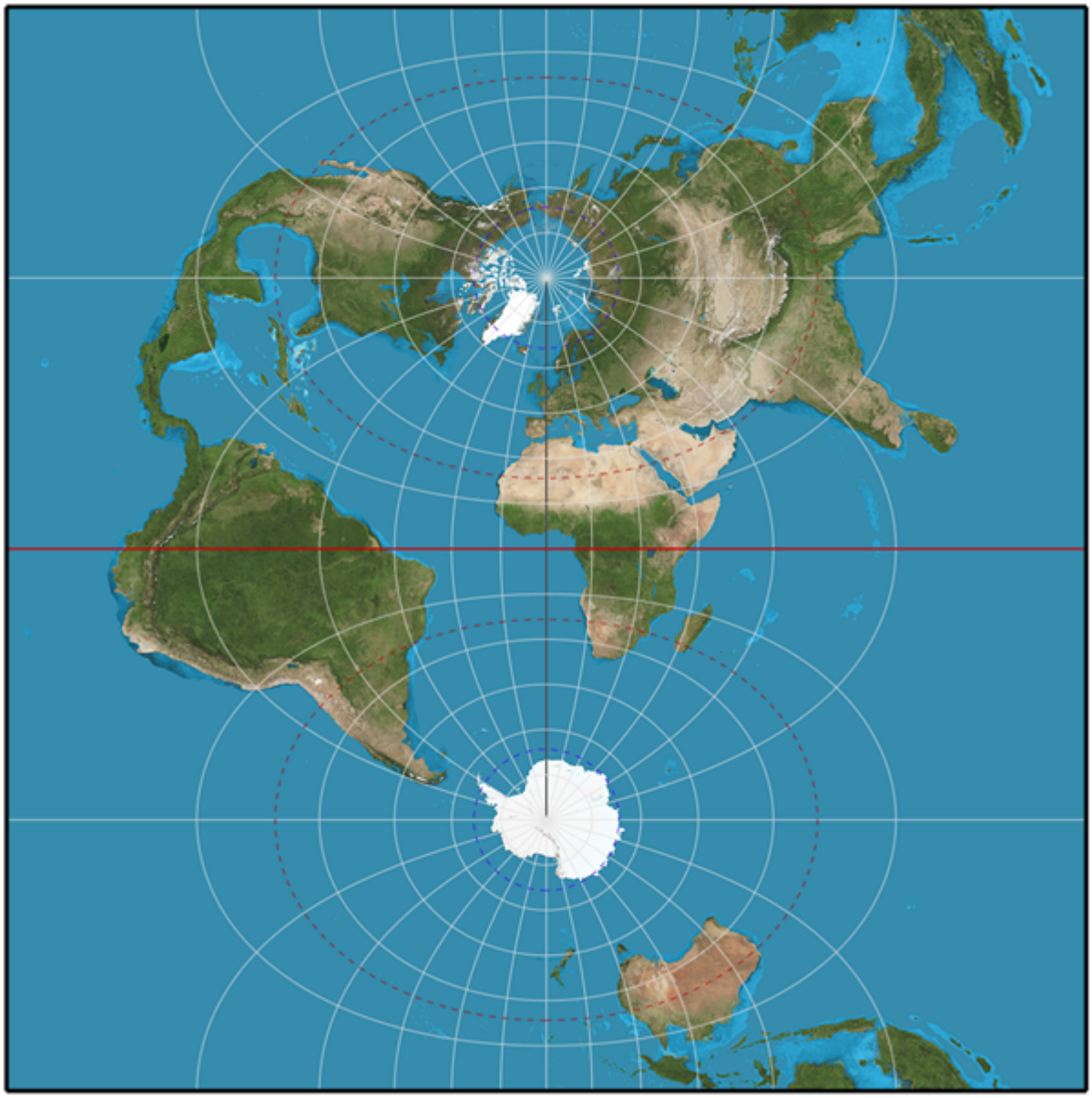
Credit: U.S. Geological Survey

Comparison between Normal and Transverse Mercator Projection

- Both projections are cylindrical: for the Normal Mercator, the axis of the cylinder coincides with the polar axis and the line of tangency with the equator. For the transverse Mercator, the axis of the cylinder lies in the equatorial plane, and the line of tangency is any chosen meridian, thereby designated the central meridian.
- Both projections may be modified to secant forms, which means the scale has been reduced so that the cylinder slices through the model globe.
- Both exist in spherical and ellipsoidal versions.
- Both projections are conformal, so that the point scale is independent of direction and local shapes are well preserved;
- Both projections have constant scale on the line of tangency (the equator for the normal Mercator and the central meridian for the transverse).



License: CC BY-SA 3.0

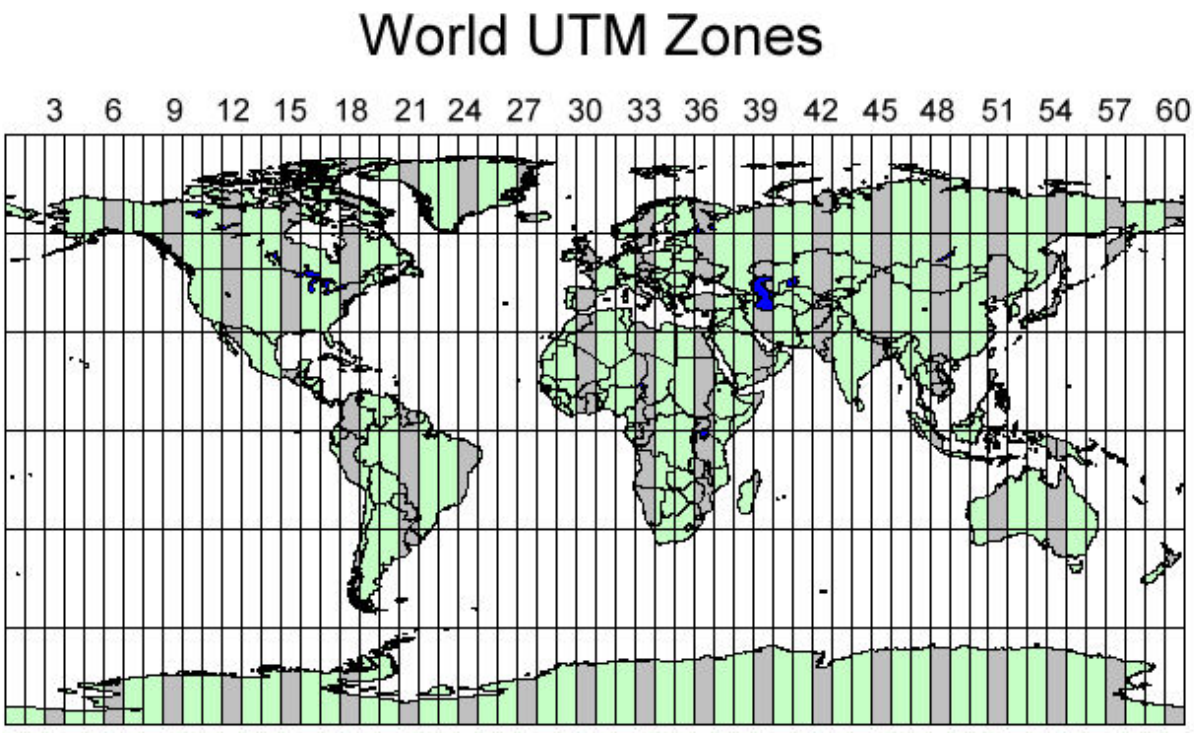


License: CC BY-SA 3.0

Universal Transverse Mercator Coordinate System

The Universal Transverse Mercator (UTM) is a system for assigning coordinates to locations on the surface of the Earth. Like the traditional method of latitude and longitude, it is a horizontal position representation, which means it ignores altitude and treats the earth as a perfect ellipsoid. However, it differs from global latitude/longitude in that it divides earth into 60 zones and projects each to the plane as a basis for its coordinates. Specifying a location means specifying the zone and the x, y coordinate in that plane. The projection from spheroid to a UTM zone is some parameterization of the transverse Mercator projection. The parameters vary by nation or region or mapping system.

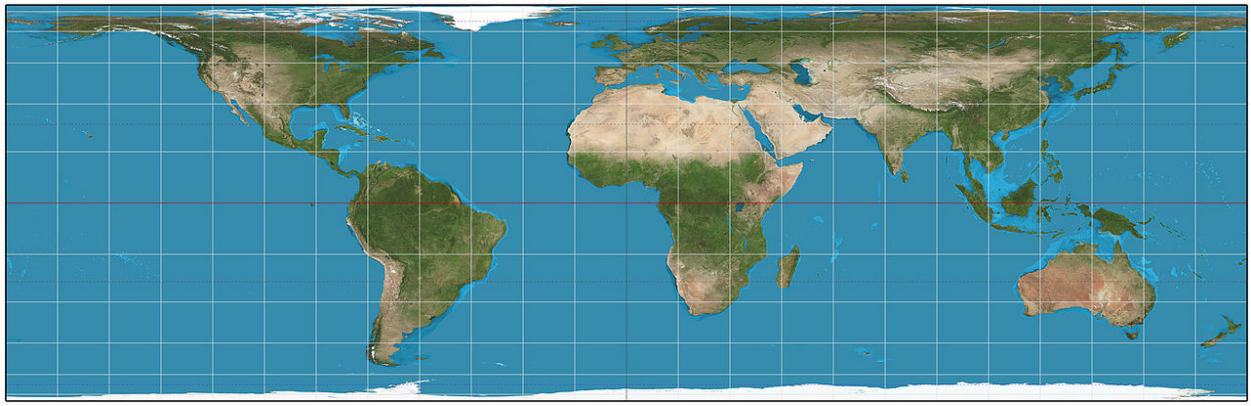
Most zones in UTM span 6 degrees of longitude, and each has a designated central meridian. The scale factor at the central meridian is specified to be 0.9996 of true scale for most UTM systems in use.



Credit: https://www.xmswiki.com/wiki/File:UTM_world_no_Image_Map.jpg

Lambert cylindrical equal-area Projection

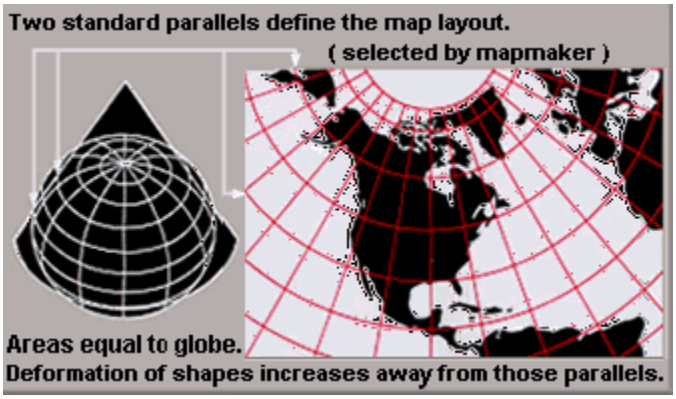
The Lambert projection is a second example for a cylindrical projection which is also an equal-area projection (see Classification by Properties of the Model the Projections preserve for more details). This projection is undistorted along the equator, which is its standard parallel, but distortion increases rapidly towards the poles. Like any cylindrical projection, it stretches parallels increasingly away from the equator. The poles accrue infinite distortion, becoming lines instead of points.



License: CC BY-SA 3.0

Conic Projections

The term "conic projection" is used to refer to any projection in which meridians are mapped to equally spaced lines radiating out from the apex and circles of latitude (parallels) are mapped to circular arcs centered on the apex. When making a conic map, the map maker arbitrarily picks two standard parallels. Those standard parallels may be visualized as secant lines where the cone intersects the globe—or, if the map maker chooses the same parallel twice, as the tangent line where the cone is tangent to the globe. The resulting conic map has low distortion in scale, shape, and area near those standard parallels. Distances along the parallels to the north of both standard parallels or to the south of both standard parallels are stretched; distances along parallels between the standard parallels are compressed. When a single standard parallel is used, distances along all other parallels are stretched.



Credit: U.S. Geological Survey

Lambert conformal conic Projection

A Lambert conformal conic projection (LCC) is a conic map projection used for aeronautical charts, portions of the State Plane Coordinate System, and many national and regional mapping systems. Conceptually, the projection seats a cone over the sphere of the Earth and projects the surface conformally onto the cone. The cone is unrolled, and the parallel that was touching the sphere is assigned unit scale. That parallel is called the reference parallel or standard parallel.

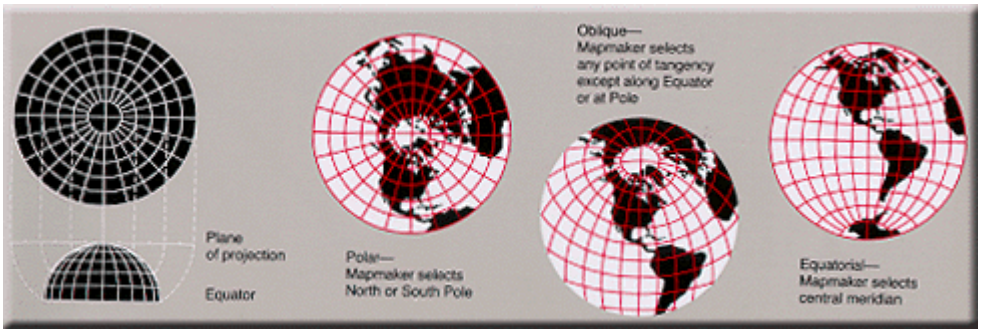
By scaling the resulting map, two parallels can be assigned unit scale, with scale decreasing between the two parallels and increasing outside them. This gives the map two standard parallels. In this way, deviation from unit scale can be minimized within a region of interest that lies largely between the two standard parallels. Unlike other conic projections, no true secant form of the projection exists because using a secant cone does not yield the same scale along both standard parallels.

License: CC BY-SA 3.0

Azimuthal Projections

Azimuthal projections have the property that directions from a central point are preserved and therefore great circles through the central point are represented by straight lines on the map. These projections also have radial symmetry in the scales and hence in the distortions: map distances from the central point are computed by a function r(d) of the true distance d, independent of the angle; correspondingly, circles with the central point as center are mapped into circles which have as center the central point on the map.

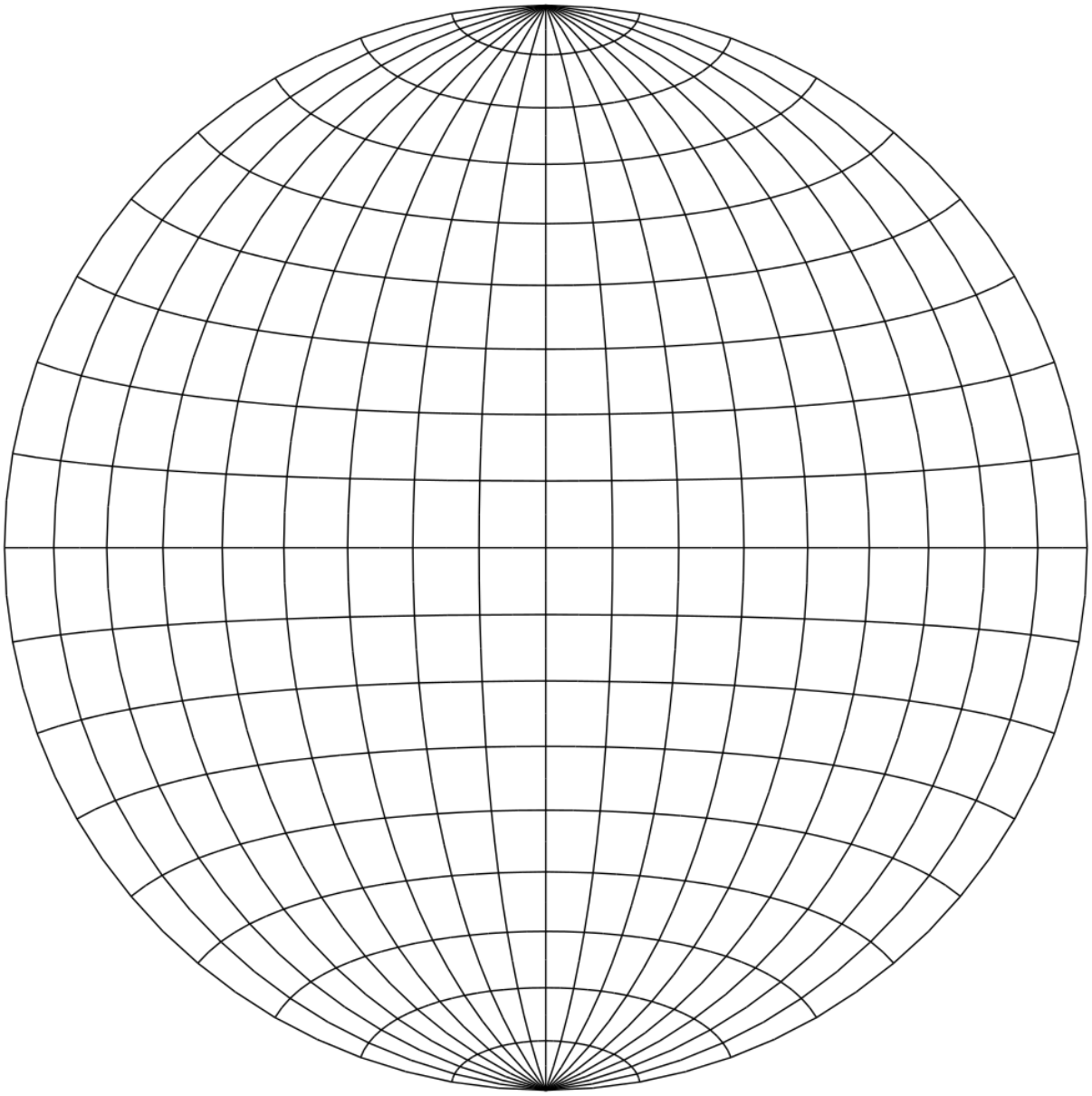
The mapping of radial lines can be visualized by imagining a plane tangent to the Earth, with the central point as tangent point.



Credit: U.S. Geological Survey

Schmidt Net

The Schmidt net is a manual drafting method for the Lambert azimuthal equal-area projection using graph paper. It results in one lateral hemisphere of the Earth with the grid of parallels and meridians. It is used in structural geology to visualize orientation values.



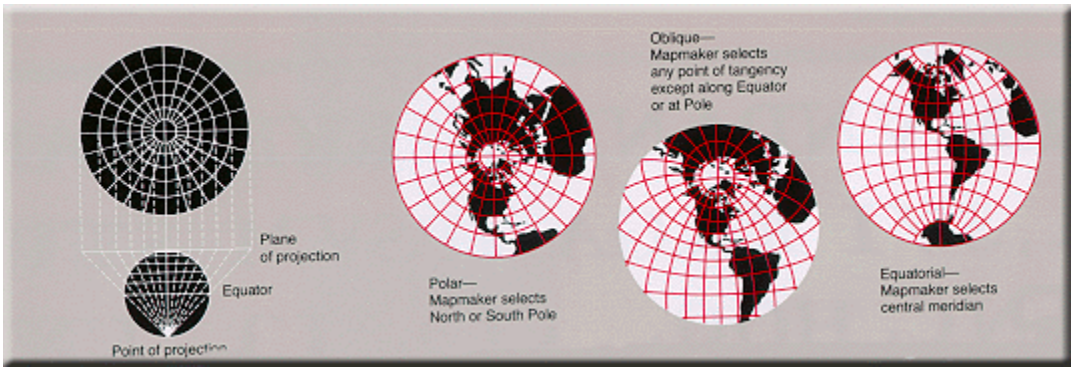
5.4.2 Projections by preservation of a property

Conformal Projections - Preserving shapes locally

Conformal, or orthomorphic, map projections preserve angles locally, implying that they map infinitesimal circles of constant size anywhere on the Earth to infinitesimal circles of varying sizes on the map. In contrast, mappings that are not conformal distort most such small circles into ellipses of distortion. An important consequence of conformality is that relative angles at each point of the map are correct, and the local scale (although varying throughout the map) in every direction around any one point is constant.

Examples for conformal projections include:

- Mercator projection
- Lambert projection



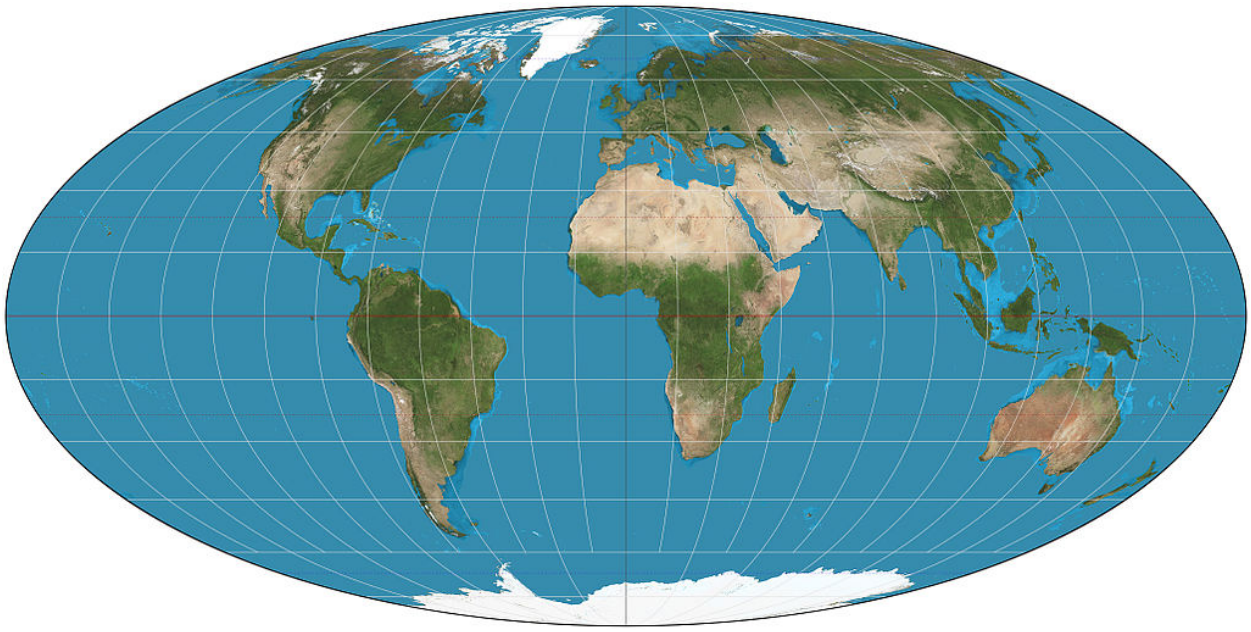
Credit: U.S. Geological Survey

Equal-area Projections - Preserving area

Equal-area maps preserve area measure, generally distorting shapes in order to do that. Equal-area maps are also called equivalent or authalic.

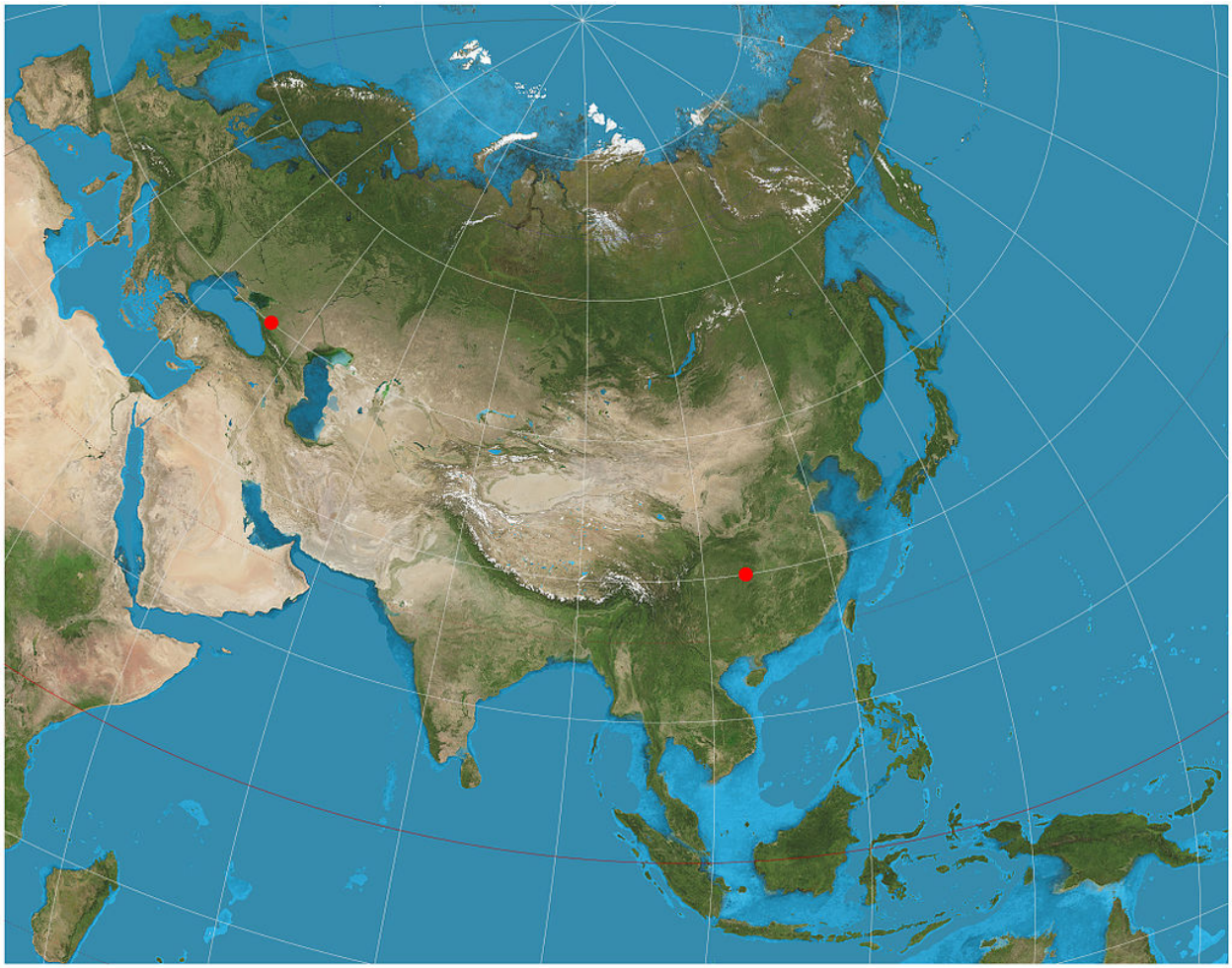
Examples of equal-area projections include:

- Lambert azimuthal projection (Schmidt Net)
- Lambert cylindrical projection
- Mollweide projection



Equidistant Projections - Preserving distance

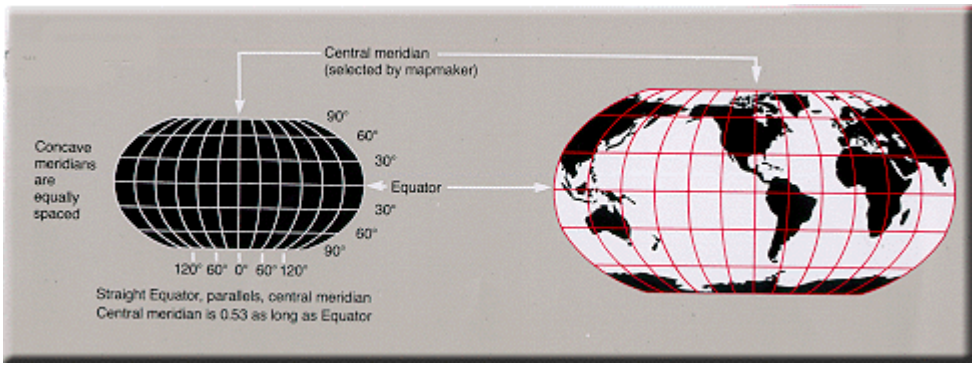
If the length of the line segment connecting two projected points on the plane is proportional to the geodesic (shortest surface) distance between the two unprojected points on the globe, then we say that distance has been preserved between those two points. An equidistant projection preserves distances from one or two special points to all other points. The special point or points may get stretched into a line or curve segment when projected. In that case, the point on the line or curve segment closest to the point being measured to must be used to measure the distance.



License: CC BY-SA 3.0

Other Projections

Other projections include gnomonic projections, retroazimuthal and compromise projections (e.g. Robinson projection). Compromise projections for instance give up the idea of perfectly preserving metric properties, seeking instead to strike a balance between distortions, or to simply make things look right. Most of these types of projections distort shape in the polar regions more than at the equator.



Credit: U.S. Geological Survey

5.4.3 Working with projections and coordinate reference systems in GemGIS

The most common task in `GemGIS` is to transform or reproject coordinates from one Coordinate Reference System (CRS) to another one. Each CRS represents a coordinate-based local, regional or global system used to locate geographical entities. A Coordinate Reference System defines a specific map projection, as well as transformations between different spatial reference systems. Within `GemGIS`, this is done by the `pyproj` package also utilized by `GeoPandas` for instance.

5.4.4 Notation

- Aspect: The aspect describes how the developable surface is placed relative to the globe: it may be normal (such that the surface's axis of symmetry coincides with the Earth's axis), transverse (at right angles to the Earth's axis) or oblique (any angle in between)
- Tangent: Tangent means the surface touches but does not slice through the globe
- Secant: Secant means the surface does slice through the globe

Text and Image Sources:

- https://en.wikipedia.org/wiki/Map_projection
- https://en.wikipedia.org/wiki/Mercator_projection
- https://en.wikipedia.org/wiki/Transverse_Mercator_projection
- https://en.wikipedia.org/wiki/Lambert_cylindrical_equal-area_projection
- https://en.wikipedia.org/wiki/Lambert_conformal_conic_projection
- https://en.wikipedia.org/wiki/Schmidt_net
- https://en.wikipedia.org/wiki/Spatial_reference_system

5.5 What interpolation methods are used?

CHAPTER

SIX

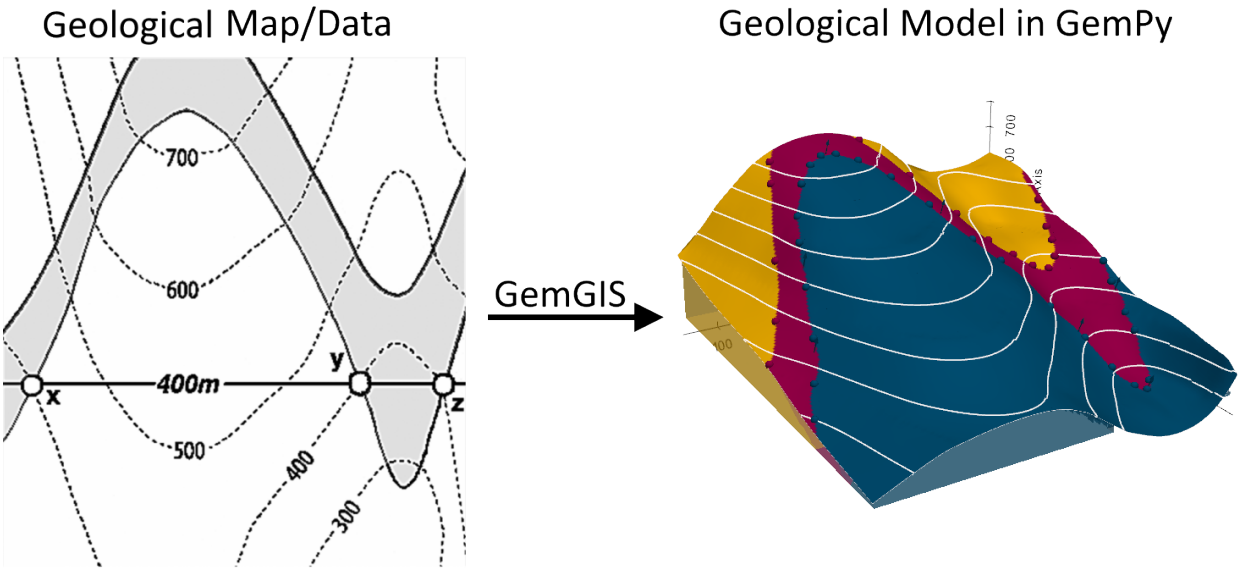
TUTORIALS AND BASIC USAGE

There is a series of tutorials available for GemGIS. In order to keep the size of the main GemGIS package as small as possible, the data is provided through a separated repository gemgis-data. You can also download the data directly following this link.

Watch our Video Tutorials: *link to be provided soon*

The following subsections elaborate on the basic API usage of GemGIS. This includes the extraction of information from input data files, the creation of new data and the preparation of data for the geomodeling with `GemPy`. The respective reading or loading functions of packages such as `GeoPandas` or `rasterio` will be used to load the data as `GeoDataFrame` or rasterio object.

The aim of this and the upcoming tutorials is to demonstrate how to prepare spatial data for geomodeling with *GemPy* to get a geological model. Raw data is usually created/digitized within QGIS which we encourage the user to use!



Source: Powell, D. (1995): Interpretation geologischer Strukturen durch Karten - Eine praktische Anleitung mit Aufgaben und Lösungen, page 15, figure 10 A, Springer Verlag Berlin, Heidelberg, New York, ISBN: 978-3-540-58607-4.

Each set of functions of GemGIS is collected in a different module. The functions of each module can be accessed as followed:

```
import gemgis as gg

data = gg.vector.function_name(...)
```

(continues on next page)

(continued from previous page)

```
data = gg.raster.function_name(...)

data = gg.visualization.function_name(...)

data = gg.web.function_name(...)

data = gg.utils.function_name(...)

data = gg.misc.functions_name(...)
```

6.1 00 Generating Data in QGIS for GemGIS

Much of the data that is used in GemGIS may already be available as vector or raster data. This data is then usually visualized in a GIS system, a geoinformation system. GIS systems can be used to create, edit, organize, analyse or present spatial data. Two of the most common GIS systems used in Earth Sciences are the open-source package QGIS and the commercial software ArcGIS. It is highly recommended to download and install QGIS if you do not have an ArcGIS license at hand.

The following sections will demonstrate how to

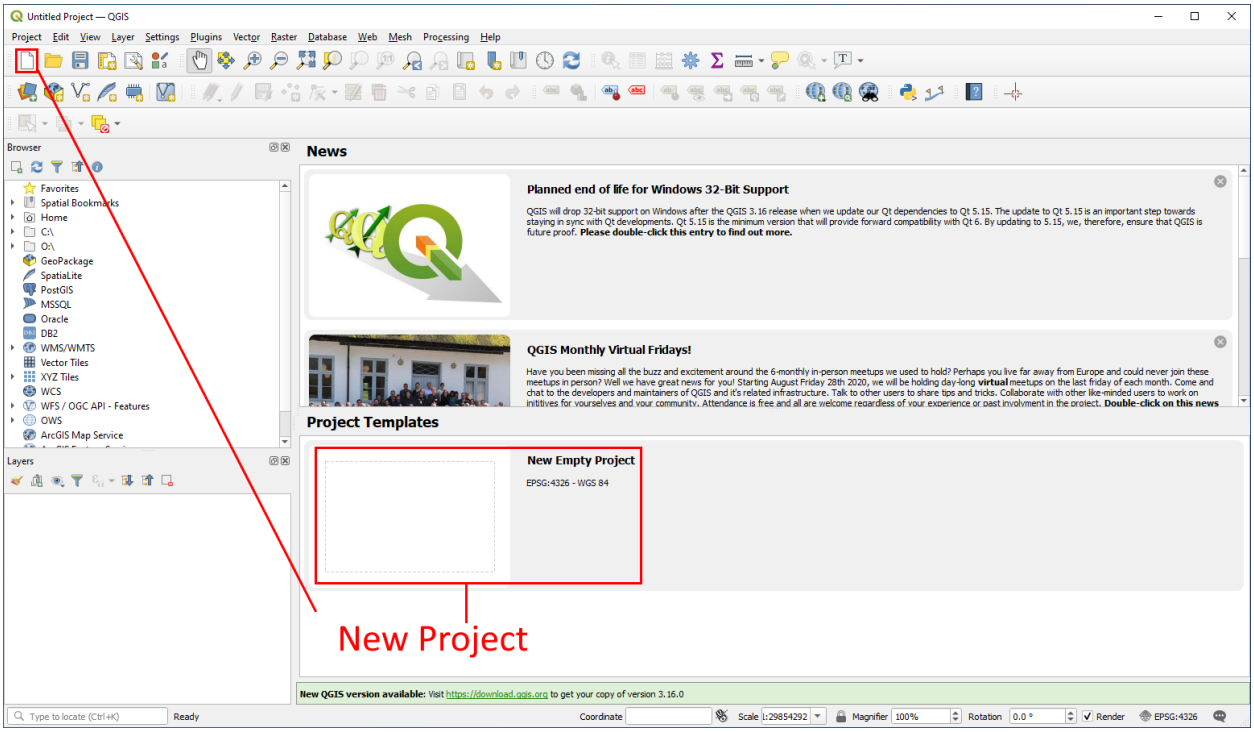
- Create a new project
- Georeference a raster
- Digitize Data

6.1.1 Downloading and Installing QGIS

QGIS can be downloaded from this webpage. Once downloaded follow the installation instructions on your screen.

6.1.2 Creating a new Project in QGIS

Create a new project in QGIS by opening the software and clicking `New Project` (CTRL-N). A blank window with no loaded layers will appear.



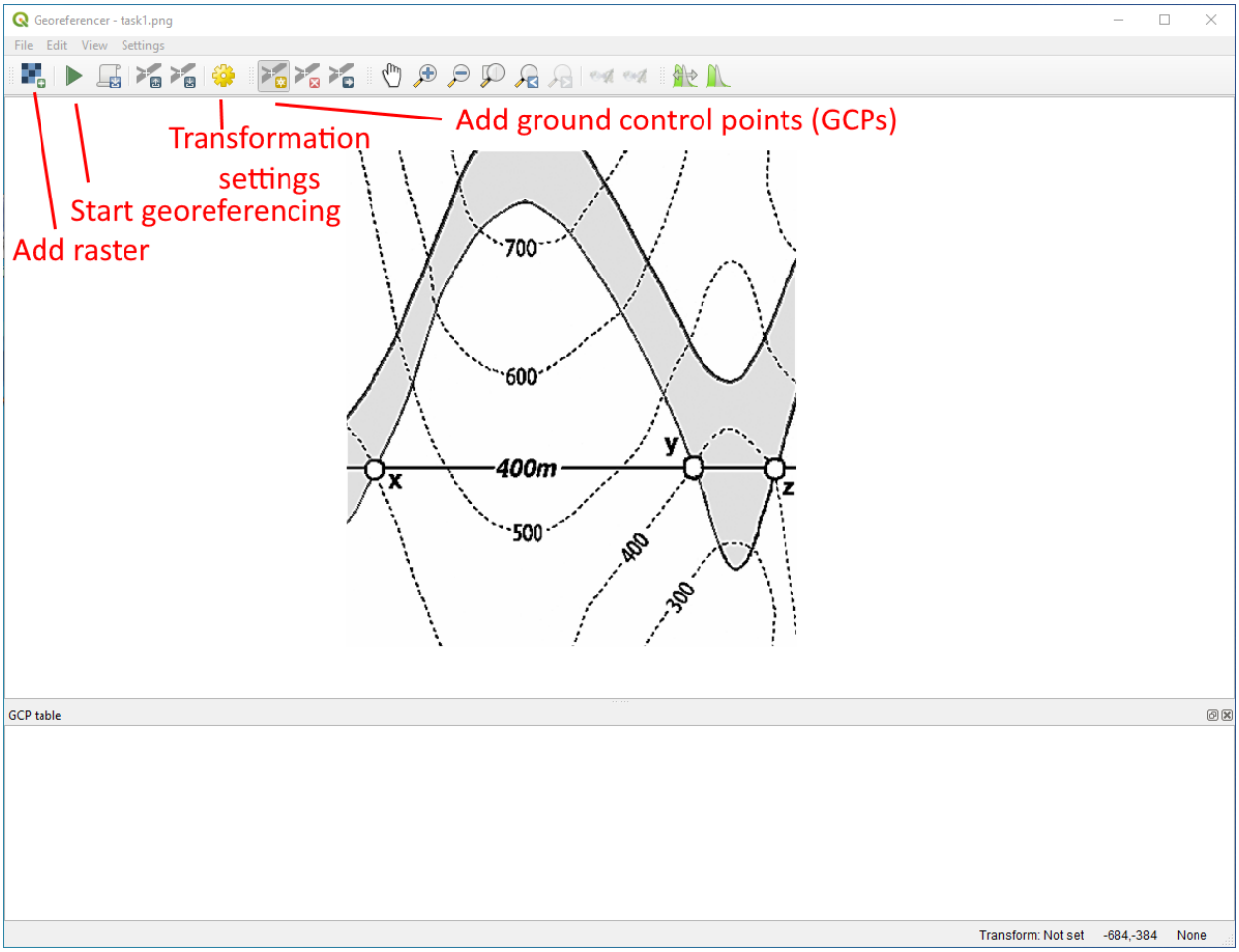
6.1.3 Installing the Georeferencing Tool

In order to work with real data and real coordinates, we need to tell QGIS at what real world position our data is located. Many dataset are already georeferenced, meaning that each point or set of data can be attributed real world coordinates. In order to be able to tell QGIS where the data is located for instance on a map, this map needs to be georeferenced. An extensive guide on how to georeference a raster can be found here.

In your menu bar, click on `Plugins` -> `Manage and Install Plugins` and search for the `Georeferencer` plugin to be able to georeference rasters. Once you have found it, install it.

6.1.4 Georeferencing a raster

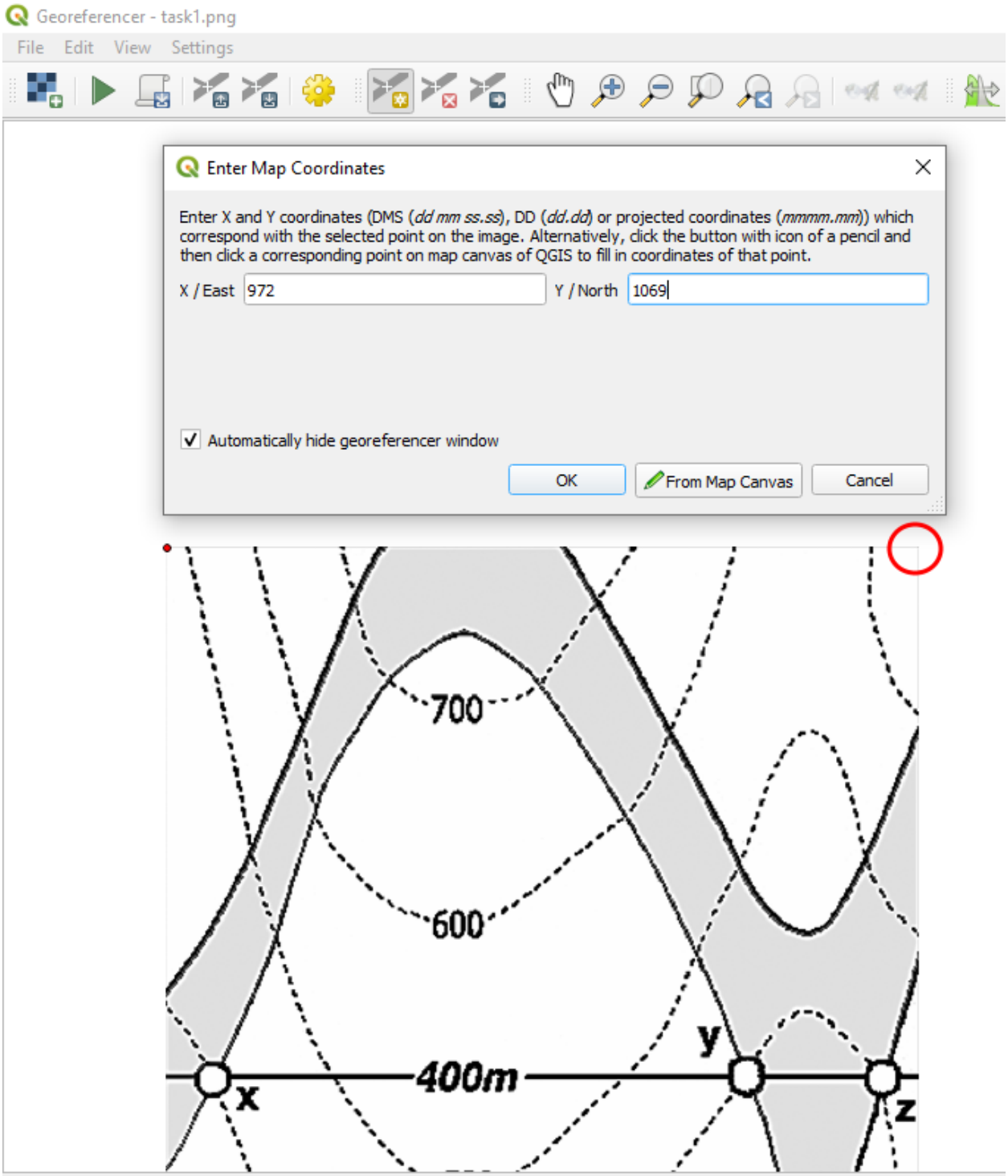
Open the georeferencing tool by clicking on `Raster` -> `Georeferencer` in your menu bar. A new blank window will open like shown below. Click the `Open Raster` button to add a new raster from your disc. Click on the gear to change the `Transformation Settings`. Click on `Add Point` to add ground control points (GCP). These GCPs are points from which the position on the map and the position in the real world are known. This could be the coordinate cross on analog maps, landmarks, cities or river bends. Click on `Start Georeferencing` once enough GCPs were set and the transformation settings have been set. The number of needed GCP points varies usually between four and six. It is not recommended to set much more points than needed for the transformation to avoid distoritions of the raster.



Source: Powell, D. (1995): Interpretation geologischer Strukturen durch Karten - Eine praktische Anleitung mit Aufgaben und Lösungen, page 15, figure 10 A, Springer Verlag Berlin, Heidelberg, New York, ISBN: 978-3-540-58607-4.

6.1.5 Adding Ground Control Points

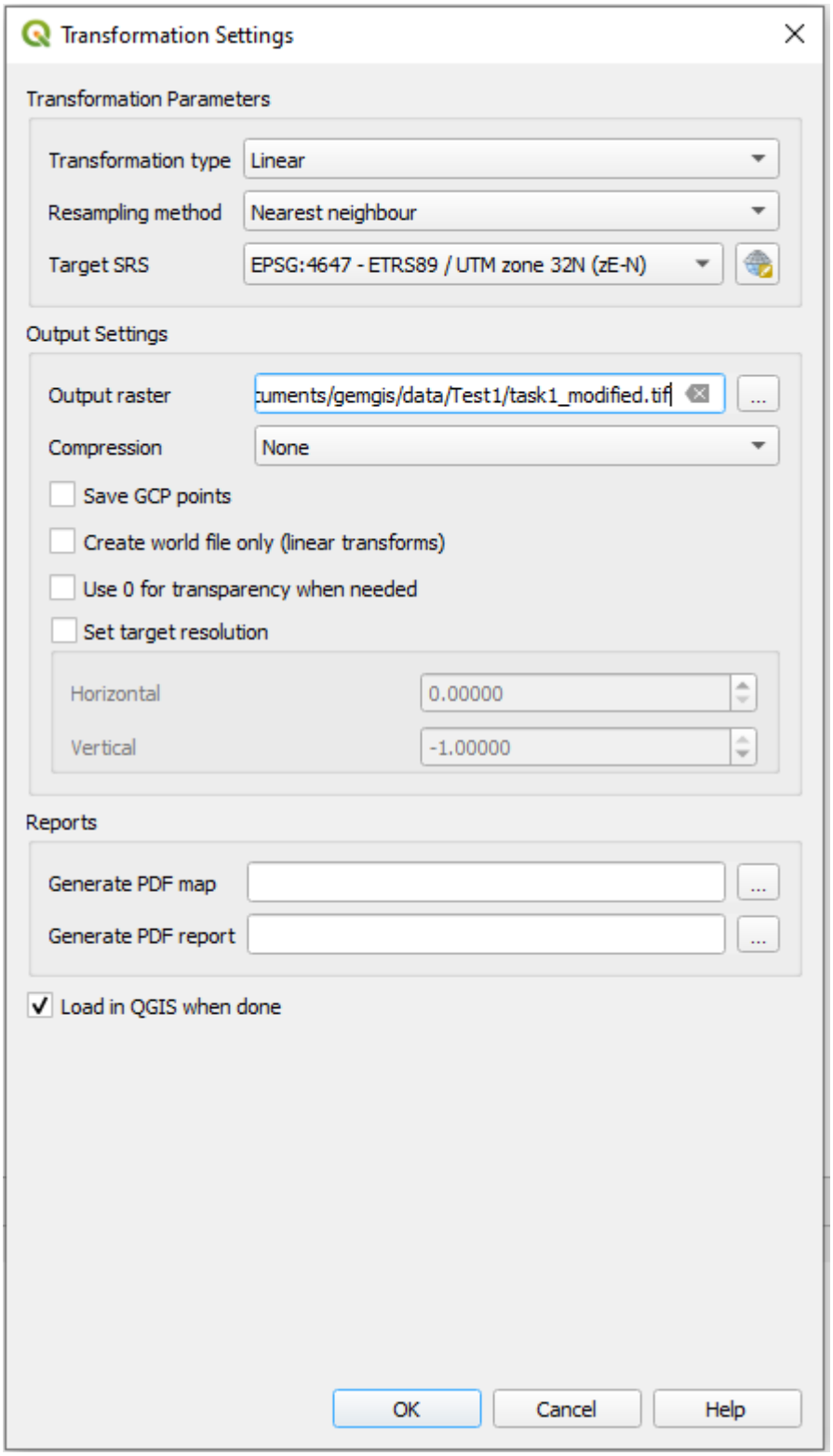
Add ground control points by clicking `Add Point` you can then click on any position on the raster. A window will open which asks you to provide the coordinates of this point. Have these coordinates ready for the georeferencing. In this case, we assume that the lower left corner is the origin (0,0) and that the image has a 1 m resultion with 972 m in W-E direction and 1069 m in N-S direction. Three points have already been added to the table. The fourth one, the upper right corner, will be added as last point before setting the Transformation Settings. These four points are sufficient to georeference the raster.


Georeferencer - task1.png
File Edit View Settings
Enter Map Coordinates
Enter X and Y coordinates (DMS (dd mm ss.ss), DD (dd.dd) or projected coordinates (mmmm.mm)) which correspond with the selected point on the image. Alternatively, click the button with icon of a pencil and then click a corresponding point on map canvas of QGIS to fill in coordinates of that point.
X / East 972
Y / North 1069
Automatically hide georeferencer window
OK
From Map Canvas
Cancel
700
600
400m
x
y
z
GCP table
Visible ID Source X Source Y Dest. X Dest. Y dX (pixels) dY (pixels) Residual (pixels)
0 -0.185599 -1069.06 0 0 0 0 0
1 971.869 -1069.22 972 0 0 0 0
2 0.277338 -0.217134 0 1069 0 0 0

Source: Powell, D. (1995): Interpretation geologischer Strukturen durch Karten - Eine praktische Anleitung mit Aufgaben und Lösungen, page 15, figure 10 A, Springer Verlag Berlin, Heidelberg, New York, ISBN: 978-3-540-58607-4.

6.1.6 Adjusting Transformation Settings

Click the gear to adjust the transformation settings. You can adjust the `Transformation Parameters` by setting the `Transformation Type`, the `Resampling Method` and the `Target CRS` (Coordinate Reference System). You can also check the box to load the the georeferenced map directly into QGIS. Click `Okay` when all settings have been adjusted to your needs.


Transformation Settings
Transformation Parameters
Transformation type
Linear
Resampling method
Nearest neighbour
Target SRS
EPSG:4647 - ETRS89 / UTM zone 32N (zE-N)
Output Settings
Output raster
cuments/gemgis/data/Test1/task1_modified.tif
Compression
None
Save GCP points
Create world file only (linear transforms)
Use 0 for transparency when needed
Set target resolution
Horizontal
0.00000
Vertical
-1.00000
Reports
Generate PDF map
Generate PDF report
Load in QGIS when done
OK
Cancel
Help

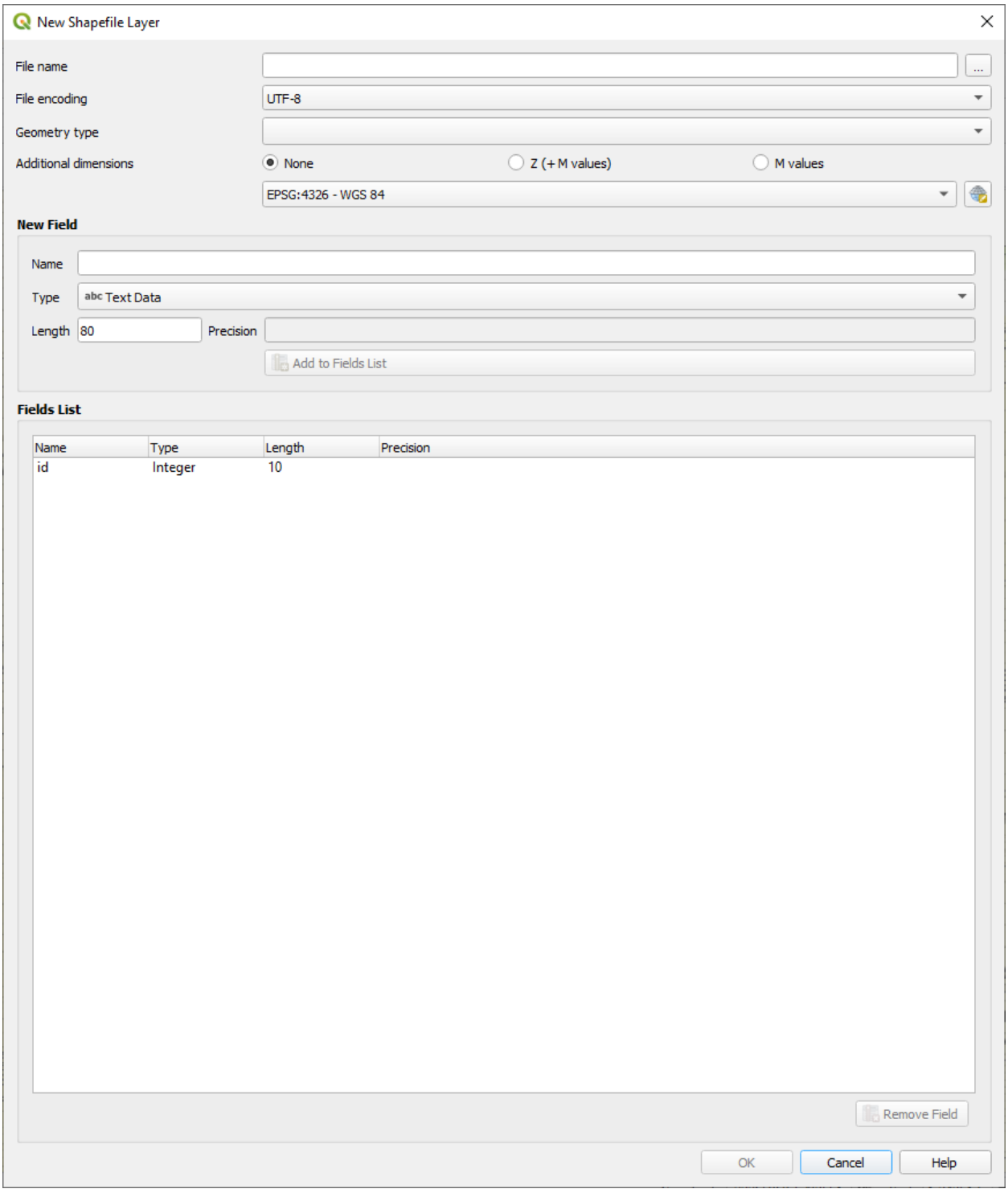
6.1.7 Start Georeferencing

Click the green arrow `Start Georeferencing` to start the georeferencing process. If the box to load the raster into QGIS once the georeferencing was successful was checked, the raster will appear in the project window and can be used for further tasks.

6.1.8 Creating a new Shapefile in QGIS

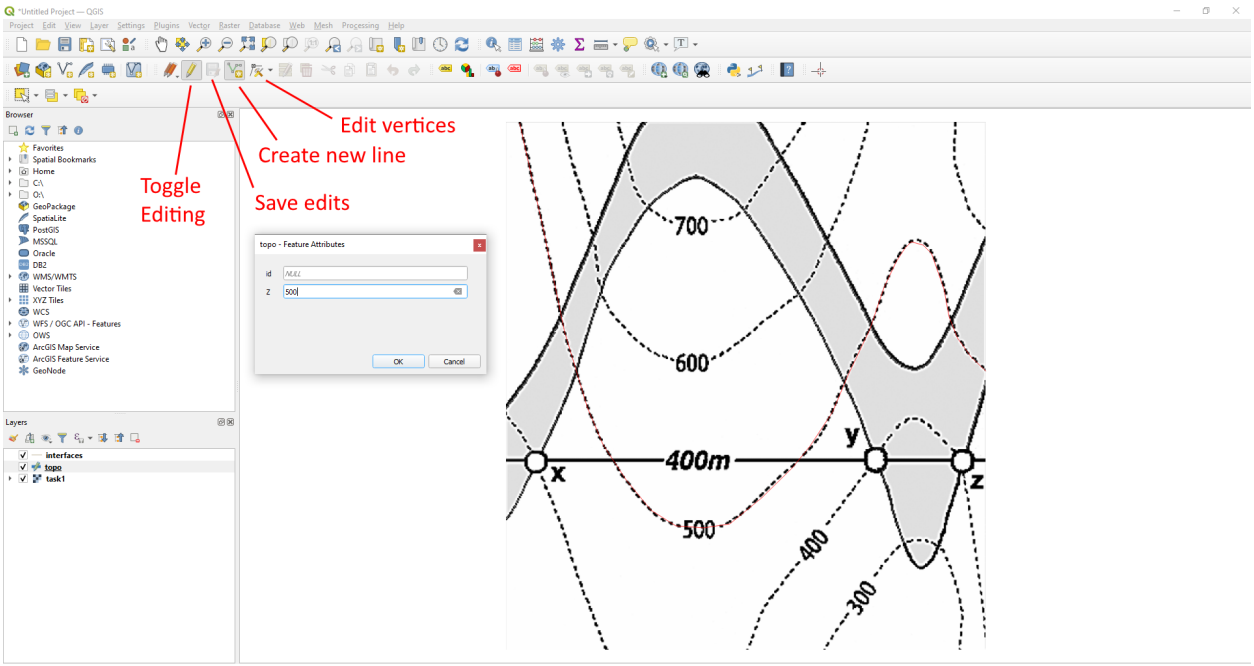
Shape files are the main vector data types that are used in QGIS. They contain information regarding the geometry of the data, the projection and additional data stored as database.

Create a new shape file by clicking `Layer` -> `Create Layer` -> `New Shapefile Layer` in the menu bar. You can select the `File Name`, the `Geometry Type`, the coordinate reference system and add new fields to the shape file within the opened dialog. You can choose between `Points`, `MultiPoints`, `Lines` and `Polygons` as geometry type. Fields containing strings or numbers can be added to the shape file. These fields could contain information about the formation a point, line or polygon belongs to, the height of a contour line, or the name of a profile trace. Once created, the shape file will appear in your layer overview.

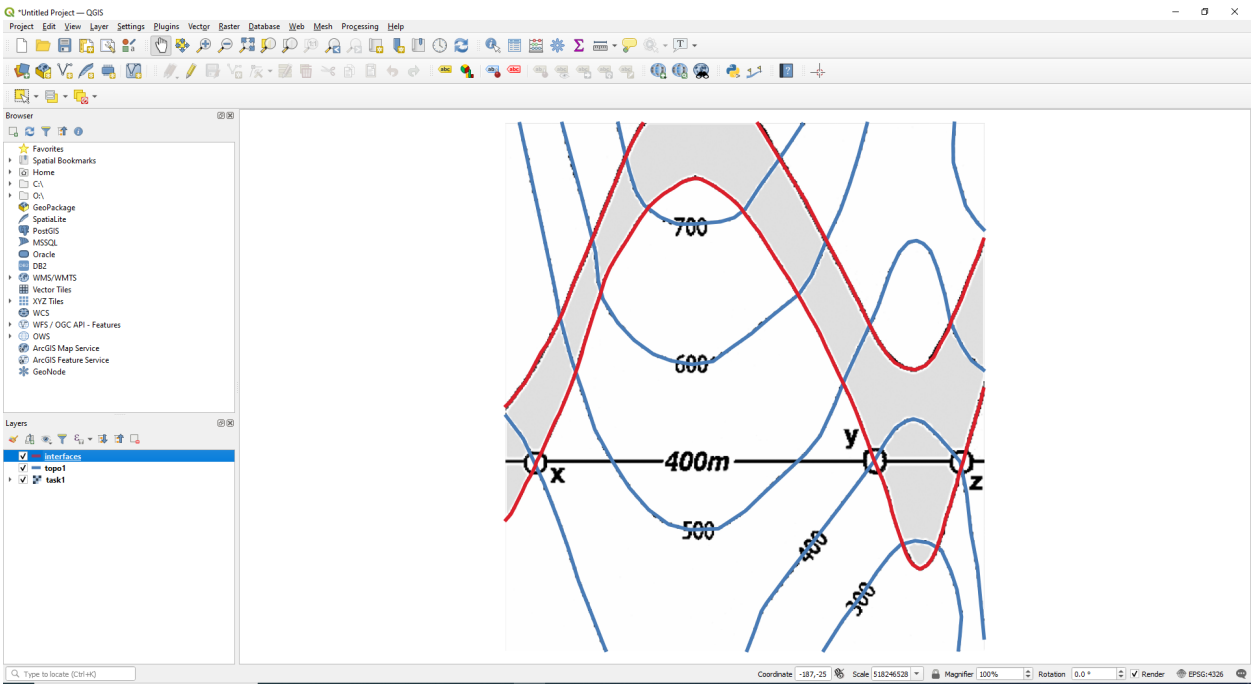

New Shapefile Layer
File name
File encoding
UTF-8
Geometry type
Additional dimensions
None
Z (+M values)
M values
EPSG:4326 - WGS 84
New Field
Name
Type
Text Data
Length
80
Precision
Add to Fields List
Fields List
Name
Type
Length
Precision
id
Integer
10
Remove Field
OK
Cancel
Help

6.1.9 Digitizing Data in QGIS

Once you have created a layer, `Toggle Editing` to start digitizing your data. Click `Add Line Feature` to add a line. You conclude the line by doing a right click. The `Features Attribute` window will open to enter additional attribute data. Do not forget to save your work! `Toggle Editing` again to stop editing your layer.



Source: Powell, D. (1995): Interpretation geologischer Strukturen durch Karten - Eine praktische Anleitung mit Aufgaben und Lösungen, page 15, figure 10 A, Springer Verlag Berlin, Heidelberg, New York, ISBN: 978-3-540-58607-4.



Source: Powell, D. (1995): Interpretation geologischer Strukturen durch Karten - Eine praktische Anleitung mit Aufgaben und Lösungen, page 15, figure 10 A, Springer Verlag Berlin, Heidelberg, New York, ISBN: 978-3-540-58607-4.

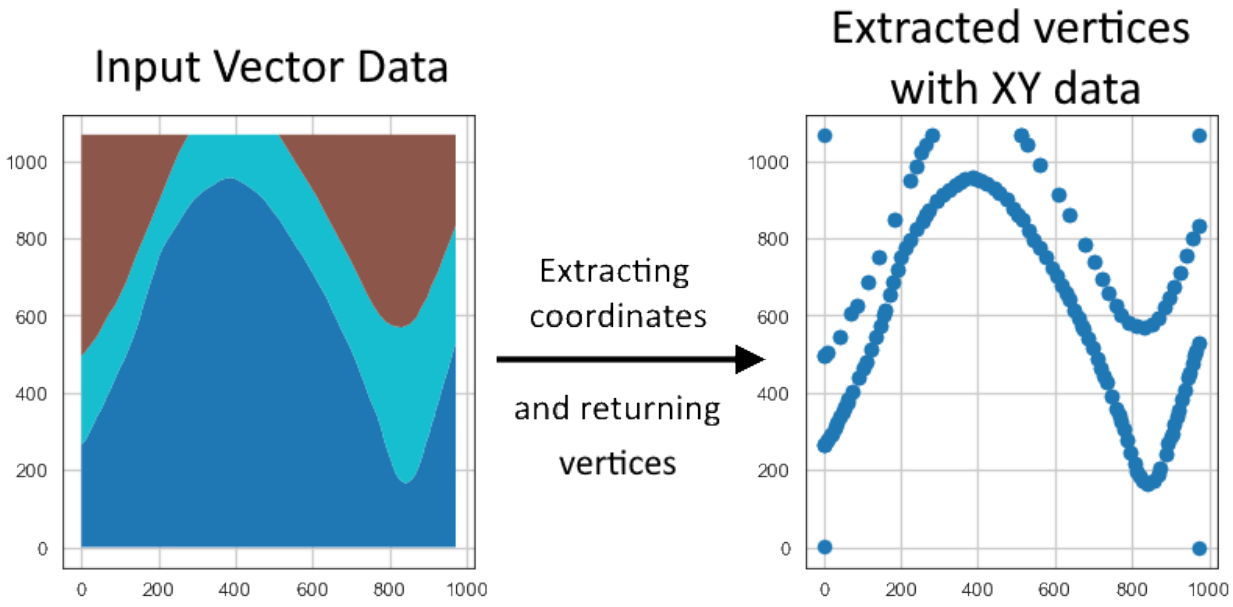
Congratulations! You are now ready to work with vector and raster data in GemGIS.

6.2 01 Extract XY Coordinates

Vector data is commonly provided as `shape` files. These files can be loaded with `GeoPandas` as `GeoDataFrames`. Each geometry object is stored as `shapely` object within the `GeoSeries` `geometry` of the `GeoDataFrames`. The basic `shapely` objects, also called Base Geometries, used here are:

- Points/Multi-Points
- Lines/Multi-Lines
- Polygons/Multi-Polygons
- Geometry Collections

The first step is to load the data using `GeoPandas`. We can inspect the different columns of the GeoDataFrame by looking at its head. In the following examples for point, line and polygon data, we have an `id` column which was created during the digitalizing of the data in QGIS, a formation column containing the name of a geological unit (this becomes important later for the actual modeling) and most importantly the geometry column consisting of the `shapely` geometry objects. The X and Y coordinates of the different geometry objects can then be extracted using `extract_xy()` of the GemGIS vector module.



6.2.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/01_extract_xy/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="01_extract_xy.zip",␣
↪dirpath=file_path)
```

```
Downloading file '01_extract_xy.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F01_extract_xy.zip' to 'C:\Users\ale93371\Documents\
↪gemgis\docs\getting_started\tutorial\data\01_extract_xy'.
```

6.2.2 Point Data

The point data stored as shape file will be loaded as GeoDataFrame. It contains an `id`, `formation` and the `geometry` column.

[3]:
```
import geopandas as gpd

gdf = gpd.read_file(file_path + 'interfaces_points.shp')

gdf.head()
```

[3]:

	id	formation	geometry
0	None	Ton	POINT (19.15013 293.31349)
1	None	Ton	POINT (61.93437 381.45933)
2	None	Ton	POINT (109.35786 480.94557)
3	None	Ton	POINT (157.81230 615.99943)
4	None	Ton	POINT (191.31803 719.09398)

Inspecting the geometry column

The elements of the geometry columns can be accessed by indexing the GeoDataFrame. It can be seen that the objects in the `geometry` column are Shapely objects.

[4]:
```
gdf.loc[0].geometry
```

[4]:

[5]:
```
gdf.loc[0].geometry.wkt
```

[5]:
```
'POINT (19.150128045807676 293.313485355882)'
```

[6]:
```
type(gdf.loc[0].geometry)
```

[6]:
```
shapely.geometry.point.Point
```

Extracting the Coordinates

The resulting GeoDataFrame has now an additional `X` and `Y` column containing the coordinates of the point objects. These can now be easily used for further processing. The geometry types of the shapely objects remained unchanged. The `id` column was dropped by default.

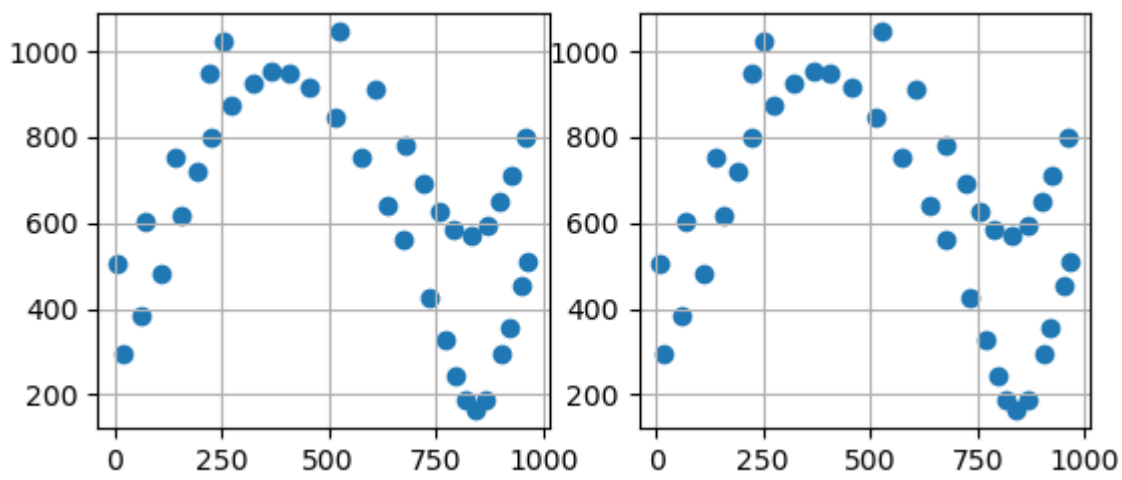
[7]:
```
gdf_xy = gg.vector.extract_xy(gdf=gdf)

gdf_xy.head()
```

[7]:
```
  formation                     geometry       X       Y
0       Ton   POINT (19.15013 293.31349)   19.15 293.31
1       Ton   POINT (61.93437 381.45933)   61.93 381.46
2       Ton  POINT (109.35786 480.94557)  109.36 480.95
3       Ton  POINT (157.81230 615.99943)  157.81 616.00
4       Ton  POINT (191.31803 719.09398)  191.32 719.09
```

Plotting the Data

The figures below show the original point data and the extracted X and Y data using `matplotlib`.

[8]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

gdf.plot(ax=ax1, aspect='equal')
ax1.grid()

gdf_xy.plot(ax=ax2, aspect='equal')
ax2.grid()
```



6.2.3 Line Data

The line data stored as shape file will be loaded as GeoDataFrame. It consists of the same columns as the shape file containing points.

[9]:
```
import geopandas as gpd
import gemgis as gg

gdf = gpd.read_file(file_path + 'interfaces_lines.shp')

gdf.head()
```

[9]:
```
     id formation                                           geometry
0  None      Sand1  LINESTRING (0.25633 264.86215, 10.59347 276.73...
```

(continues on next page)

(continued from previous page)

```
1  None       Ton  LINESTRING (0.18819 495.78721, 8.84067 504.141...
2  None       Ton  LINESTRING (970.67663 833.05262, 959.37243 800...
```

Inspecting the geometry column

The elements of the geometry columns can be accessed by indexing the GeoDataFrame. It can be seen that the objects in the `geometry` column are Shapely objects.

[10]:
```
gdf.loc[0].geometry
```

[10]:

[11]:
```
gdf.loc[0].geometry.wkt[:100]
```

[11]:
```
'LINESTRING (0.256327195431048 264.86214748436396, 10.59346813871597 276.73370778641777,␣
↪17.134940141'
```

[12]:
```
type(gdf.loc[0].geometry)
```

[12]:
```
shapely.geometry.linestring.LineString
```

Extracting the Coordinates to Point Objects

The resulting GeoDataFrame has now an additional `X` and `Y` column. These values represent the single vertices of each LineString. The geometry types of the shapely objects in the GeoDataFrame were converted from LineStrings to Points to match the X and Y column data. The `id` column was dropped by default. The index of the new GeoDataFrame was reset.

[13]:
```
gdf_xy = gg.vector.extract_xy(gdf=gdf)

gdf_xy.head()
```

[13]:
```
  formation                    geometry     X      Y
0     Sand1   POINT (0.25633 264.86215)  0.26 264.86
1     Sand1  POINT (10.59347 276.73371) 10.59 276.73
2     Sand1  POINT (17.13494 289.08982) 17.13 289.09
3     Sand1  POINT (19.15013 293.31349) 19.15 293.31
4     Sand1  POINT (27.79512 310.57169) 27.80 310.57
```

Plotting the Data

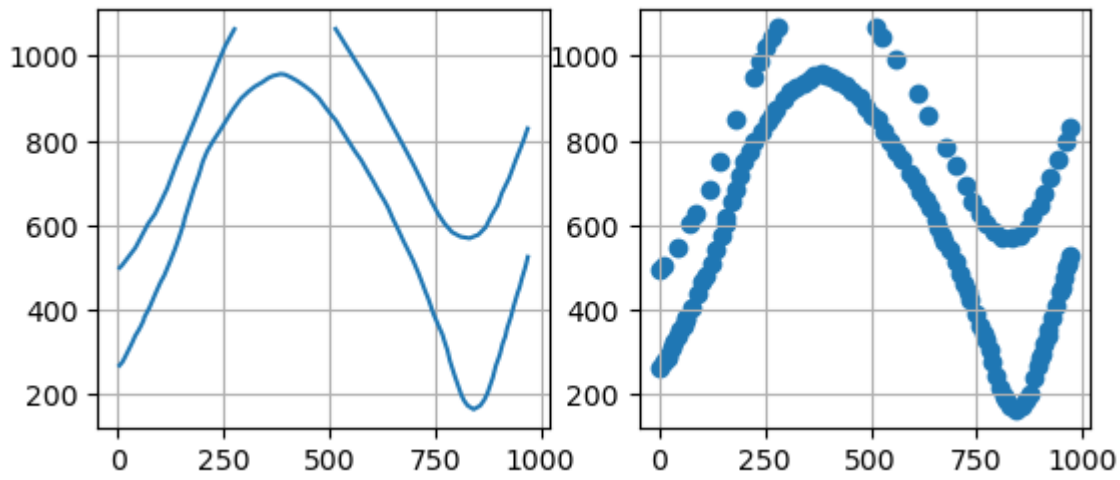
The figures below show the original line data and the extracted point data with the respective X and Y data using `matplotlib`. It can be seen that the single vertices the original LineStrings were made of were extracted.

[14]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

gdf.plot(ax=ax1, aspect='equal')
ax1.grid()
```

(continues on next page)

(continued from previous page)

```
gdf_xy.plot(ax=ax2, aspect='equal')
ax2.grid()
```



Extracting the Coordinates to list of X and Y coordinates in separate cells

The coordinates of LineStrings in a GeoDataFrame can also be extracted and are stored as lists in respective X and Y columns using `extract_xy_linestring(..)`.

[15]:
```
gdf_xy = gg.vector.extract_xy_linestring(gdf=gdf)
gdf_xy
```

[15]:
```
     id formation                                           geometry  \
0  None     Sand1  LINESTRING (0.25633 264.86215, 10.59347 276.73...
1  None       Ton  LINESTRING (0.18819 495.78721, 8.84067 504.141...
2  None       Ton  LINESTRING (970.67663 833.05262, 959.37243 800...

                                                   X  \
0  [0.256327195431048, 10.59346813871597, 17.1349...
1  [0.1881868620686138, 8.840672956663411, 41.092...
2  [970.6766251230017, 959.3724321757514, 941.291...

                                                   Y
0  [264.86214748436396, 276.73370778641777, 289.0...
1  [495.787213546976, 504.1418419288791, 546.4230...
2  [833.052616499831, 800.0232029873156, 754.8012...
```

6.2.4 Polygon Data

The polygon data stored as shape file will be loaded as GeoDataFrame. It can be seen that the objects in the geometry column are Shapely objects.

[16]:
```
import geopandas as gpd
import gemgis as gg

gdf = gpd.read_file(file_path + 'interfaces_polygons.shp')

gdf.head()
```

[16]:
```
     id formation                                           geometry
0  None     Sand1  POLYGON ((0.25633 264.86215, 10.59347 276.7337...
1  None       Ton  POLYGON ((0.25633 264.86215, 0.18819 495.78721...
2  None     Sand2  POLYGON ((0.18819 495.78721, 0.24897 1068.7595...
3  None     Sand2  POLYGON ((511.67477 1068.85246, 971.69794 1068...
```

Inspecting the geometry column

The elements of the geometry columns can be accessed by indexing the GeoDataFrame. It can be seen that the objects in the `geometry` column are Shapely objects.

[17]:
```
gdf.loc[0].geometry
```

[17]:

[18]:
```
gdf.loc[0].geometry.wkt[:100]
```

[18]:
```
'POLYGON ((0.256327195431048 264.86214748436396, 10.59346813871597 276.73370778641777,␣
↪17.13494014188'
```

[19]:
```
type(gdf.loc[0].geometry)
```

[19]:
```
shapely.geometry.polygon.Polygon
```

Extracting the Coordinates

The resulting GeoDataFrame has now an additional `X` and `Y` column. These values represent the single vertices of each Polygon. The geometry types of the shapely objects in the GeoDataFrame were converted from Polygons to Points to match the X and Y column data. The `id` column was dropped by default. The index of the new GeoDataFrame was reset.

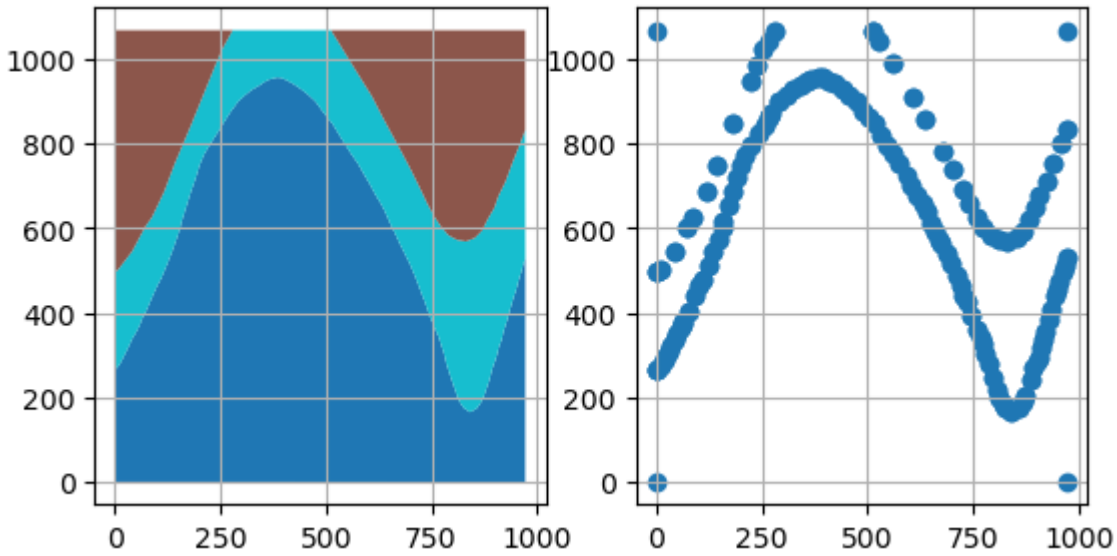
[20]:
```
gdf_xy = gg.vector.extract_xy(gdf=gdf)

gdf_xy.head()
```

[20]:
```
  formation                    geometry     X      Y
0     Sand1   POINT (0.25633 264.86215)  0.26 264.86
1     Sand1  POINT (10.59347 276.73371) 10.59 276.73
2     Sand1  POINT (17.13494 289.08982) 17.13 289.09
3     Sand1  POINT (19.15013 293.31349) 19.15 293.31
4     Sand1  POINT (27.79512 310.57169) 27.80 310.57
```

Plotting the Data

The figures below show the original polygon data and the extracted point data with the respective X and Y data using `matplotlib`. It can be seen that the single vertices the original Polygons were made of were extracted.

[21]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

gdf.plot(ax=ax1, column='formation', aspect='equal')
ax1.grid()

gdf_xy.plot(ax=ax2, aspect='equal')
ax2.grid()
```



Removing the corner points of the polygons

In the above plot, it can be seen, that the corner points are still present in the extracted X and Y point pairs. The additional argument `remove_total_bounds` can be used to remove vertices that are equal to either of the total bounds of the polygon gdf.

The total bounds of the original gdf:

[22]:
```
gdf.total_bounds
```

[22]:
```
array([ 1.88186862e-01, -6.89945891e-03,  9.72088904e+02,  1.06885246e+03])
```

The length of the extracted vertices:

[23]:
```
len(gdf_xy)
```

[23]:
```
269
```

Extracting the vertices but removing the total bounds. A total of 18 points were removed that were within 0.1 units of the total bounds.

[24]:
```
gdf_xy_without_bounds = gg.vector.extract_xy(gdf=gdf, remove_total_bounds=True,␣
↪threshold_bounds=0.1)
print(len(gdf_xy_without_bounds))
gdf_xy_without_bounds.head()
```

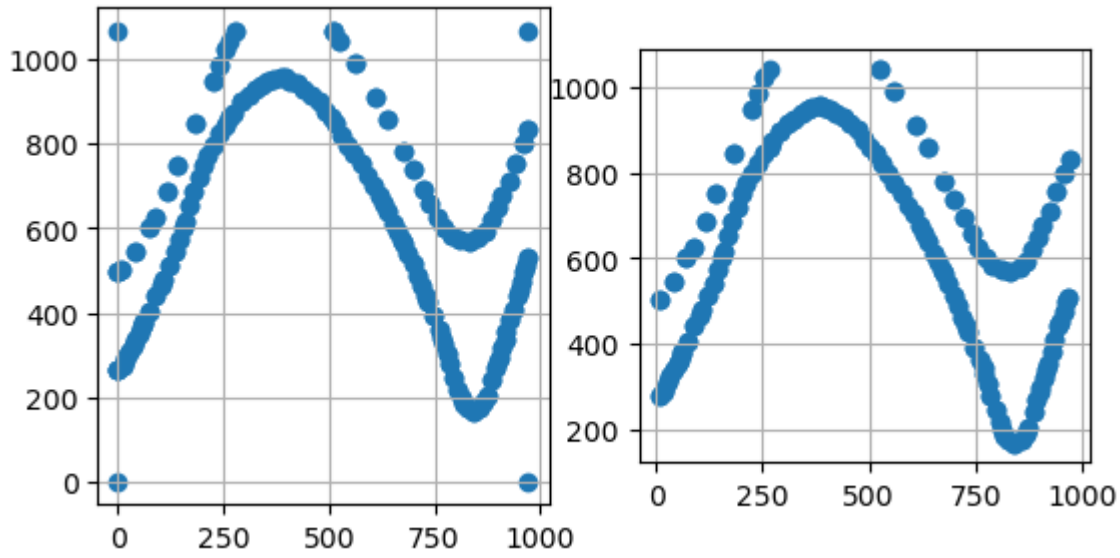
```
251
```

[24]:

	formation	geometry	X	Y
1	Sand1	POINT (10.59347 276.73371)	10.59	276.73
2	Sand1	POINT (17.13494 289.08982)	17.13	289.09
3	Sand1	POINT (19.15013 293.31349)	19.15	293.31
4	Sand1	POINT (27.79512 310.57169)	27.80	310.57
5	Sand1	POINT (34.41735 324.13919)	34.42	324.14

The removal of the points can also be inspected visually.

[25]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

gdf_xy.plot(ax=ax1, aspect='equal')
ax1.grid()

gdf_xy_without_bounds.plot(ax=ax2, aspect='equal')
ax2.grid()
```



6.2.5 Geometry Collections

Geometry collections contain different types of geometries. Here, a GeoDataFrame is created with one GeometryCollection object and two LineStrings.

[26]:
```
from shapely.geometry import LineString
import geopandas as gpd
import gemgis as gg

line1 = LineString([(0, 0), (1, 1), (1, 2), (2, 2)])
line2 = LineString([(0, 0), (1, 1), (2, 1), (2, 2)])
collection = line1.intersection(line2)
```

[27]:
```
type(collection)
```

[27]:
```
shapely.geometry.collection.GeometryCollection
```

Creating the GeoDataFrame with different geom_types

[28]:
```
gdf = gpd.GeoDataFrame(geometry=[collection, line1, line2])
gdf
```

[28]:
```
                                            geometry
0  GEOMETRYCOLLECTION (LINESTRING (0.00000 0.0000...
1  LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
2  LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
```

Extracting the Coordinates

The resulting GeoDataFrame has now an additional `X` and `Y` column. These values represent the single vertices of each Polygon. The geometry types of the shapely objects in the GeoDataFrame were converted from Polygons to Points to match the X and Y column data. The `id` column was dropped by default. The index of the new GeoDataFrame was reset.

NB: By default, points within a geometry collection are dropped as they usually do not represent true layer boundaries and rather corner points.

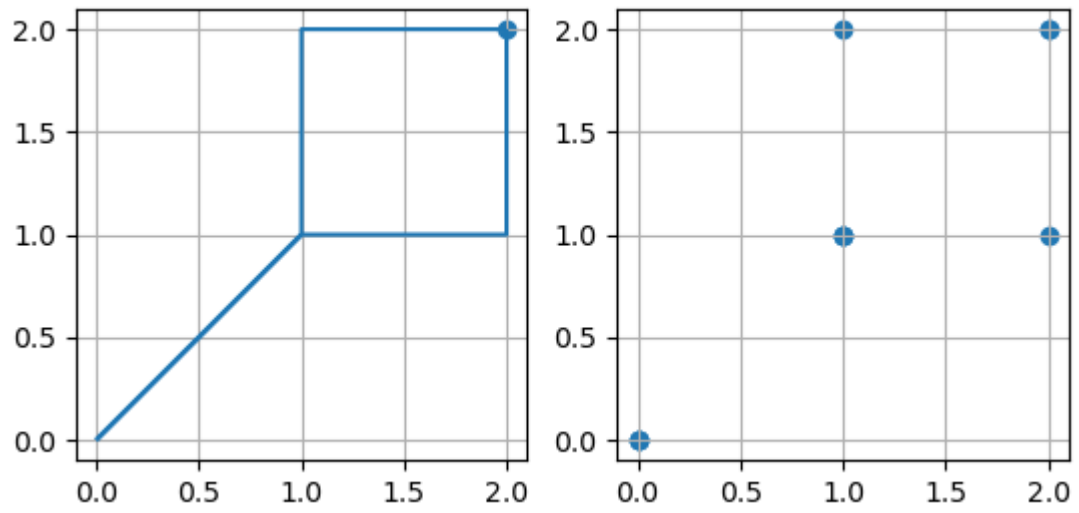
[29]:
```
gdf_xy = gg.vector.extract_xy(gdf=gdf)

gdf_xy
```

[29]:
```
                   geometry    X    Y
0  POINT (0.00000 0.00000) 0.00 0.00
1  POINT (1.00000 1.00000) 1.00 1.00
2  POINT (0.00000 0.00000) 0.00 0.00
3  POINT (1.00000 1.00000) 1.00 1.00
4  POINT (1.00000 2.00000) 1.00 2.00
5  POINT (2.00000 2.00000) 2.00 2.00
6  POINT (0.00000 0.00000) 0.00 0.00
7  POINT (1.00000 1.00000) 1.00 1.00
8  POINT (2.00000 1.00000) 2.00 1.00
9  POINT (2.00000 2.00000) 2.00 2.00
```

Plotting the Data

The figures below show the original polygon data and the extracted point data with the respective X and Y data with `matplotlib`. Note the point in the upper right of the left plot which was dropped during the extraction of the vertices.

[30]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

gdf.plot(ax=ax1, aspect='equal')
ax1.grid()

gdf_xy.plot(ax=ax2, aspect='equal')
ax2.grid()
```



6.2.6 Additional Arguments

Several additional arguments can be passed to adapt the functionality of the function. For further reference, see the API Reference for extract_xy.

- reset_index (bool)
- drop_id (bool)
- drop_level0 (bool)
- drop_level1 (bool)
- drop_index (bool)
- drop_points (bool)
- overwrite_xy (bool)
- target_crs(str, pyproj.crs.crs.CRS)
- bbox (list)
- remove_total_bounds (bool)
- threshold_bounds (float, int)

Original Function

Original function with default values of arguments.

[31]:
```
gdf_xy = gg.vector.extract_xy(gdf=gdf,
                              reset_index=True,
                              drop_id=True,
                              drop_level0=True,
                              drop_level1=True,
                              drop_index=True,
                              drop_points=True,
                              overwrite_xy=True,
                              target_crs=gdf.crs,
                              bbox = None)

gdf_xy.head()
```

[31]:
```
                   geometry    X    Y
0  POINT (0.00000 0.00000) 0.00 0.00
1  POINT (1.00000 1.00000) 1.00 1.00
2  POINT (0.00000 0.00000) 0.00 0.00
3  POINT (1.00000 1.00000) 1.00 1.00
4  POINT (1.00000 2.00000) 1.00 2.00
```

Avoid resetting the index and do not drop ID column

This time, the index is not reset and the id column is not dropped.

[32]:
```
gdf_xy = gg.vector.extract_xy(gdf=gdf,
                              reset_index=False,
                              drop_id=False,
                              drop_level0=True,
                              drop_level1=True,
                              drop_index=False,
                              drop_points=False,
                              overwrite_xy=True,
                              target_crs=gdf.crs,
                              bbox = None)

gdf_xy.head()
```

[32]:
```
                     geometry      points    X    Y
0 0  POINT (0.00000 0.00000)  [0.0, 0.0] 0.00 0.00
  0  POINT (1.00000 1.00000)  [1.0, 1.0] 1.00 1.00
1 0  POINT (0.00000 0.00000)  [0.0, 0.0] 0.00 0.00
  0  POINT (1.00000 1.00000)  [1.0, 1.0] 1.00 1.00
  0  POINT (1.00000 2.00000)  [1.0, 2.0] 1.00 2.00
```

Resetting the index and keeping index columns

The index is reset but the previous index columns `level_0` and `level_1` are kept.

[33]:
```
gdf_xy = gg.vector.extract_xy(gdf=gdf,
                              reset_index=True,
                              drop_id=False,
                              drop_level0=False,
                              drop_level1=False,
                              drop_index=False,
                              drop_points=False,
                              overwrite_xy=True,
                              target_crs=gdf.crs,
                              bbox = None)

gdf_xy.head()
```

[33]:

	level_0	level_1	geometry	points	X	Y
0	0	0	POINT (0.00000 0.00000)	[0.0, 0.0]	0.00	0.00
1	0	0	POINT (1.00000 1.00000)	[1.0, 1.0]	1.00	1.00
2	1	0	POINT (0.00000 0.00000)	[0.0, 0.0]	0.00	0.00
3	1	0	POINT (1.00000 1.00000)	[1.0, 1.0]	1.00	1.00
4	1	0	POINT (1.00000 2.00000)	[1.0, 2.0]	1.00	2.00

6.2.7 Background Functions

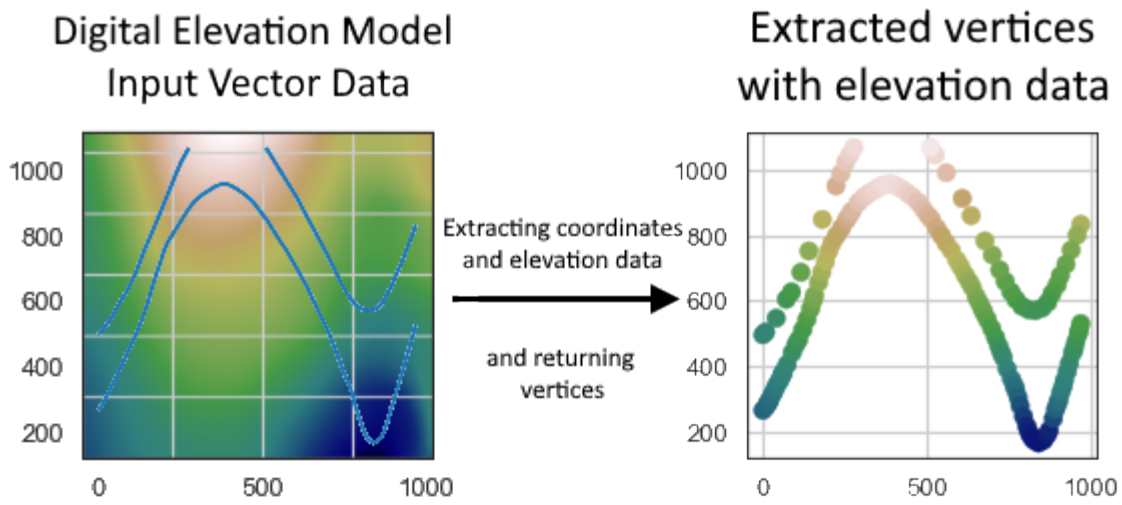
The function `extract_xy` is a combination of the following functions:

- `extract_xy_points`
- `extract_xy_linestrings`
- `explode_geometry_collection`
- `explode_multilinestrings`
- `explode_polygons`
- `set_dtype`

For more information of these functions see the API Reference.

6.3 02 Extract XYZ Coordinates

The elevation or depth of input data is needed to locate it in a 3D space. The data can either be provided when creating the data, i.e. when digitizing contour lines or by extracting it from a digital elevation model (DEM) or from an existing surface of an interface in the subsurface. For consistency, the elevation column will be denoted with `Z`. The input vector data can be loaded again as `GeoDataFrame` using `GeoPandas`. The raster from which elevation data will be extracted can either be provided as `NumPy ndarray` or opened with `rasterio` if a raster file is available on your hard disk.



6.3.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/02_extract_xyz/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="02_extract_xyz.zip",␣
↪dirpath=file_path)
```

```
Downloading file '02_extract_xyz.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F02_extract_xyz.zip' to 'C:\Users\ale93371\Documents\
↪gemgis\docs\getting_started\tutorial\data\02_extract_xyz'.
```

6.3.2 Point Data

The point data stored as shape file will be loaded as GeoDataFrame. The raster will be loaded using rasterio.

[3]:
```
import rasterio
from rasterio.plot import show
import geopandas as gpd

gdf = gpd.read_file(file_path + 'interfaces_points.shp')

dem = rasterio.open(file_path + 'raster.tif')
```

The GeoDataFrame consists of `id`, `formation` and `geometry` columns.

[4]:
```
gdf.head()
```

[4]:
```
     id formation                     geometry
0  None       Ton   POINT (19.15013 293.31349)
1  None       Ton   POINT (61.93437 381.45933)
2  None       Ton  POINT (109.35786 480.94557)
3  None       Ton  POINT (157.81230 615.99943)
4  None       Ton  POINT (191.31803 719.09398)
```

The differend bands of the raster data can be returned using the `read(..)` function. This will return the values of the raster at each cell location.

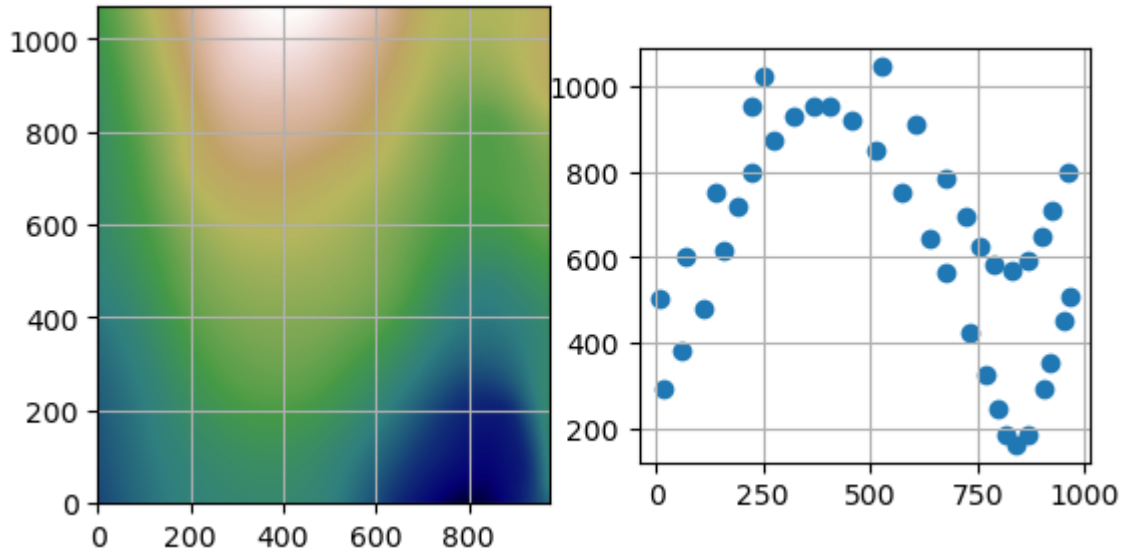
[5]:
```
dem.read()
```

[5]:
```
array([[[482.82904, 485.51953, 488.159  , ..., 618.8612 , 620.4424 ,
         622.05786],
        [481.6521 , 484.32193, 486.93958, ..., 618.8579 , 620.44556,
         622.06714],
        [480.52563, 483.18893, 485.80444, ..., 618.8688 , 620.4622 ,
         622.08923],
        ...,
        [325.49225, 327.21985, 328.94498, ..., 353.6889 , 360.03125,
         366.3984 ],
        [325.0538 , 326.78473, 328.51276, ..., 351.80603, 357.84106,
         363.96167],
        [324.61444, 326.34845, 328.0794 , ..., 350.09247, 355.87598,
         361.78635]]], dtype=float32)
```

Plotting the data

The figures below show the original raster and point data. They can be plotted using `matplotlib` functions.

[6]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

ax1.imshow(dem.read(1), cmap='gist_earth', vmin=250, vmax=750, extent=[0,972,0,1069])
ax1.grid()

gdf.plot(ax=ax2, aspect='equal')
ax2.grid()
```



Extracting the Coordinates

The `X`, `Y` and `Z` coordinates of the GeoDataFrame can be extracted using the function `extract_xyz(..)`.

The resulting GeoDataFrame has now an additional `X`, `Y` and `Z` column containing the coordinates of the point objects. These can now be easily used for further processing. The geometry types of the shapely objects remained unchanged. The id column was dropped by default.

[7]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf,
                                dem=dem)

gdf_xyz.head()
```

[7]:
```
  formation                      geometry      X      Y      Z
0       Ton   POINT (19.15013 293.31349)  19.15 293.31 364.99
1       Ton   POINT (61.93437 381.45933)  61.93 381.46 400.34
2       Ton  POINT (109.35786 480.94557) 109.36 480.95 459.55
3       Ton  POINT (157.81230 615.99943) 157.81 616.00 525.69
4       Ton  POINT (191.31803 719.09398) 191.32 719.09 597.63
```

Plotting the Result

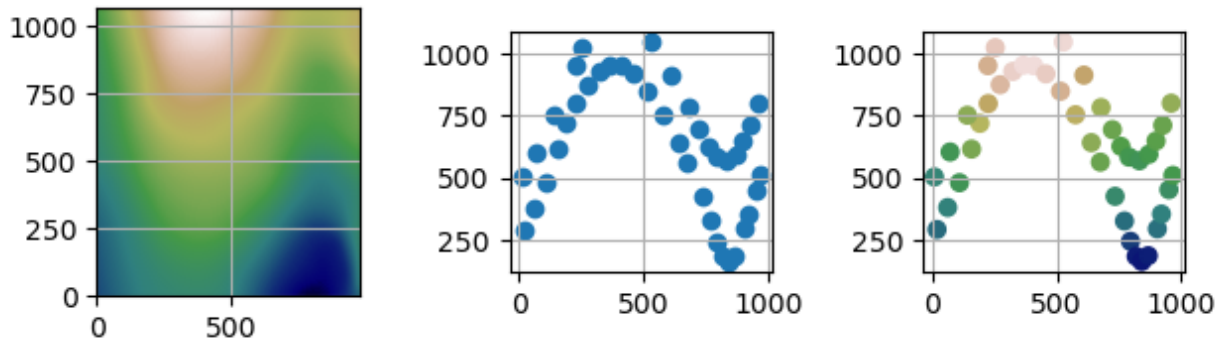
The figures below show the elevation data (blue = 250 m, white = 750 m), the original point data and the point data including color-coded X, Y and Z values.

[8]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2,ax3) = plt.subplots(1,3)

ax1.imshow(dem.read(1), cmap='gist_earth', vmin=250, vmax=750, extent=[0,972,0,1069])
ax1.grid()
```

(continues on next page)

(continued from previous page)

```
gdf.plot(ax=ax2, aspect='equal')
ax2.grid()

gdf_xyz.plot(ax=ax3, aspect='equal', column='Z', cmap='gist_earth',vmin=250, vmax=750)
ax3.grid()

plt.tight_layout()
```



6.3.3 Line Data

The point data stored as shape file will be loaded as GeoDataFrame. The raster will be loaded using rasterio.

[9]:
```
import geopandas as gpd
import rasterio
from rasterio.plot import show
import gemgis as gg

gdf = gpd.read_file(file_path + 'interfaces_lines.shp')

dem = rasterio.open(file_path + 'raster.tif')
```

[10]:
```
gdf.head()
```

[10]:
```
     id formation                                           geometry
0  None     Sand1  LINESTRING (0.25633 264.86215, 10.59347 276.73...
1  None       Ton  LINESTRING (0.18819 495.78721, 8.84067 504.141...
2  None       Ton  LINESTRING (970.67663 833.05262, 959.37243 800...
```

[11]:
```
dem.read()
```

[11]:
```
array([[[482.82904, 485.51953, 488.159  , ..., 618.8612 , 620.4424 ,
         622.05786],
        [481.6521 , 484.32193, 486.93958, ..., 618.8579 , 620.44556,
         622.06714],
        [480.52563, 483.18893, 485.80444, ..., 618.8688 , 620.4622 ,
         622.08923],
        ...,
        [325.49225, 327.21985, 328.94498, ..., 353.6889 , 360.03125,
         366.3984 ],
```

(continues on next page)

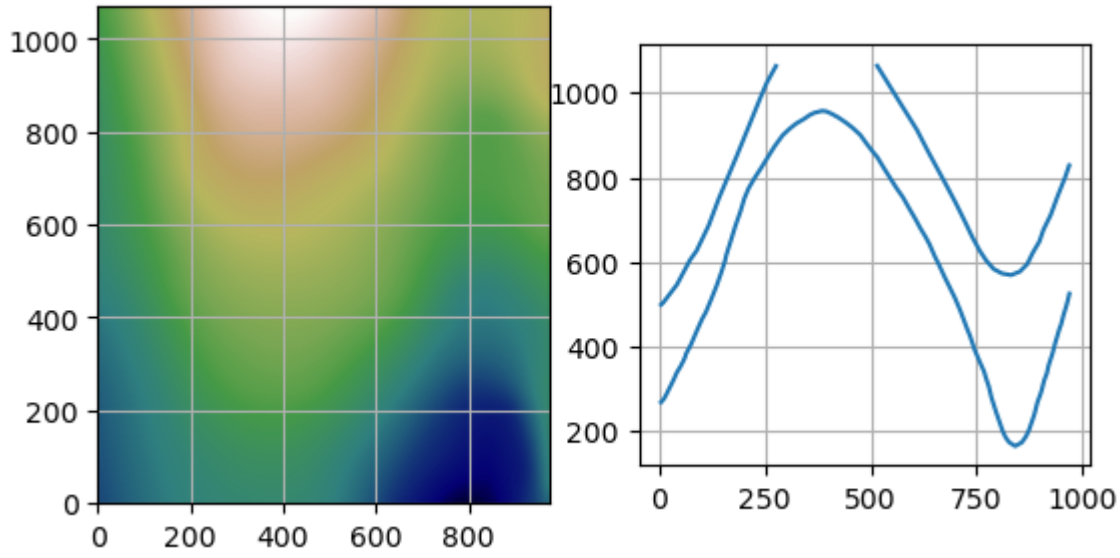
(continued from previous page)

```
        [325.0538 , 326.78473, 328.51276, ..., 351.80603, 357.84106,
         363.96167],
        [324.61444, 326.34845, 328.0794 , ..., 350.09247, 355.87598,
         361.78635]]], dtype=float32)
```

Plotting the Data

The figures below show the original raster and point data.

[12]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

ax1.imshow(dem.read(1), cmap='gist_earth', vmin=250, vmax=750, extent=[0,972,0,1069])
ax1.grid()

gdf.plot(ax=ax2, aspect='equal')
ax2.grid()
```



Extracting the Coordinates

The `X`, `Y` and `Z` coordinates of the GeoDataFrame can be extracted using the function `extract_xyz(..)`.

The resulting GeoDataFrame has now an additional `X`, `Y` and `Z` column. These represent the values of the extracted vertices. The geometry types of the shapely objects in the GeoDataFrame were converted from LineStrings to Points to match the X, Y and Y column data. The `id` column was dropped by default. The index of the new GeoDataFrame was reset.

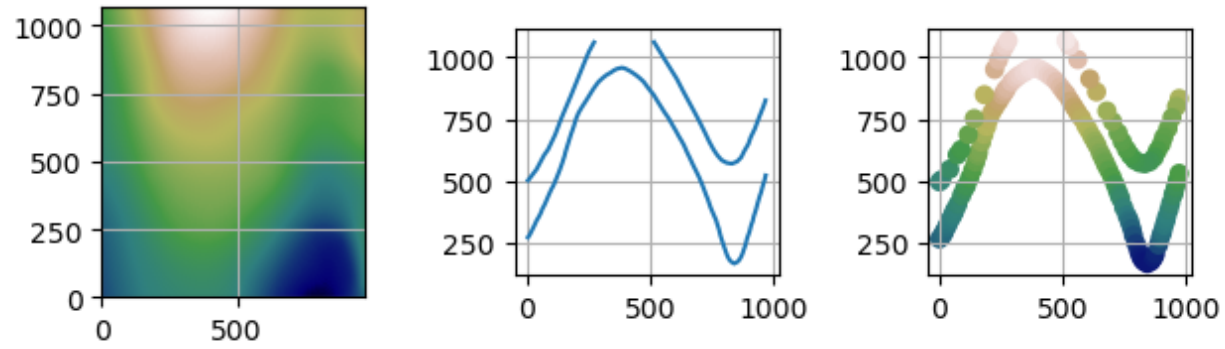
[13]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf,
                                dem=dem)

gdf_xyz.head()
```

[13]:
```
  formation                    geometry      X      Y      Z
0     Sand1   POINT (0.25633 264.86215)   0.26 264.86 353.97
1     Sand1  POINT (10.59347 276.73371)  10.59 276.73 359.04
2     Sand1  POINT (17.13494 289.08982)  17.13 289.09 364.28
3     Sand1  POINT (19.15013 293.31349)  19.15 293.31 364.99
4     Sand1  POINT (27.79512 310.57169)  27.80 310.57 372.81
```

Plotting the Result

The figures below show the elevation data (blue = 250 m, white = 750 m), the original LineString data and the extracted point data including color-coded X, Y and Z values.

[14]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2,ax3) = plt.subplots(1,3)

ax1.imshow(dem.read(1), cmap='gist_earth', vmin=250, vmax=750, extent=[0,972,0,1069])
ax1.grid()

gdf.plot(ax=ax2, aspect='equal')
ax2.grid()

gdf_xyz.plot(ax=ax3, aspect='equal', column='Z', cmap='gist_earth',vmin=250, vmax=750)
ax3.grid()

plt.tight_layout()
```



6.3.4 Polygon Data

The point data stored as shape file will be loaded as GeoDataFrame. The raster will be loaded using rasterio.

[15]:
```
import geopandas as gpd
import rasterio
from rasterio.plot import show
import gemgis as gg

gdf = gpd.read_file(file_path + 'interfaces_polygons.shp')

dem = rasterio.open(file_path + 'raster.tif')
```

[16]:
```
gdf.head()
```

[16]:
```
     id formation                                           geometry
0  None     Sand1  POLYGON ((0.25633 264.86215, 10.59347 276.7337...
1  None       Ton  POLYGON ((0.25633 264.86215, 0.18819 495.78721...
2  None     Sand2  POLYGON ((0.18819 495.78721, 0.24897 1068.7595...
3  None     Sand2  POLYGON ((511.67477 1068.85246, 971.69794 1068...
```

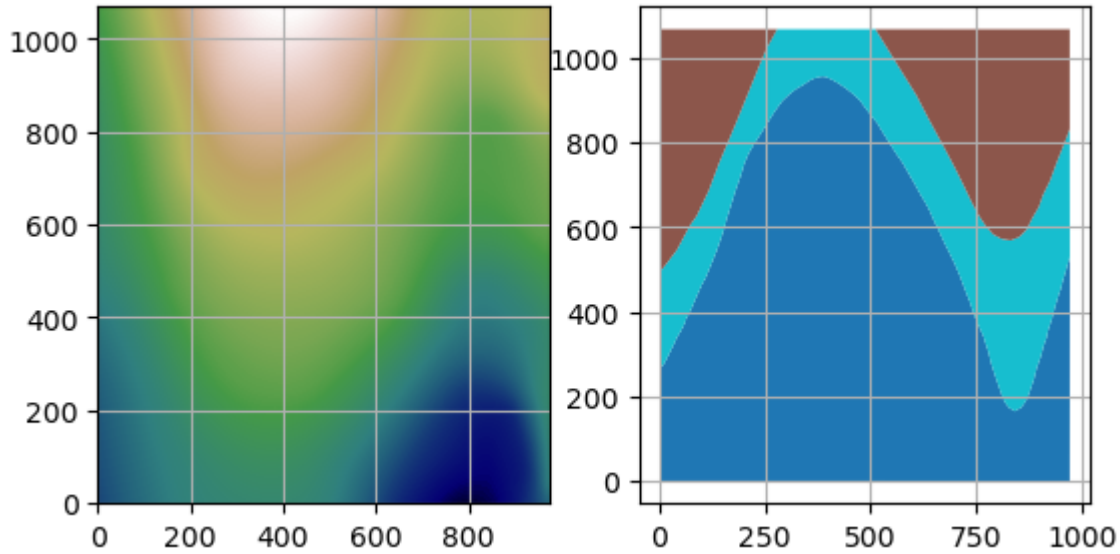
[17]:
```
dem.read()
```

[17]:
```
array([[[482.82904, 485.51953, 488.159  , ..., 618.8612 , 620.4424 ,
         622.05786],
        [481.6521 , 484.32193, 486.93958, ..., 618.8579 , 620.44556,
         622.06714],
        [480.52563, 483.18893, 485.80444, ..., 618.8688 , 620.4622 ,
         622.08923],
        ...,
        [325.49225, 327.21985, 328.94498, ..., 353.6889 , 360.03125,
         366.3984 ],
        [325.0538 , 326.78473, 328.51276, ..., 351.80603, 357.84106,
         363.96167],
        [324.61444, 326.34845, 328.0794 , ..., 350.09247, 355.87598,
         361.78635]]], dtype=float32)
```

Plotting the Data

The figures below show the original raster and point data.

[18]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2) = plt.subplots(1,2)

ax1.imshow(dem.read(1), cmap='gist_earth', vmin=250, vmax=750, extent=[0,972,0,1069])
ax1.grid()

gdf.plot(ax=ax2, aspect='equal', column='formation')
ax2.grid()
```



Extracting the Coordinates

The `X`, `Y` and `Z` coordinates of the GeoDataFrame can be extracted using the function `extract_xyz(..)`.

The resulting GeoDataFrame has now an additional `X`, `Y` and `Z` column. These represent the values of the extracted vertices. The geometry types of the shapely objects in the GeoDataFrame were converted from LineStrings to Points to match the X, Y and Y column data. The `id` column was dropped by default. The index of the new GeoDataFrame was reset.

[19]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf,
                                dem=dem,
                                remove_total_bounds=True,
                                threshold_bounds=1)

gdf_xyz.head()
```

[19]:

	formation	geometry	X	Y	Z
0	Sand1	POINT (10.59347 276.73371)	10.59	276.73	359.04
1	Sand1	POINT (17.13494 289.08982)	17.13	289.09	364.28
2	Sand1	POINT (19.15013 293.31349)	19.15	293.31	364.99
3	Sand1	POINT (27.79512 310.57169)	27.80	310.57	372.81
4	Sand1	POINT (34.41735 324.13919)	34.42	324.14	377.43

Plotting the Result

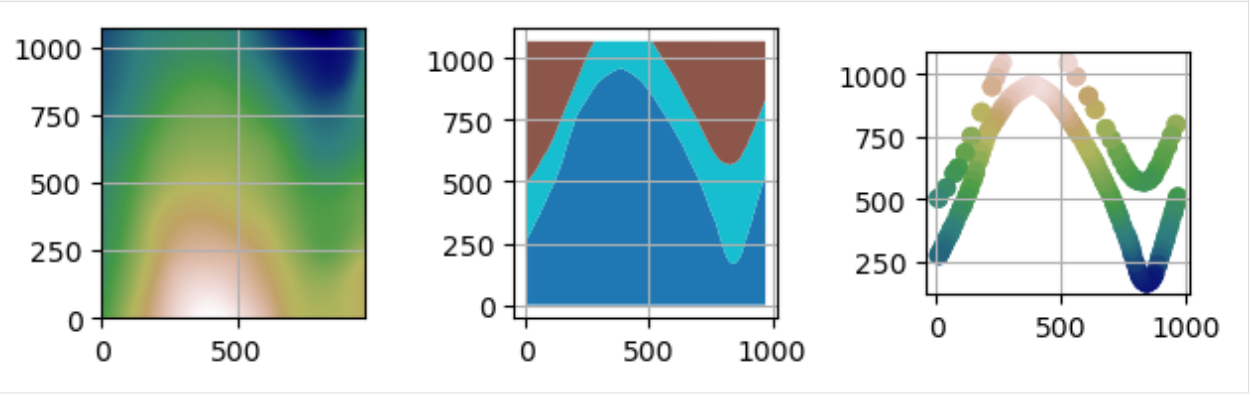
The figures below show the elevation data (blue = 250 m, white = 750 m), the original Polygon data and the extracted point data including color-coded X, Y and Z values.

[20]:
```
import matplotlib.pyplot as plt

fig, (ax1,ax2,ax3) = plt.subplots(1,3)
```

(continues on next page)

(continued from previous page)

```
ax1.imshow(dem.read(1), origin='lower', cmap='gist_earth', vmin=250, vmax=750, extent=[0,
↪972,0,1069])
ax1.grid()

gdf.plot(ax=ax2, aspect='equal', column='formation')
ax2.grid()

gdf_xyz.plot(ax=ax3, aspect='equal', column='Z', cmap='gist_earth',vmin=250, vmax=750)
ax3.grid()

plt.tight_layout()
```



6.3.5 Additional Arguments

Several additional arguments can be passed to adapt the functionality of the function. For further reference, see the API Reference for extract_xyz.

- reset_index (bool)
- drop_id (bool)
- drop_level0 (bool)
- drop_level1 (bool)
- drop_index (bool)
- drop_points (bool)
- target_crs(str, pyproj.crs.crs.CRS)
- bbox (list)
- remove_total_bounds (bool)
- threshold_bounds (float, int)

Original Function

Original function with default arguments.

[21]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf,
                                dem=dem,
                                reset_index=True,
                                drop_id=True,
                                drop_level0=True,
                                drop_level1=True,
                                drop_index=True,
                                drop_points=True,
                                target_crs=gdf.crs,
                                bbox = None)

gdf_xyz.head()
```

[21]:
```
  formation                    geometry     X      Y      Z
0     Sand1   POINT (0.25633 264.86215)  0.26 264.86 353.97
1     Sand1  POINT (10.59347 276.73371) 10.59 276.73 359.04
2     Sand1  POINT (17.13494 289.08982) 17.13 289.09 364.28
3     Sand1  POINT (19.15013 293.31349) 19.15 293.31 364.99
4     Sand1  POINT (27.79512 310.57169) 27.80 310.57 372.81
```

Avoid resetting the index and do not drop ID column

This time, the index is not reset and the id column is not dropped.

[22]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf,
                                dem=dem,
                                reset_index=False,
                                drop_id=False,
                                drop_level0=True,
                                drop_level1=True,
                                drop_index=False,
                                drop_points=False,
                                target_crs=gdf.crs,
                                bbox = None)

gdf_xyz.head()
```

[22]:
```
       id formation                    geometry  \
0 0  None     Sand1   POINT (0.25633 264.86215)
  0  None     Sand1  POINT (10.59347 276.73371)
  0  None     Sand1  POINT (17.13494 289.08982)
  0  None     Sand1  POINT (19.15013 293.31349)
  0  None     Sand1  POINT (27.79512 310.57169)

                                      points     X      Y      Z
0 0  [0.256327195431048, 264.86214748436396]  0.26 264.86 353.97
  0  [10.59346813871597, 276.73370778641777] 10.59 276.73 359.04
  0   [17.134940141888464, 289.089821570188] 17.13 289.09 364.28
  0   [19.150128045807676, 293.3134853558 82] 19.15 293.31 364.99
  0    [27.79511673965105, 310.571692592952] 27.80 310.57 372.81
```

Resetting the index and keeping index columns

The index is reset but the previous index columns `level_0` and `level_1` are kept.

[23]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf,
                                dem=dem,
                                reset_index=True,
                                drop_id=False,
                                drop_level0=False,
                                drop_level1=False,
                                drop_index=False,
                                drop_points=False,
                                target_crs=gdf.crs,
                                bbox = None)

gdf_xyz.head()
```

[23]:
```
   level_0  level_1    id formation                    geometry  \
0        0        0  None     Sand1   POINT (0.25633 264.86215)
1        0        0  None     Sand1  POINT (10.59347 276.73371)
2        0        0  None     Sand1  POINT (17.13494 289.08982)
3        0        0  None     Sand1  POINT (19.15013 293.31349)
4        0        0  None     Sand1  POINT (27.79512 310.57169)

                                   points     X      Y      Z
0  [0.256327195431048, 264.86214748436396]  0.26 264.86 353.97
1  [10.59346813871597, 276.73370778641777] 10.59 276.73 359.04
2   [17.134940141888464, 289.089821570188] 17.13 289.09 364.28
3   [19.150128045807676, 293.313485355882] 19.15 293.31 364.99
4    [27.79511673965105, 310.571692592952] 27.80 310.57 372.81
```

6.3.6 Background Functions

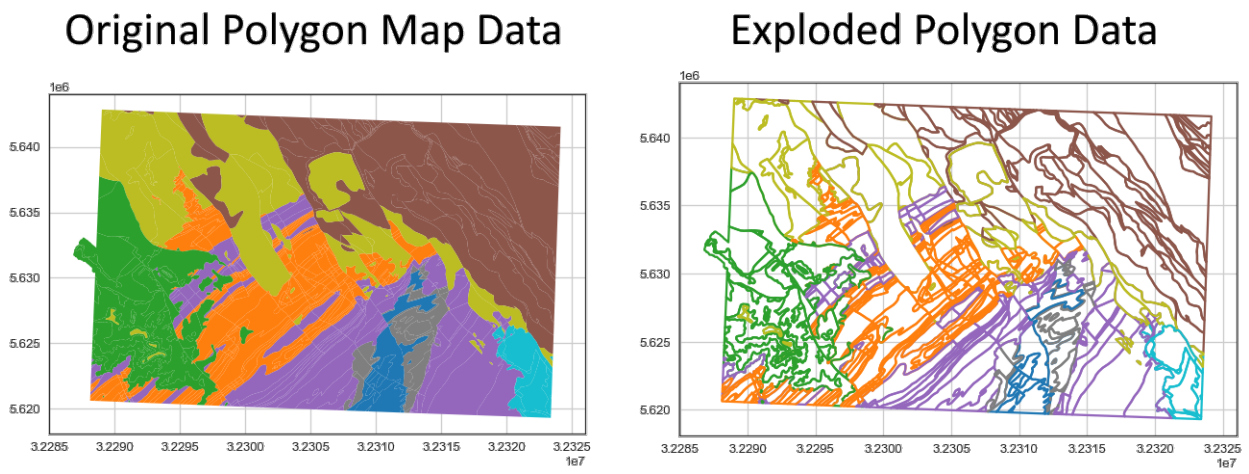
The function `extract_xy` is a combination of the following functions and their subfunctions:

- `extract_xyz_rasterio`
- `extract_xyz_array`
- `extract_xy` and subsequent functions

For more information of these functions see the API Reference.

6.4 03 Exploding Geometries

It is necessary for some operations to explode or to split the existing geometries such as LineStrings, MultiLineStrings or Polygons in single parts.



Source: Geological Map 1:50,000, Geological Survey NRW

6.4.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/03_exploding_geometries/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="03_exploding_geometries.zip",␣
↪dirpath=file_path)
```

```
Downloading file '03_exploding_geometries.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F03_exploding_geometries.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\tutorial\data\03_exploding_geometries'.
```

6.4.2 Exploding LineStrings into single elements

LineStrings can be exploded into a list of Shapely Points using the `explode_linestring()` function of the vector module. The function uses the built-in Shapely attribute `linestring.coords` to extract the vertices.

[3]:
```
from shapely.geometry import LineString
import matplotlib.pyplot as plt
import geopandas as gpd
import pandas as pd
import gemgis as gg

linestring = LineString([(0,0), (5,5), (10,0), (15,5)])

linestring
```

[3]:

Exploding the LineString

Exploding the LineString and returning a list of points.

[4]:
```
point_list = gg.vector.explode_linestring(linestring=linestring)
point_list
```

[4]:
```
[<POINT (0 0)>, <POINT (5 5)>, <POINT (10 0)>, <POINT (15 5)>]
```

Inspecting the elements of the returned list

Each element of the list is a Shapely Point object.

[5]:
```
point_list[0]
```

[5]:

[6]:
```
point_list[0].wkt
```

[6]:
```
'POINT (0 0)'
```

Creating GeoDataFrame

A GeoDataFrame is created for better visualization.

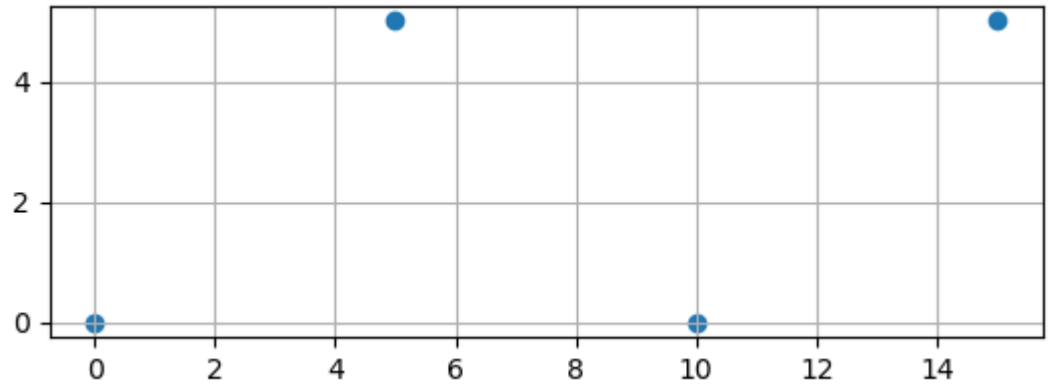
[7]:
```
point_gdf = gpd.GeoDataFrame(geometry=point_list)

point_gdf
```

[7]:
```
                    geometry
0   POINT (0.00000 0.00000)
1   POINT (5.00000 5.00000)
2  POINT (10.00000 0.00000)
3  POINT (15.00000 5.00000)
```

Plotting the Data

The extracted vertices of the lines can now be plotted using `matplotlib`.

[8]:
```
point_gdf.plot()
plt.grid()
```



6.4.3 Exploding LineStrings into single elements

LineStrings can be split into a list of single LineStrings using the `explode_linestring_to_element()` function of the vector module. It is based on the Shapely `split()` function.

[9]:
```
from shapely.geometry import LineString
import matplotlib.pyplot as plt
import geopandas as gpd
import pandas as pd
import gemgis as gg

linestring = LineString([(0,0), (5,5), (10,0), (15,5)])

linestring
```

[9]:

Creating GeoDataFrame

Creating GeoDataFrame from LineString

[10]:
```
linestring_gdf = gpd.GeoDataFrame(geometry=[linestring])

linestring_gdf
```

[10]:
```
                                            geometry
0  LINESTRING (0.00000 0.00000, 5.00000 5.00000, ...
```

Splitting the LineString

A list of single LineStrings will be created when exploding the input LineString. This list can easily be converted to a GeoDataFrame. It can be seen that the input LineString was split into three parts and that the end points of each part coincide with the original LineString vertices.

[11]:
```
linestring_list = gg.vector.explode_linestring_to_elements(linestring=linestring)

linestring_list
```

[11]:
```
[<LINESTRING (0 0, 5 5)>, <LINESTRING (5 5, 10 0)>, <LINESTRING (10 0, 15 5)>]
```

Inspecting the different elements of the returned list

The different elements of the list are Shapely LineString.

[12]:
```
linestring_list[0]
```

[12]:

[13]:
```
linestring_list[0].wkt
```

[13]:
```
'LINESTRING (0 0, 5 5)'
```

[14]:
```
linestring_list[1]
```

[14]:

[15]:
```
linestring_list[1].wkt
```

[15]:
```
'LINESTRING (5 5, 10 0)'
```

[16]:
```
linestring_list[2]
```

[16]:

[17]:
```
linestring_list[2].wkt
```

[17]:
```
'LINESTRING (10 0, 15 5)'
```

Creating GeoDataFrame

Creating a GeoDataFrame from the list of LineStrings.

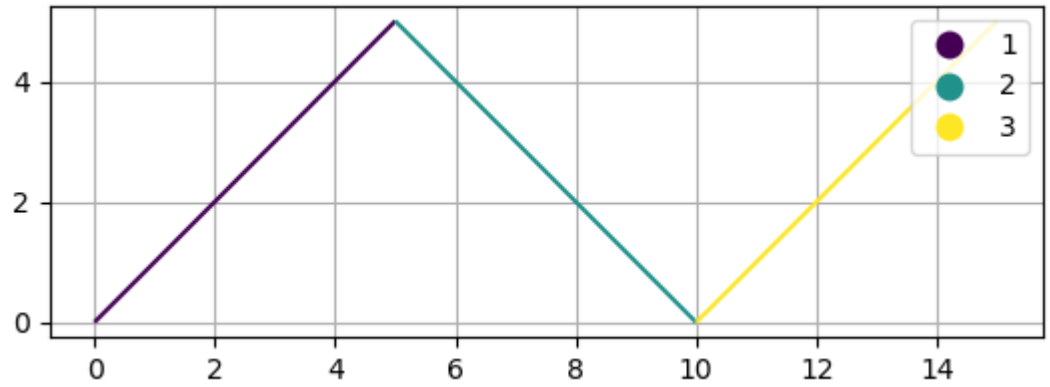
[18]:
```
linestring_gdf = gpd.GeoDataFrame(geometry=linestring_list)

linestring_gdf['id'] = ['1', '2', '3']
linestring_gdf
```

[18]:
```
                                       geometry id
0    LINESTRING (0.00000 0.00000, 5.00000 5.00000)  1
1   LINESTRING (5.00000 5.00000, 10.00000 0.00000)  2
2  LINESTRING (10.00000 0.00000, 15.00000 5.00000)  3
```

Plotting the Result

Plotting the single LineStrings.

[19]:
```
linestring_gdf.plot(column='id', legend=True, cmap='viridis')
plt.grid()
```



6.4.4 Exploding MultiLineStrings

MultiLineStrings can be split into a list of single LineStrings (with multiple vertices) using the `explode_multilinestring()` function of the vector module. This can also be achieved by accessing the LineStrings via `list(multilinestring.coords)`. If MultiLineStrings are provided as a GeoDataFrame, the function `explode_multilinestrings` can be used. This functions uses the built-in `gdf.explode()` function of GeoPandas.

[20]:
```
from shapely.geometry import MultiLineString
import matplotlib.pyplot as plt
import geopandas as gpd
import gemgis as gg

linestrings = [((0,0), (5,5)), ((10,0), (15,5))]

multilinestring = MultiLineString(linestrings)
multilinestring
```

[20]:

Creating GeoDataFrame

Creating a GeoDataFrame from the MultiLineString.

[21]:
```
multilinestring_gdf = gpd.GeoDataFrame(geometry=[multilinestring])

multilinestring_gdf
```

[21]:
```
                                            geometry
0  MULTILINESTRING ((0.00000 0.00000, 5.00000 5.0...
```

Splitting the MultiLineString

A list of single LineStrings (with multiple vertices) will be created when exploding the input MultiLineString. This list can easily be converted to a GeoDataFrame. It can be seen that the input MultiLineString was split into two single LineStrings and that the end points of each part coincide with the original MultiLineString vertices.

[22]:
```
multilinestring_list = gg.vector.explode_multilinestring(multilinestring=multilinestring)

multilinestring_list
```

[22]:
```
[<LINESTRING (0 0, 5 5)>, <LINESTRING (10 0, 15 5)>]
```

Inspecting the elements of the returned list

The elements of the created list are Shapely LineStrings.

[23]:
```
multilinestring_list[0]
```

[23]:

[24]:
```
multilinestring_list[0].wkt
```

[24]:
```
'LINESTRING (0 0, 5 5)'
```

[25]:
```
multilinestring_list[1]
```

[25]:

[26]:
```
multilinestring_list[1].wkt
```

[26]:
```
'LINESTRING (10 0, 15 5)'
```

Creating GeoDataFrame

Creating a GeoDataFrame from the list of LineStrings.

[27]:
```
gdf = gpd.GeoDataFrame(geometry=multilinestring_list)
gdf
```

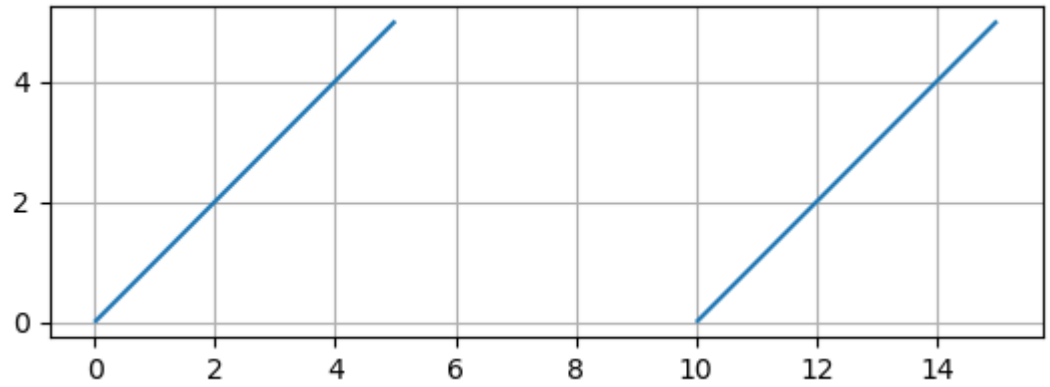
[27]:

	geometry
0	LINESTRING (0.00000 0.00000, 5.00000 5.00000)
1	LINESTRING (10.00000 0.00000, 15.00000 5.00000)

Plotting the Result

Plotting the different LineStrings.

[28]:
```
gdf.plot()
plt.grid()
```



Creating GeoDataFrame

A GeoDataFrame containig multiple MultiLineStrings can be exploded to LineStrings using `explode_multilinestrings(..)`.

[29]:
```
splitted_multilinestring_gdf = gg.vector.explode_multilinestrings(pd.
↪concat([multilinestring_gdf, multilinestring_gdf]).reset_index().drop('index', axis=1))
splitted_multilinestring_gdf
```

[29]:
```
                                        geometry
0    LINESTRING (0.00000 0.00000, 5.00000 5.00000)
1  LINESTRING (10.00000 0.00000, 15.00000 5.00000)
2    LINESTRING (0.00000 0.00000, 5.00000 5.00000)
3  LINESTRING (10.00000 0.00000, 15.00000 5.00000)
```

6.4.5 Additional Arguments

Several additional arguments can be passed to adapt the functionality of the function. For further reference, see the API Reference for extract_xy.

- reset_index (bool)
- drop_level0 (bool)
- drop_level1 (bool)

Original Function

Using the original function but not resetting the index.

[30]:
```
splitted_multilinestring_gdf = gg.vector.explode_multilinestrings(pd.
↪concat([multilinestring_gdf, multilinestring_gdf]).reset_index().drop('index', axis=1),
                                                                  reset_index=False,
                                                                  drop_level0=False,
                                                                  drop_level1=False)
splitted_multilinestring_gdf
```

[30]:
```
                                          geometry
0 0    LINESTRING (0.00000 0.00000, 5.00000 5.00000)
  1  LINESTRING (10.00000 0.00000, 15.00000 5.00000)
1 0    LINESTRING (0.00000 0.00000, 5.00000 5.00000)
  1  LINESTRING (10.00000 0.00000, 15.00000 5.00000)
```

Resetting index but not additional columns

Resetting the index but not dropping the additional columns that were created.

[31]:
```
splitted_multilinestring_gdf = gg.vector.explode_multilinestrings(pd.
↪concat([multilinestring_gdf, multilinestring_gdf]).reset_index().drop('index', axis=1),
                                                                  reset_index=True,
                                                                  drop_level0=False,
                                                                  drop_level1=False)
splitted_multilinestring_gdf
```

[31]:
```
   level_0  level_1                                     geometry
0        0        0    LINESTRING (0.00000 0.00000, 5.00000 5.00000)
1        0        1  LINESTRING (10.00000 0.00000, 15.00000 5.00000)
2        1        0    LINESTRING (0.00000 0.00000, 5.00000 5.00000)
3        1        1  LINESTRING (10.00000 0.00000, 15.00000 5.00000)
```

6.4.6 Exploding Polygons

Polygons can be split into a list of single Points using the `explode_polygon()` function of the vector module. If Polygons are provided as a GeoDataFrame, the function `explode_polygons` can be used. This functions uses the built-in `gdf.boundary` attribute of a GeoDataFrame to convert Polygons into LineStrings and MultiLineStrings.

[32]:
```
from shapely.geometry import Polygon
import matplotlib.pyplot as plt
import geopandas as gpd
import gemgis as gg

polygon = Polygon([(0, 0), (10, 0), (10, 10), (0, 10)])
polygon
```

[32]:

Exploding the Polygon

A list of single Points will be created when exploding the input Polygon. This list can easily be converted to a GeoDataFrame. It can be seen that the input Polygon was split into five single points corresponding to the corner points of the Polygon.

[33]:
```
points_list = gg.vector.explode_polygon(polygon=polygon)
points_list
```

[33]:
```
[<POINT (0 0)>, <POINT (10 0)>, <POINT (10 10)>, <POINT (0 10)>, <POINT (0 0)>]
```

Inspecting the elements of the returned list

The elements of the created list are Shapely Points

[34]:
```
points_list[0]
```

[34]:

[35]:
```
points_list[0].wkt
```

[35]:
```
'POINT (0 0)'
```

Creating GeoDataFrame

Creating a GeoDataFrame from the list of points.

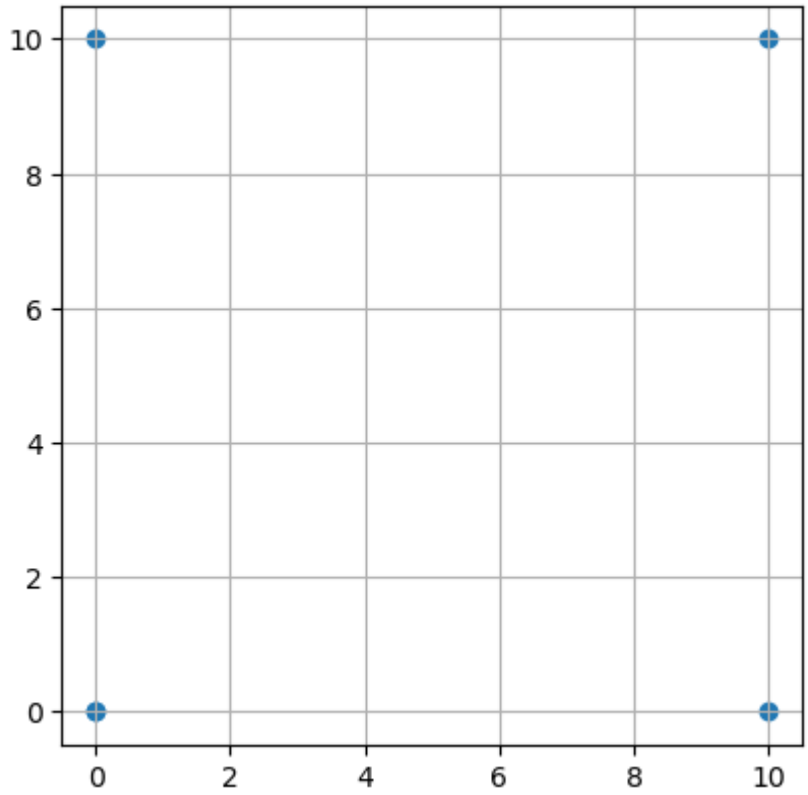
[36]:
```
gdf = gpd.GeoDataFrame(geometry=points_list)
gdf
```

[36]:
```
                    geometry
0    POINT (0.00000 0.00000)
1   POINT (10.00000 0.00000)
2  POINT (10.00000 10.00000)
3   POINT (0.00000 10.00000)
4    POINT (0.00000 0.00000)
```

Plotting the Result

Plotting the five points.

[37]:
```
gdf.plot()
plt.grid()
```



Creating GeoDataFrame

A GeoDataFrame containig multiple polygons can be exploded to LineStrings and MultiLineStrings using `explode_polygons(..)`.

[38]:
```
gdf = gpd.GeoDataFrame(geometry=[polygon, polygon])
gdf
```

[38]:
```
                                            geometry
0  POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1...
1  POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1...
```

Exploding the Polygons

Exploding the polygons into LineStrings and MultiLineStrings.

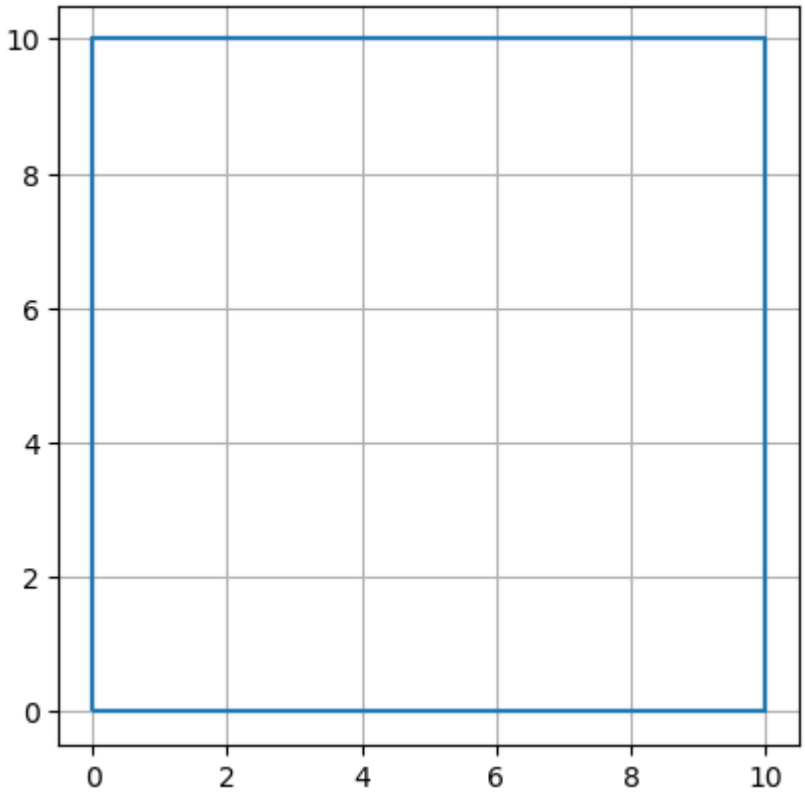
[39]:
```
boundary_gdf = gg.vector.explode_polygons(gdf=gdf)
boundary_gdf
```

[39]:
```
                                            geometry
0  LINESTRING (0.00000 0.00000, 10.00000 0.00000,...
1  LINESTRING (0.00000 0.00000, 10.00000 0.00000,...
```

Plotting the Result

Plotting the LineStrings.

[40]:
```
boundary_gdf.plot()
plt.grid()
```



6.4.7 Exploding Geometry Collections

Shapely Geometry Collections can be exploded into a list of Base Geometries using `explode_geometry_collection(..)`.

[41]:
```
from shapely.geometry import LineString
a = LineString([(0, 0), (1, 1), (1, 2), (2, 2)])
b = LineString([(0, 0), (1, 1), (2, 1), (2, 2)])
collection = a.intersection(b)
type(collection)
```

[41]:
```
shapely.geometry.collection.GeometryCollection
```

[42]:
```
collection
```

[42]:

Explode Geometry Collection

The Shapely Geometry Collection can be exploded to a list of Base Geometries using `explode_geometry_collection(..)`.

[43]:
```
collection_exploded = gg.vector.explode_geometry_collection(collection=collection)
collection_exploded
```

[43]:
```
[<LINESTRING (0 0, 1 1)>, <POINT (2 2)>]
```

Inspecting the elements in the returned list

The elements of the returned list can be inspected to show the exploded geometries of the Geometry Collection.

[44]:
```
collection_exploded[0]
```

[44]:

[45]:
```
collection_exploded[0].wkt
```

[45]:
```
'LINESTRING (0 0, 1 1)'
```

[46]:
```
collection_exploded[1]
```

[46]:

[47]:
```
collection_exploded[1].wkt
```

[47]:
```
'POINT (2 2)'
```

6.4.8 Exploding Collections of different Geometries in GeoDataFrame

A GeoDataFrame containing different Base Geometries can be exploded to single Base Geometries.

[48]:
```
from shapely.geometry import Polygon, LineString

line1 = LineString([(0, 0), (1, 1), (1, 2), (2, 2)])
line2 = LineString([(0, 0), (1, 1), (2, 1), (2, 2)])
collection = a.intersection(b)
polygon = Polygon([(0, 0), (10, 0), (10, 10), (0, 10)])
```

Creating GeoDataFrame

[49]:
```
gdf = gpd.GeoDataFrame(geometry=[line1, line2, collection, polygon])
gdf
```

[49]:
```
                                            geometry
0  LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
1  LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
2  GEOMETRYCOLLECTION (LINESTRING (0.00000 0.0000...
3  POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1...
```

Exploding Geometry Collection GeoDataFrame

A GeoDataFrame containing different Base Geometries or Geometry Collections can be exploded using `explode_geometry_collections(..)`.

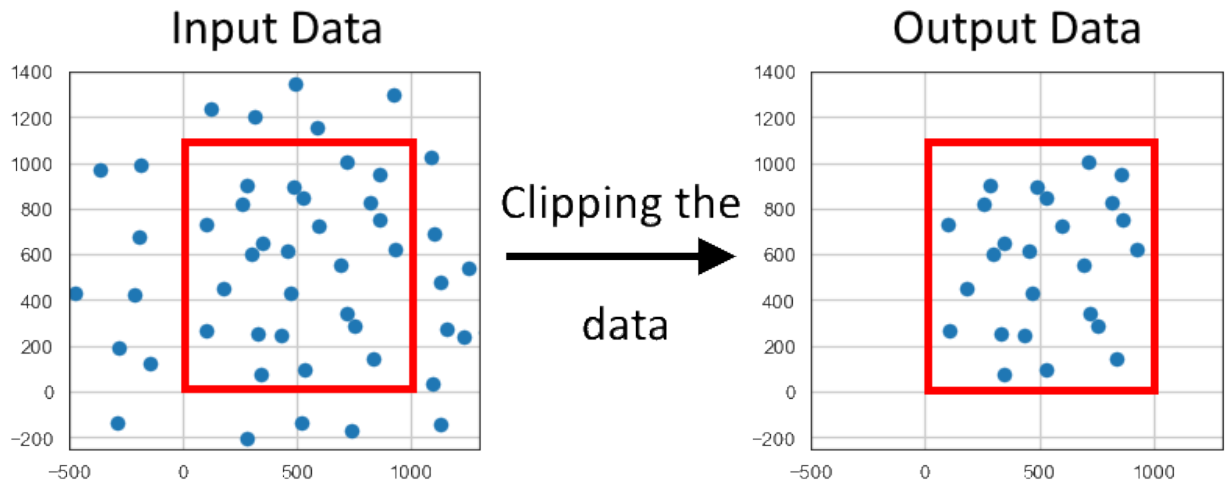
[50]:
```
gdf_exploded = gg.vector.explode_geometry_collections(gdf=gdf)
gdf_exploded
```

[50]:
```
                                              geometry
0  LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
1  LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
2      LINESTRING (0.00000 0.00000, 1.00000 1.00000)
3  POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1...
```

6.5 04 Clipping Vector and Raster Data

Loaded datasets may be cropped or clipped if the spatial extent of data is far beyond what is needed for the tasks at hand. The clipping can be done by either providing a rectengular extent or by providing a Shapely polygon. The clipping can be applied to Point, MultiPoint, LineString, MultiLineString, Polygon and MultiPolygon data.

The vector data is loaded with GeoPandas and the raster data is loaded with rasterio and then clipped with the built-in GemGIS functions.



6.5.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/04_clipping_data/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="04_clipping_data.zip",␣
↪dirpath=file_path)
```

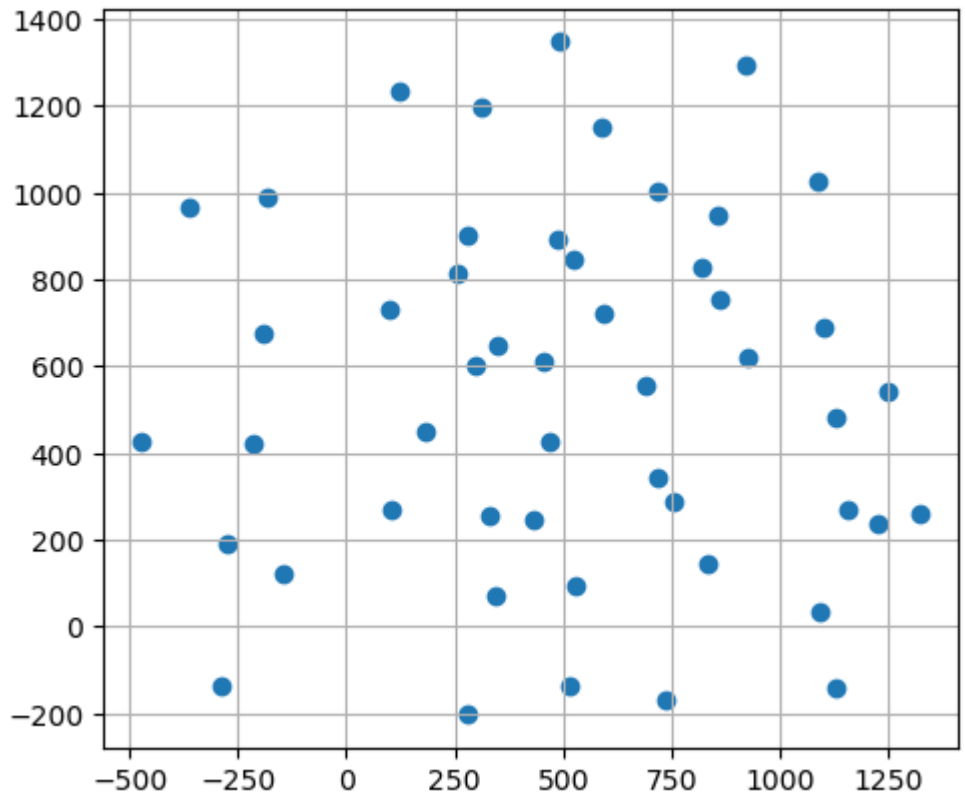
6.5.2 Clipping Vector Data by extent

[3]:
```
import geopandas as gpd
import matplotlib.pyplot as plt

gdf = gpd.read_file(file_path + 'randompoints.shp')

gdf.head()
```

[3]:
```
     id                     geometry
0  None  POINT (281.52576 902.08681)
1  None  POINT (925.86670 618.57679)
2  None  POINT (718.13118 342.79887)
3  None  POINT (331.01114 255.68397)
4  None  POINT (300.08278 600.53525)
```

Plotting the Data

[4]:
```
gdf.plot(aspect='equal')
plt.grid()
```



Setting the extent to which the data will be clipped

[5]:
```
bbox = [0,972, 0, 1069]
```

Clipping the data

The data is clipped with the built-in function `clip_by_bbox()`.

[6]:
```
gdf_clipped = gg.vector.clip_by_bbox(gdf=gdf,
                                     bbox=bbox)

gdf_clipped.head()
```

[6]:

	geometry	X	Y
0	POINT (344.32400 73.45078)	344.32	73.45
1	POINT (529.46815 95.88742)	529.47	95.89
2	POINT (432.04380 246.92094)	432.04	246.92
3	POINT (331.01114 255.68397)	331.01	255.68
4	POINT (718.13118 342.79887)	718.13	342.80

Checking the number of data points

When printing the lengths of the GeoDataFrames, it can that the number of points was reduced to half.

[7]:
```
print(len(gdf))
print(len(gdf_clipped))
```

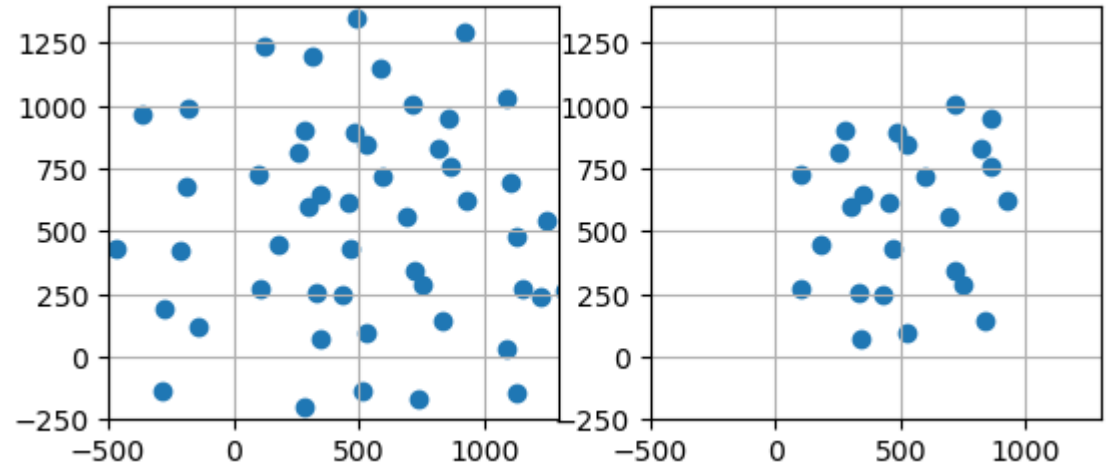
```
50
25
```

Plotting the data

By plotting the data, it can be seen that the number of points have been reduced.

[8]:
```
fig, (ax1, ax2) = plt.subplots(1,2)

gdf.plot(ax=ax1, aspect='equal')
ax1.grid()
ax1.set_xlim(-500, 1300)
ax1.set_ylim(-250, 1400)


gdf_clipped.plot(ax=ax2, aspect='equal')
ax2.grid()
ax2.set_xlim(-500, 1300)
ax2.set_ylim(-250, 1400)
```

[8]:
```
(-250.0, 1400.0)
```



Additional Arguments

Additional arguments can be passed to the function to reset the index and to drop columns. These arguments are true by default.

[9]:
```
gdf_clipped = gg.vector.clip_by_bbox(gdf=gdf,
                                     bbox=bbox,
                                     reset_index=True,
                                     drop_index=True,
                                     drop_id=True,
                                     drop_points=True,
                                     drop_level0=True,
                                     drop_level1=True)

gdf_clipped.head()
```

[9]:
```
                       geometry      X      Y
0   POINT (344.32400 73.45078) 344.32  73.45
1   POINT (529.46815 95.88742) 529.47  95.89
2  POINT (432.04380 246.92094) 432.04 246.92
3  POINT (331.01114 255.68397) 331.01 255.68
4  POINT (718.13118 342.79887) 718.13 342.80
```

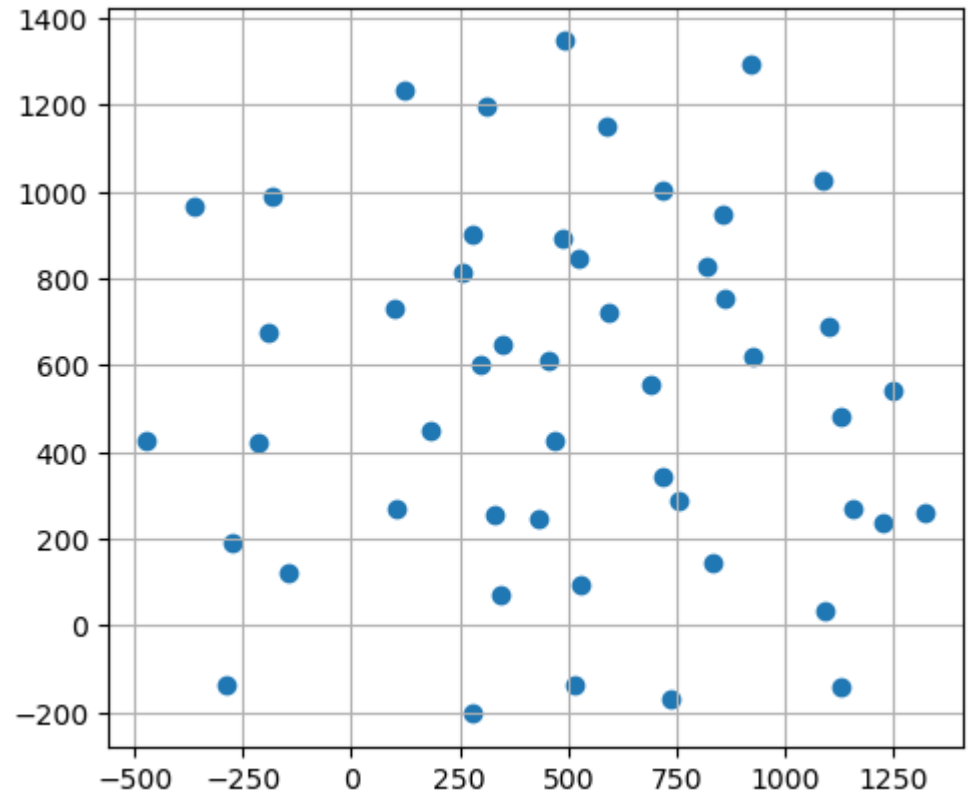
6.5.3 Clipping Vector Data by a polygon

[10]:
```
import geopandas as gpd
import matplotlib.pyplot as plt
import gemgis as gg

gdf = gpd.read_file(file_path + 'randompoints.shp')

gdf.head()
```

[10]:
```
     id                     geometry
0  None  POINT (281.52576 902.08681)
1  None  POINT (925.86670 618.57679)
2  None  POINT (718.13118 342.79887)
3  None  POINT (331.01114 255.68397)
4  None  POINT (300.08278 600.53525)
```

Plotting the Data

[11]:
```
gdf.plot(aspect='equal')
plt.grid()
```



Setting the polygon to which the data will be clipped

[12]:
```
from shapely.geometry import Polygon

polygon = Polygon([(0,0),(972, 0), (972,1069), (0, 1069)])

polygon.wkt
```

[12]:
```
'POLYGON ((0 0, 972 0, 972 1069, 0 1069, 0 0))'
```

Clipping the data

The data is clipped with the built-in function `clip_by_polygon()`.

[13]:
```
gdf_clipped = gg.vector.clip_by_polygon(gdf=gdf,
                                        polygon=polygon)

gdf_clipped.head()
```

[13]:
```
                       geometry
0   POINT (344.32400 73.45078)
1   POINT (529.46815 95.88742)
2  POINT (432.04380 246.92094)
3  POINT (331.01114 255.68397)
4  POINT (718.13118 342.79887)
```

Checking the number of data points

When printing the lengths of the GeoDataFrames, it can that the number of points was reduced to half.

[14]:
```
print(len(gdf))
print(len(gdf_clipped))
```

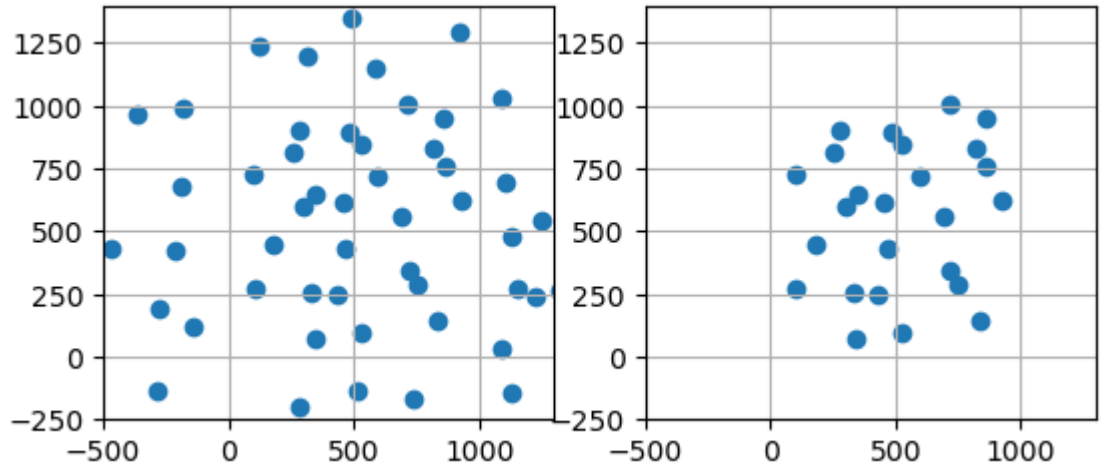
```
50
25
```

Plotting the data

By plotting the data, it can be seen that the number of points have been reduced.

[15]:
```
fig, (ax1, ax2) = plt.subplots(1,2)

gdf.plot(ax=ax1, aspect='equal')
ax1.grid()
ax1.set_xlim(-500, 1300)
ax1.set_ylim(-250, 1400)


gdf_clipped.plot(ax=ax2, aspect='equal')
ax2.grid()
ax2.set_xlim(-500, 1300)
ax2.set_ylim(-250, 1400)
```

[15]:
```
(-250.0, 1400.0)
```



Additional Arguments

Additional arguments can be passed to the function to reset the index and to drop columns. These arguments are true by default.

[16]:
```
gdf_clipped = gg.vector.clip_by_polygon(gdf=gdf,
                                        polygon=polygon,
                                        reset_index=True,
                                        drop_index=True,
                                        drop_id=True,
                                        drop_points=True,
                                        drop_level0=True,
                                        drop_level1=True)

gdf_clipped.head()
```

[16]:
```
                        geometry
0   POINT (344.32400 73.45078)
1   POINT (529.46815 95.88742)
2  POINT (432.04380 246.92094)
3  POINT (331.01114 255.68397)
4  POINT (718.13118 342.79887)
```

6.5.4 Clipping Raster Data by extent

[17]:
```
import rasterio
import gemgis as gg

raster = rasterio.open(file_path + 'raster.tif')

raster.read()
```

[17]:
```
array([[[482.82904, 485.51953, 488.159  , ..., 618.8612 , 620.4424 ,
         622.05786],
```

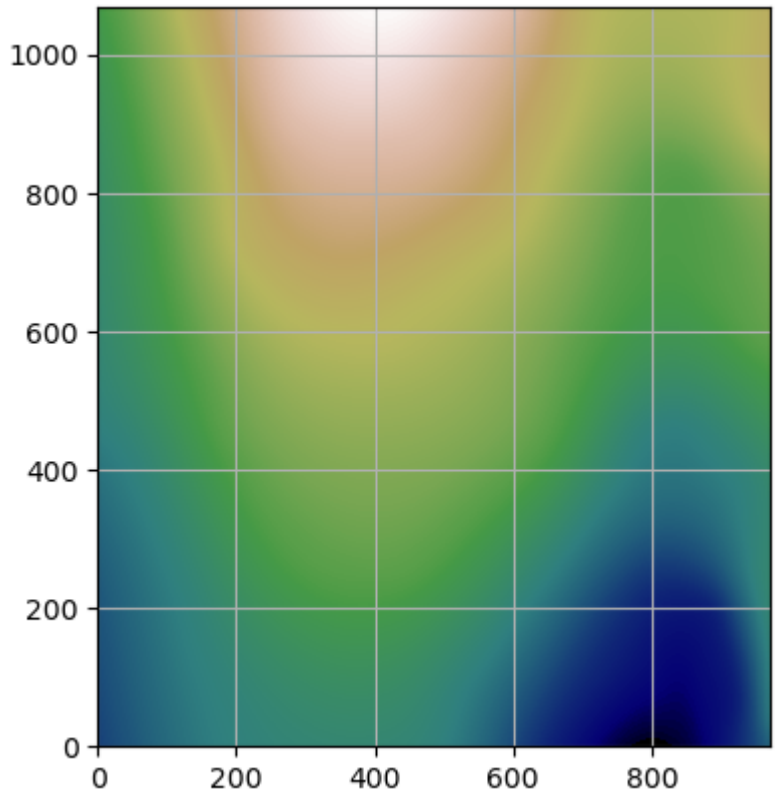
(continues on next page)

(continued from previous page)

```
        [481.6521 , 484.32193, 486.93958, ..., 618.8579 , 620.44556,
         622.06714],
        [480.52563, 483.18893, 485.80444, ..., 618.8688 , 620.4622 ,
         622.08923],
        ...,
        [325.49225, 327.21985, 328.94498, ..., 353.6889 , 360.03125,
         366.3984 ],
        [325.0538 , 326.78473, 328.51276, ..., 351.80603, 357.84106,
         363.96167],
        [324.61444, 326.34845, 328.0794 , ..., 350.09247, 355.87598,
         361.78635]]], dtype=float32)
```

Plotting the input raster

[18]:
```
plt.imshow(raster.read(1), cmap='gist_earth', extent= [0,972,0,1069])
plt.grid()
```



Setting the extent to which the data will be clipped

[19]:
```
bbox = [250,750,250,750]
```

Clipping the raster

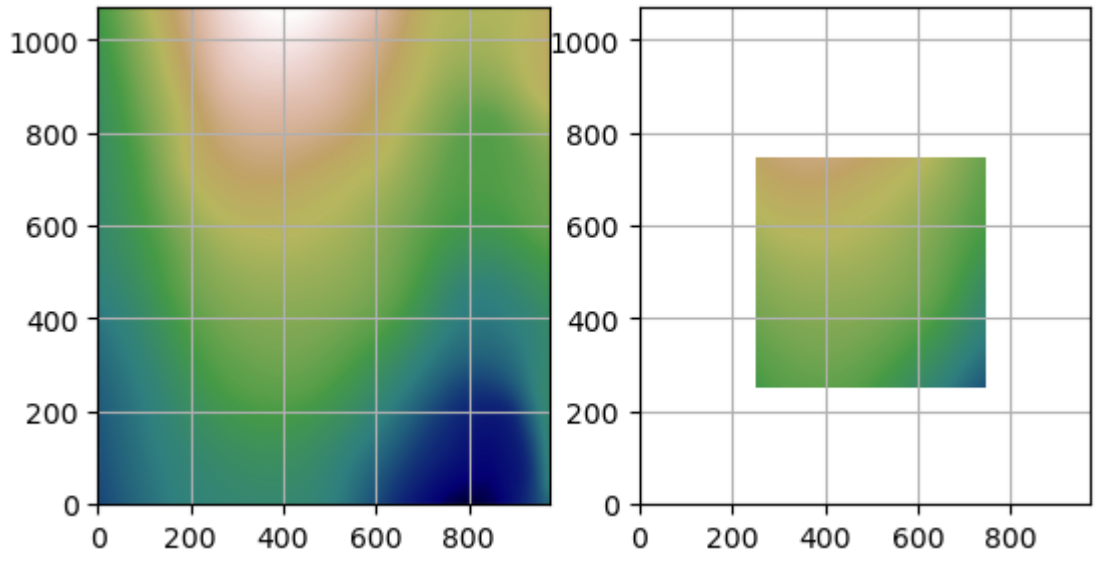
The data is clipped with the built-in function `clip_by_bbox()`.

[20]:
```
raster_clipped = gg.raster.clip_by_bbox(raster=raster,
                                        bbox=bbox)
```

Plotting the result

[21]:
```
fig, (ax1, ax2) = plt.subplots(1,2)

ax1.imshow(raster.read(1), cmap='gist_earth', extent= [0,972,0,1069], vmin=250, vmax=750)
ax1.grid()
ax1.set_xlim(0,972)
ax1.set_ylim(0,1069)


ax2.imshow(raster_clipped, cmap='gist_earth', extent= [250,750,250,750], vmin=250,␣
↪vmax=750)
ax2.grid()
ax2.set_xlim(0, 972)
ax2.set_ylim(0, 1069)
```

[21]:
```
(0.0, 1069.0)
```



Additional Arguments

Additional arguments can be passed to directly save the clipped raster to disc.

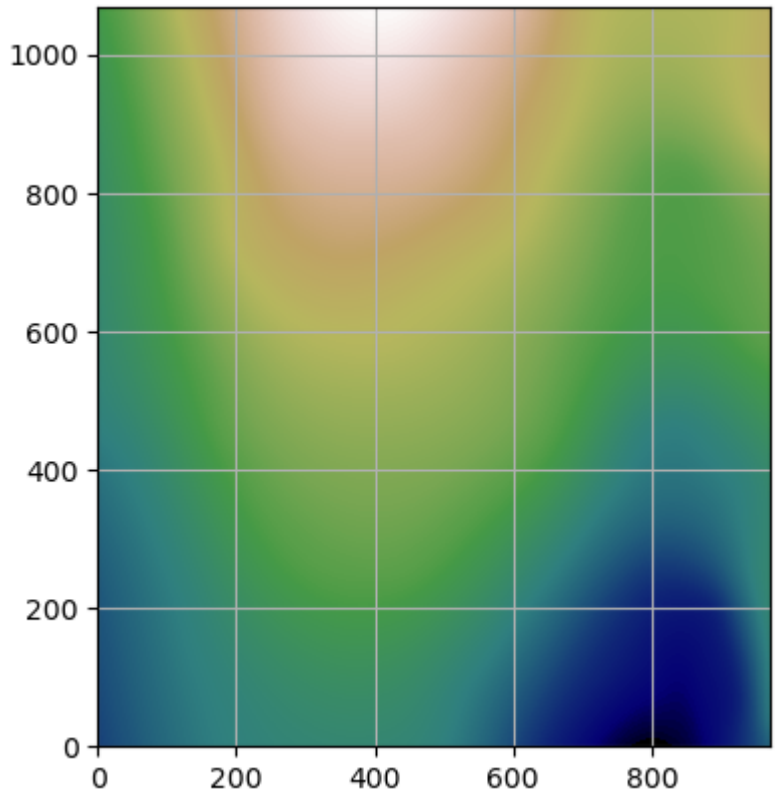
[22]:
```
raster_clipped = gg.raster.clip_by_bbox(raster=raster,
                                        bbox=bbox,
                                        raster_extent=None,
                                        save_clipped_raster=False,
                                        path='raster_clipped.tif')
```

6.5.5 Clipping Raster Data by polygon

[23]:
```
import rasterio
import gemgis as gg

raster = rasterio.open(file_path + 'raster.tif')
```

Plotting the input raster

[24]:
```
plt.imshow(raster.read(1), cmap='gist_earth', extent= [0,972,0,1069])
plt.grid()
```



Setting the extent to which the data will be clipped

[25]:
```
from shapely.geometry import Polygon

polygon = Polygon([(250,250),(750,250), (750,750), (250, 750)])

polygon
```

[25]:

Clipping the raster

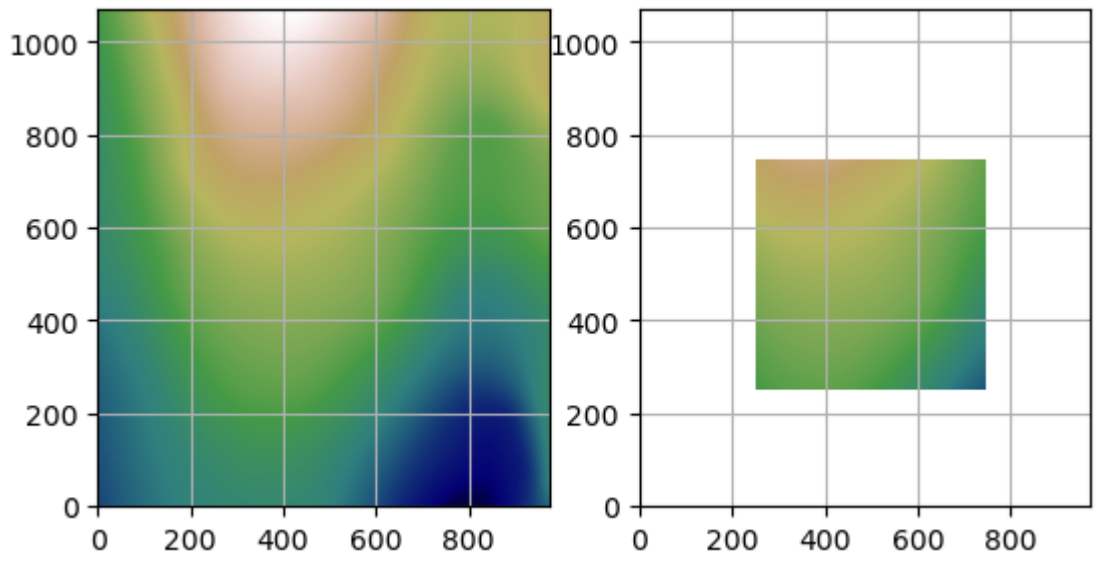
The data is clipped with the built-in function `clip_by_bbox()`.

[26]:
```
raster_clipped = gg.raster.clip_by_polygon(raster=raster,
                                           polygon=polygon)
```

Plotting the result

[27]:
```
fig, (ax1, ax2) = plt.subplots(1,2)

ax1.imshow(raster.read(1), cmap='gist_earth', extent= [0,972,0,1069], vmin=250, vmax=750)
ax1.grid()
ax1.set_xlim(0,972)
ax1.set_ylim(0,1069)


ax2.imshow(raster_clipped, cmap='gist_earth', extent= [250,750,250,750], vmin=250,␣
↪vmax=750)
ax2.grid()
ax2.set_xlim(0, 972)
ax2.set_ylim(0, 1069)
```

[27]:
```
(0.0, 1069.0)
```



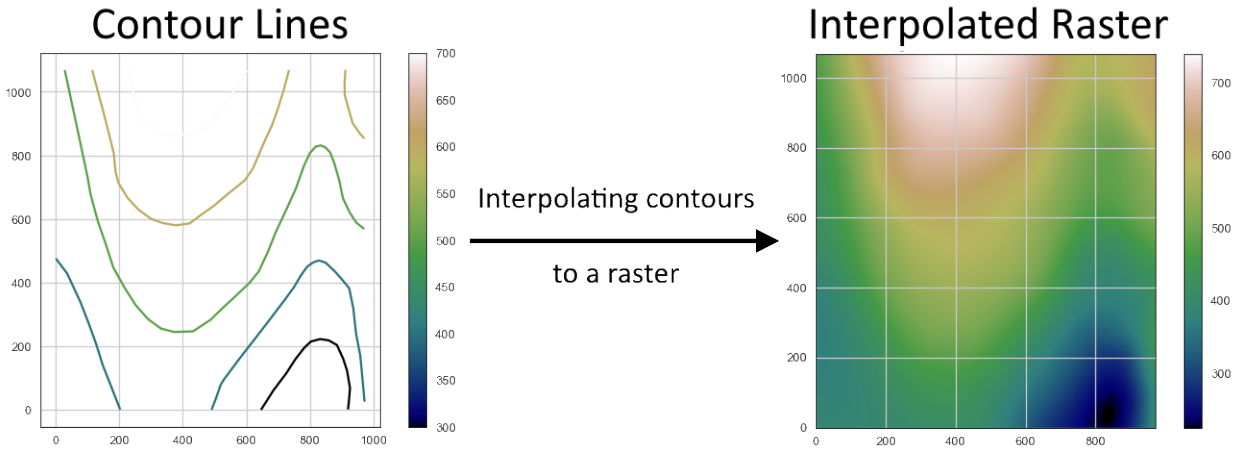
Additional Arguments

Additional arguments can be passed to directly save the clipped raster to disc.

[28]:
```
raster_clipped = gg.raster.clip_by_polygon(raster=raster,
                                           polygon=polygon,
                                           raster_extent=None,
                                           save_clipped_raster=False,
                                           path='raster_clipped.tif')
```

6.6 05 Interpolating Rasters

In case the digital elevation model of a simple geological map needs to be interpolated from topographic contours, several methods have been implemented in GemGIS. These include the methods `nearest`, `cubic` and `linear` of `scipy.interpolate.griddata` and `Rbf` of `scipy.interpolate.rbf`. The different methods can be accessed by passing the argument `method= 'method_name'`. More information about the interpolation methods can be found here.



6.6.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/05_interpolating_rasters/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="05_interpolating_rasters.zip",␣
↪dirpath=file_path)
```

```
Downloading file '05_interpolating_rasters.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F05_interpolating_rasters.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\tutorial\data\05_interpolating_rasters'.
```

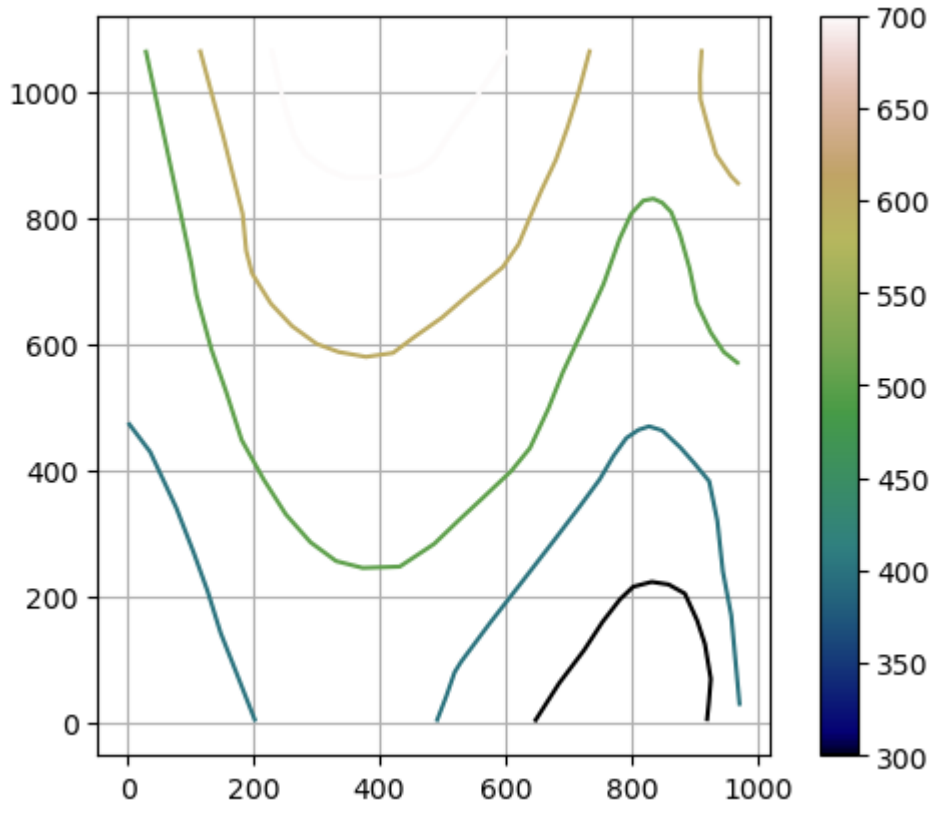
6.6.2 Loading Data

[3]:
```
import geopandas as gpd

contours = gpd.read_file(file_path + 'topo.shp')

contours.head()
```

[3]:
```
     id    Z                                                geometry
0  None  400  LINESTRING (0.74088 475.44101, 35.62873 429.24...
1  None  300  LINESTRING (645.96500 0.52496, 685.14093 61.86...
2  None  400  LINESTRING (490.29223 0.52496, 505.75641 40.73...
3  None  600  LINESTRING (911.43347 1068.58451, 908.85610 10...
4  None  700  LINESTRING (228.43207 1068.58451, 239.77247 10...
```

6.6.3 Plotting Data

[4]:
```
import matplotlib.pyplot as plt

contours.plot(aspect='equal',column='Z', cmap='gist_earth', legend=True)
plt.grid()
```



6.4.4 Interpolating the Raster

It is recommended to use the method `rbf` for the interpolation of contour lines. More information about RBF can be found here in this paper.

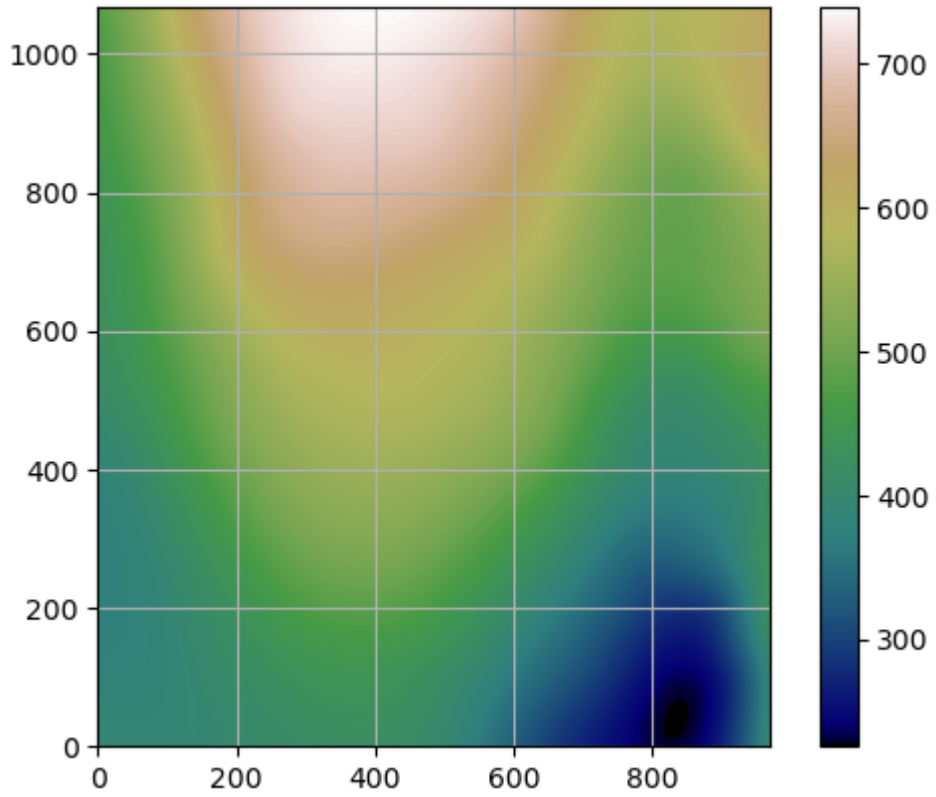
[5]:
```
raster = gg.vector.interpolate_raster(gdf=contours,
                                      method='rbf')
raster[:2]
```

[5]:
```
array([[393.56371914, 393.50838517, 393.45386851, ..., 396.15856133,
        398.11421775, 400.06334288],
       [393.41982945, 393.36494644, 393.31088433, ..., 396.20694282,
        398.16690286, 400.12027997]])
```

6.4.5 Plotting the interpolated raster

[6]:
```
im = plt.imshow(raster, cmap='gist_earth', origin='lower')
plt.grid()
plt.colorbar(im)
```

[6]:
```
<matplotlib.colorbar.Colorbar at 0x21cf685bac0>
```



6.6.6 Interpolating the Raster

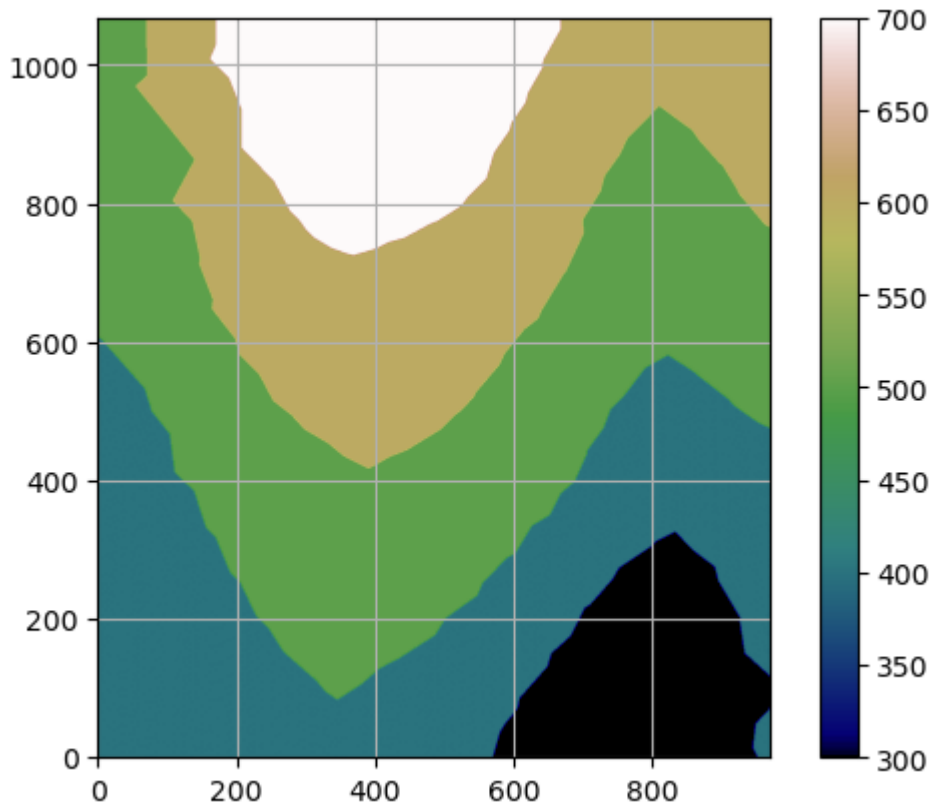
Using different interpolation functions such as `nearest` will result in different, more unrealistic results.

[7]:
```
raster = gg.vector.interpolate_raster(gdf=contours,
                                      method='nearest')
```

6.6.7 Plotting the interpolated raster

[8]:
```
im = plt.imshow(raster, cmap='gist_earth', origin='lower')
plt.grid()
plt.colorbar(im)
```

[8]:
```
<matplotlib.colorbar.Colorbar at 0x21cf68ba590>
```



6.6.8 Additional Arguments

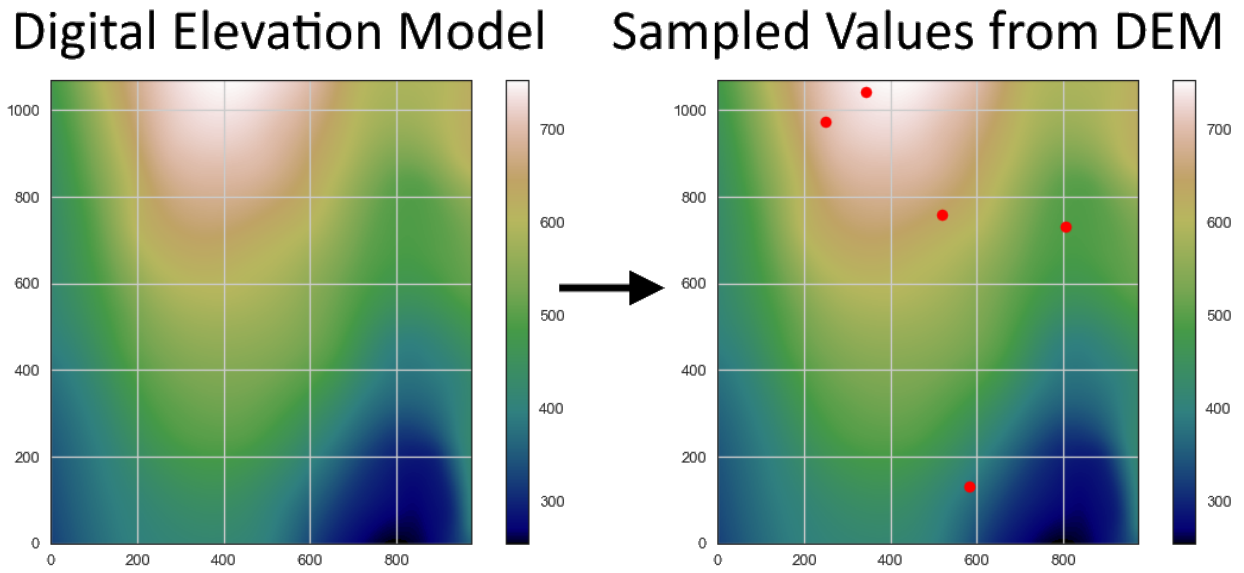
Several additional arguments can be passed to adapt the functionality of the function. For further reference, see the API Reference for extract_xy.

- n - Number of points, default all
- res - Resolution of the interpolated raster, default 1
- extent - Extent of the interpolated raster, default according to the input data
- seed - seed for drawing random values

- kwargs - optional keyword arguments for rbf and griddata interpolation functions - https://docs.scipy.org/doc/scipy/reference/interpolate.html

6.7 06 Sampling from Rasters

It may be important for some tasks to sample values of a raster at a certain point, e.g. an orientation measurement. The functionality to sample from an array or rasterio object directly or random sampling has been implemented in GemGIS and will be introduced here.



6.7.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/06_sampling_from_rasters/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="06_sampling_from_rasters.zip",␣
↪dirpath=file_path)
```

```
Downloading file '06_sampling_from_rasters.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F06_sampling_from_rasters.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\tutorial\data\06_sampling_from_rasters'.
```

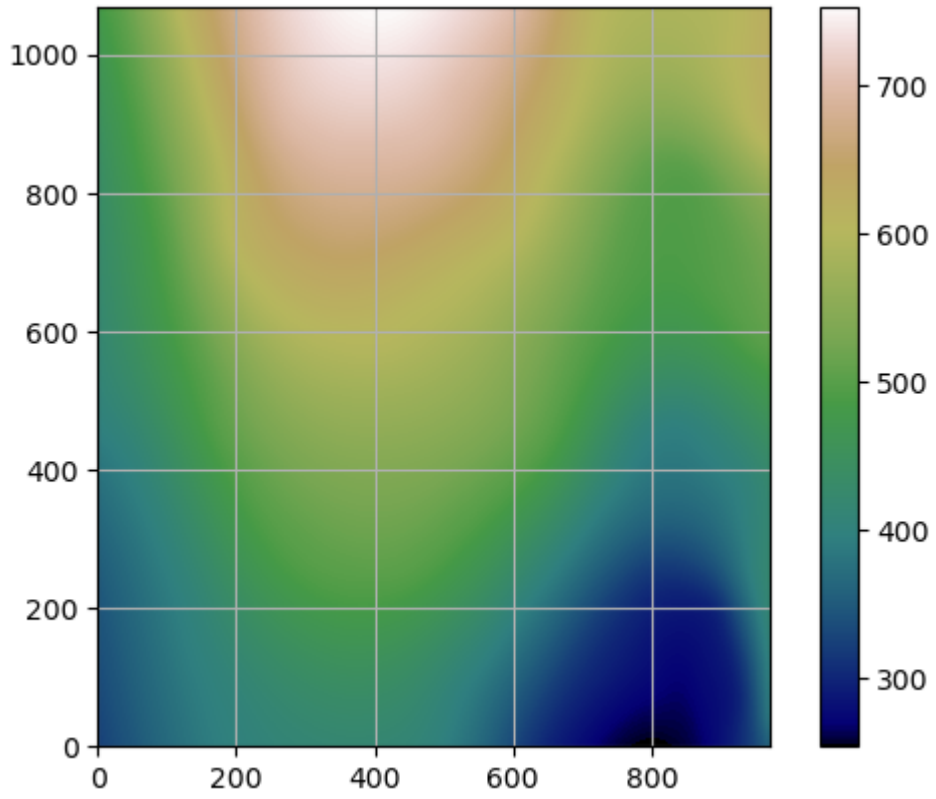
6.7.2 Loading the raster

[3]:
```
import rasterio
import numpy as np

raster = rasterio.open(file_path + 'raster.tif')
```

6.7.3 Plotting the raster

[4]:
```
import matplotlib.pyplot as plt

im = plt.imshow(raster.read(1), cmap='gist_earth',extent = [0, 972, 0, 1069])
plt.grid()
plt.colorbar(im)
```

[4]:
```
<matplotlib.colorbar.Colorbar at 0x179b6974100>
```



6.7.4 Sampling from Array

The function `gg.raster.sample_from_array(..)` will be used to sample a single point or given lists/arrays of coordinates from an array.

Sampling from one given point.

[5]:
```
sample = gg.raster.sample_from_array(array = raster.read(1),
                                     extent = [0, 972, 0, 1069],
                                     point_x = 500,
                                     point_y = 500)

sample
```

[5]:
```
562.0227
```

Sampling from given lists of x and y coordinates.

[6]:
```
point_x = [100, 200, 300, 400 ,500]
point_y = [100, 200, 300, 400 ,500]

sample = gg.raster.sample_from_array(array = raster.read(1),
                                     extent = [0, 972, 0, 1069],
                                     point_x = point_x,
                                     point_y = point_y)

sample
```

[6]:
```
array([378.79877, 435.21704, 505.29126, 541.74146, 562.0227 ],
      dtype=float32)
```

Sampling from given arrays of x and y coordinates.

[7]:
```
point_x = np.array([100, 200, 300, 400 ,500.])
point_y = np.array([100, 200, 300, 400 ,500.])

sample = gg.raster.sample_from_array(array = raster.read(1),
                                     extent = [0, 972, 0, 1069],
                                     point_x = point_x,
                                     point_y = point_y)

sample
```

[7]:
```
array([378.79877, 435.21704, 505.29126, 541.74146, 562.0227 ],
      dtype=float32)
```

6.7.5 Sampling from Raster

The function `gg.raster.sample_from_raster(..)` will be used to sample a single point or given lists/arrays of coordinates from raster.

Sampling from one given point.

[8]:
```
sample = gg.raster.sample_from_rasterio(raster = raster,
                                        point_x = 500,
                                        point_y = 500)

sample
```

[8]:
```
561.646728515625
```

Sampling from given lists of x and y coordinates.

[9]:
```
point_x = [100, 200, 300, 400 ,500]
point_y = [100, 200, 300, 400 ,500]

sample = gg.raster.sample_from_rasterio(raster = raster,
                                        point_x = point_x,
                                        point_y = point_y)

sample
```

[9]:
```
[376.72430419921875,
 435.217041015625,
 505.291259765625,
 540.6767578125,
 561.646728515625]
```

Sampling from given arrays of x and y coordinates.

[10]:
```
point_x = np.array([100, 200, 300, 400 ,500.])
point_y = np.array([100, 200, 300, 400 ,500.])

sample = gg.raster.sample_from_rasterio(raster = raster,
                                        point_x = point_x,
                                        point_y = point_y)

sample
```

[10]:
```
[376.72430419921875,
 435.217041015625,
 505.291259765625,
 540.6767578125,
 561.646728515625]
```

6.7.6 Sample Randomly from Raster

The function `gg.raster.sample_randomly(..)` will be used to sample one or multiple points from an array or a raster.

Sample one point randomly from raster.

[11]:
```
sample = gg.raster.sample_randomly(raster=raster,
                                   n=1)

sample
```

[11]:
```
(559.26953125, [315.92081696676496, 479.1266134083539])
```

Sample multiple points randomly from raster.

[12]:
```
sample = gg.raster.sample_randomly(raster=raster,
                                   n=5)

sample
```

[12]:
```
([349.976806640625,
  593.2823486328125,
  469.182861328125,
  517.493896484375,
  457.33819580078125],
 [array([656.19476672, 428.41935331, 398.478624  , 419.74758082,
         117.59572753]),
  array([148.35611075, 563.45782055, 165.23874243, 307.06250898,
         453.00801777])])
```

6.7.7 Sample Randomly from Array

Sample one point randomly from array.

[13]:
```
sample = gg.raster.sample_randomly(raster=raster.read(1),
                                   n=1,
                                   extent=[0, 972, 0, 1069])

sample
```

[13]:
```
(410.9246826171875, [231.80451313385362, 47.36589009664959])
```

Sample multiple points randomly from array.

[14]:
```
sample = gg.raster.sample_randomly(raster=raster.read(1),
                                   n=5,
                                   extent=[0, 972, 0, 1069])

sample
```

[14]:
```
([292.591064453125,
  608.958740234375,
  604.081298828125,
  482.37969970703125,
```

(continues on next page)

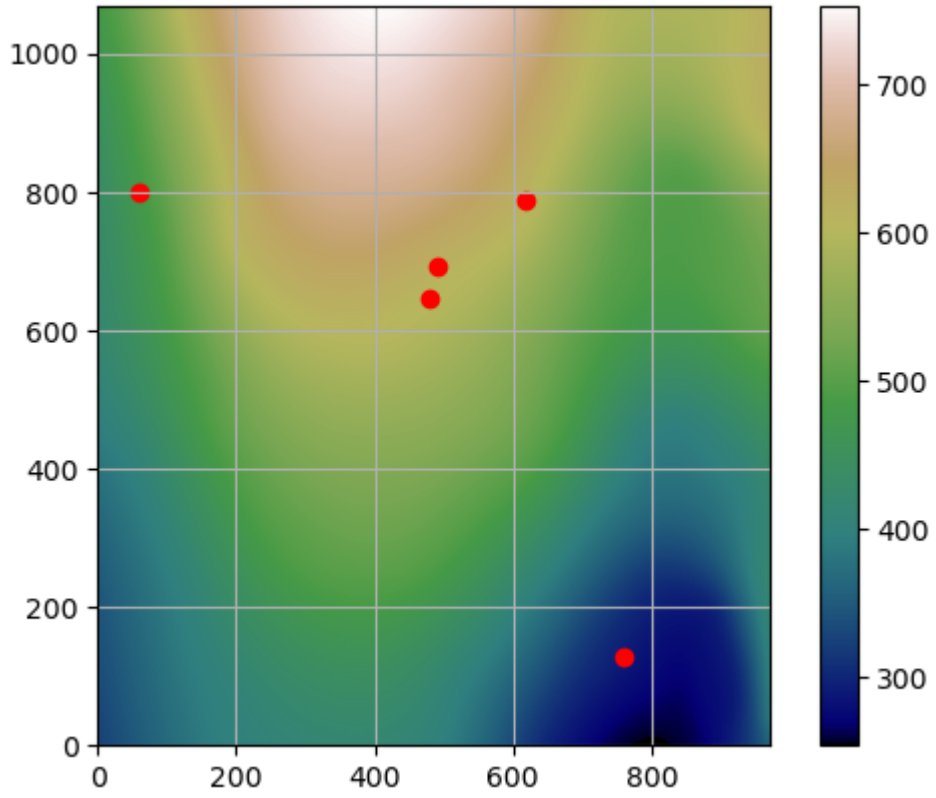
(continued from previous page)

```
 617.5330810546875],
[array([758.6160817 , 617.25201554, 479.26358106,  60.94460541,
        489.87045745]),
 array([128.41568426, 788.7398796 , 644.85742373, 799.84713491,
        692.40925628])])
```

6.7.8 Plotting the points on the raster

[15]:
```
import matplotlib.pyplot as plt

im = plt.imshow(raster.read(1), cmap='gist_earth',extent = [0, 972, 0, 1069])
plt.scatter(sample[1][0], sample[1][1], color='red')
plt.grid()
plt.colorbar(im)
```

[15]:
```
<matplotlib.colorbar.Colorbar at 0x179b7be5f60>
```

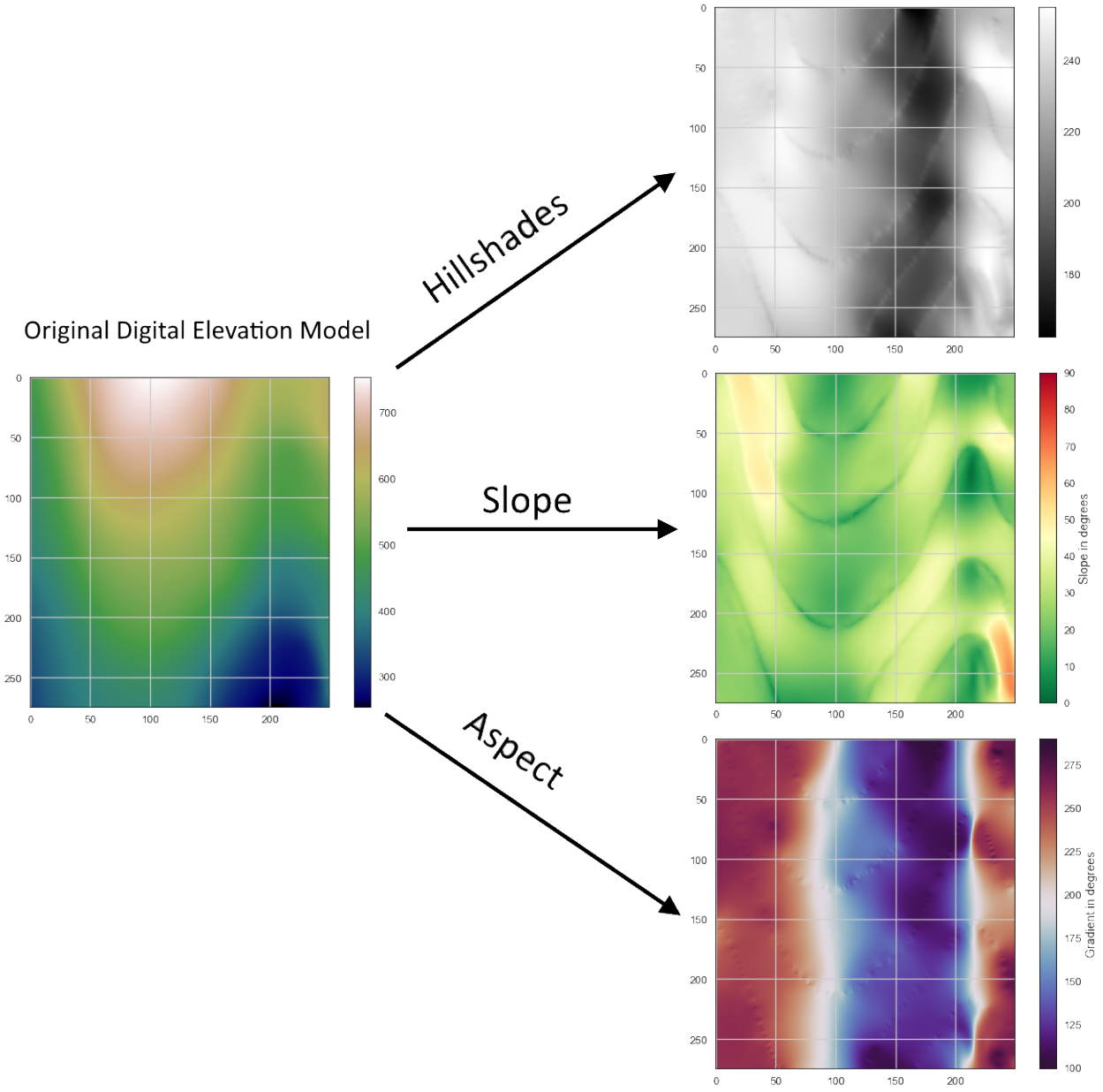


6.7.9 Sampling slope and aspect values

In a same way, slope and aspect values can be sampled from a raster when using `calculate_slope(..)` and `calculate_hillshades(..)`.

6.8 07 Calculating Raster Properties

Raster properties such as hillshades for better visualization of the digital elevation model and slope/aspect for the calculation of orientation values can easily be calculated in GemGIS. Hillshades are a 3D grayscale representation of the topography defined by a light source with a given azimuth and altitude. The slope defines the gradient of the raster and the aspect defines the direction of the gradient. Hence, the combination of both paramters results in an orientation value.



6.8.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/07_calculating_raster_properties/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="07_calculating_raster_
↪properties.zip", dirpath=file_path)
```

```
Downloading file '07_calculating_raster_properties.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2F07_calculating_raster_properties.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\data\07_calculating_
↪raster_properties'.
```

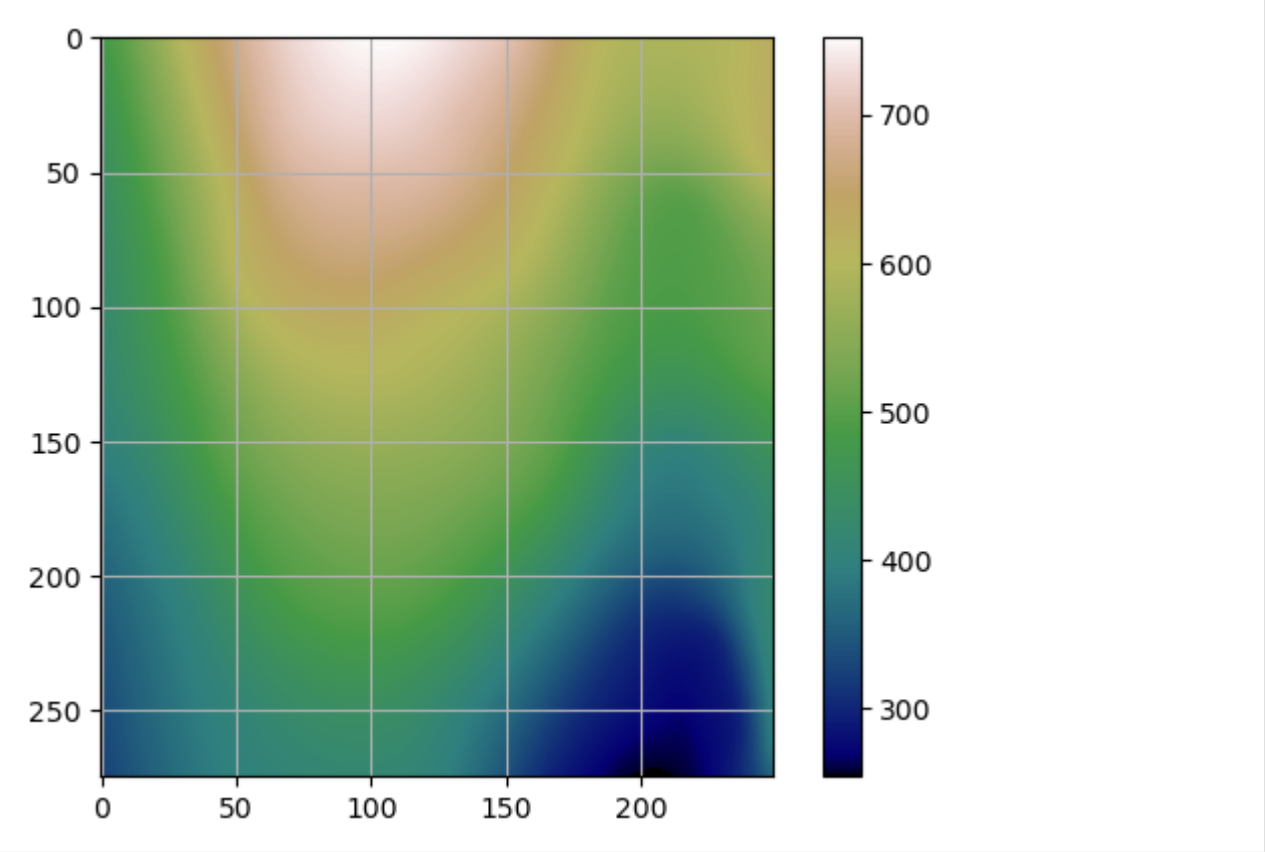
6.8.2 Loading the raster

[3]:
```
import rasterio

raster = rasterio.open(file_path + 'raster.tif')
```

6.8.3 Plotting the raster

[4]:
```
import matplotlib.pyplot as plt

im = plt.imshow(raster.read(1), cmap='gist_earth')
plt.grid()
plt.colorbar(im)
```

[4]:
```
<matplotlib.colorbar.Colorbar at 0x24793033fd0>
```



6.8.4 Calculate hillshades

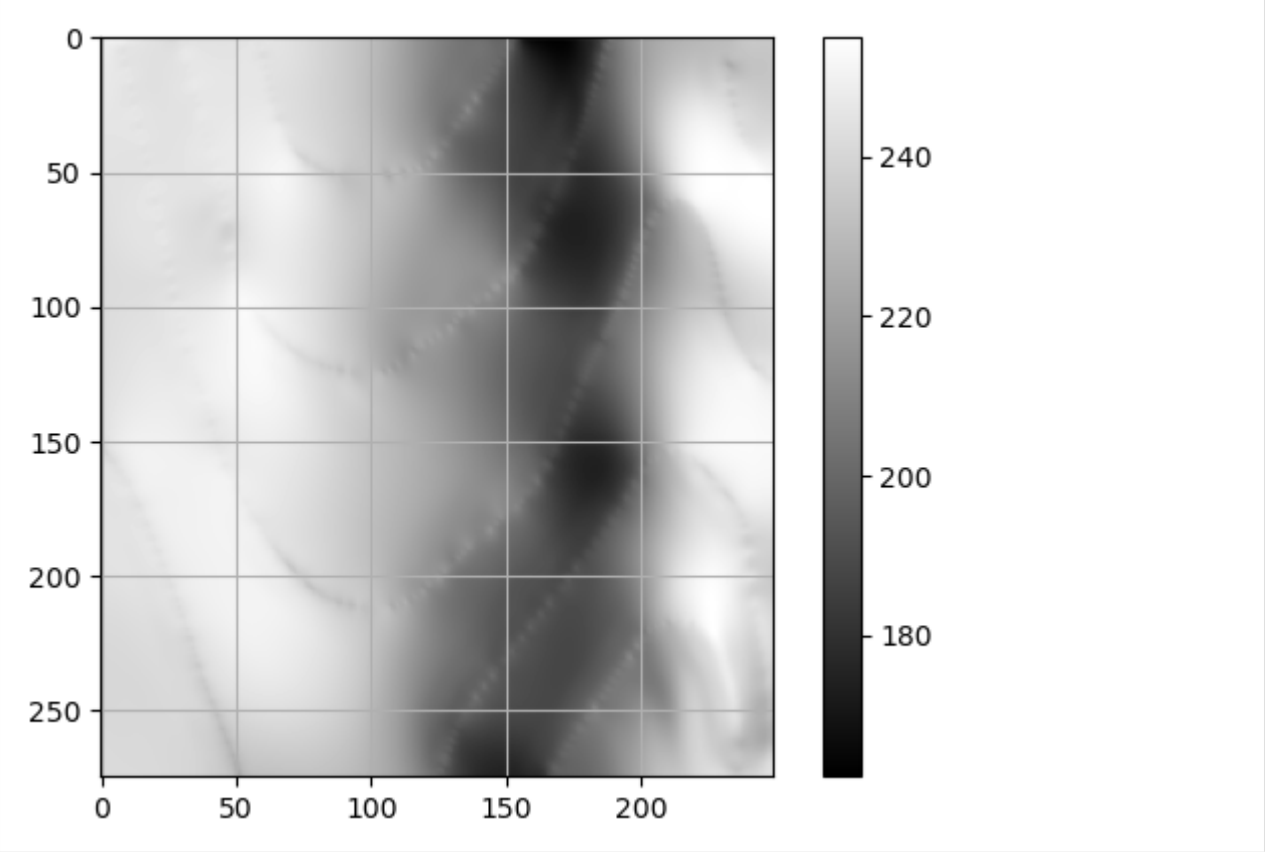
The hillshades can be calculated using `calculate_hillshades(..)`.

[5]:
```
hillshades = gg.raster.calculate_hillshades(raster)
```

6.8.5 Plotting hillshades

[6]:
```
im = plt.imshow(hillshades, cmap='gray')
plt.grid()
plt.colorbar(im)
```

[6]:
```
<matplotlib.colorbar.Colorbar at 0x247942e96c0>
```



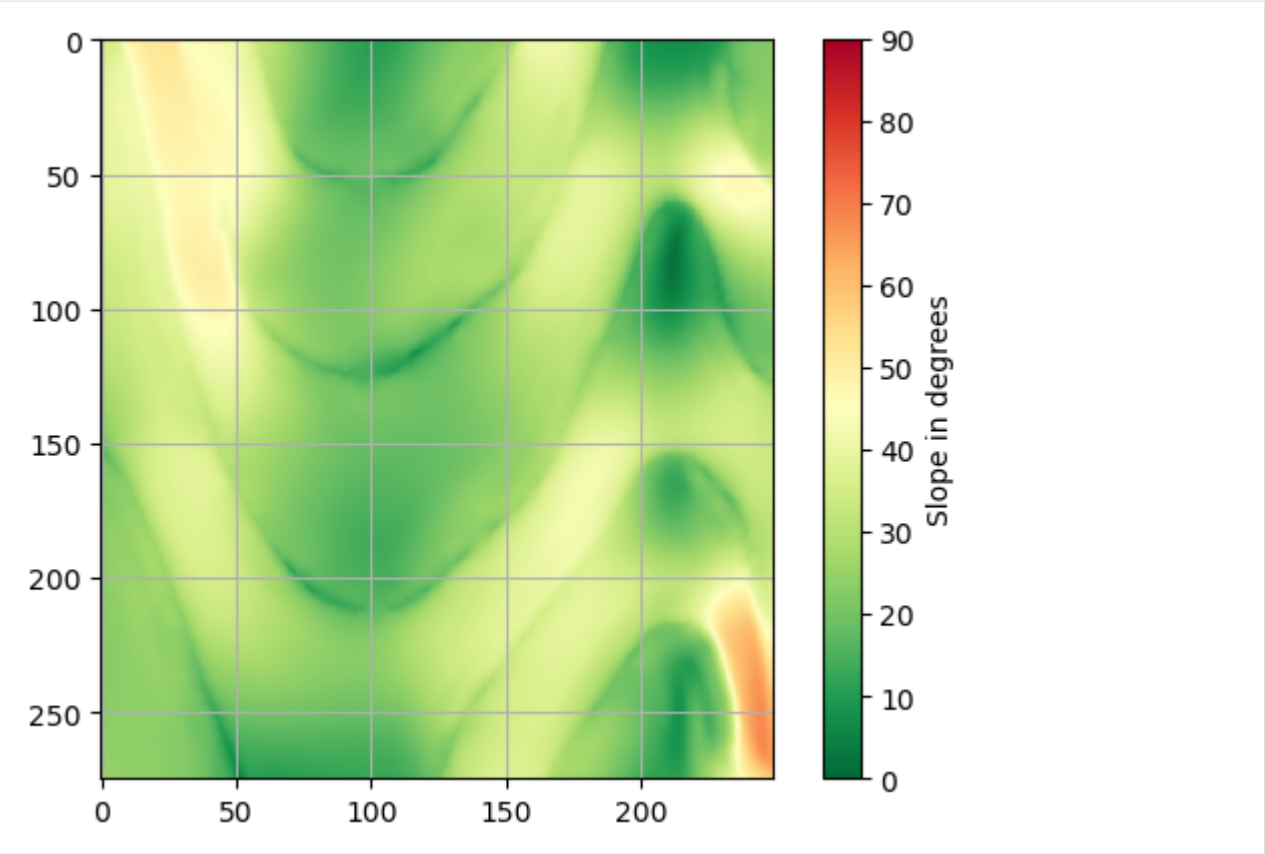
6.8.6 Calculate slope

The slope can be calculated using `calculate_slope(..)`.

[7]:
```
slope = gg.raster.calculate_slope(raster)
```

6.8.7 Plotting hillshades

[8]:
```
im = plt.imshow(slope, cmap='RdYlGn_r', vmin=0, vmax=90)
plt.grid()
cbar = plt.colorbar(im)
cbar.set_label('Slope in degrees')
```



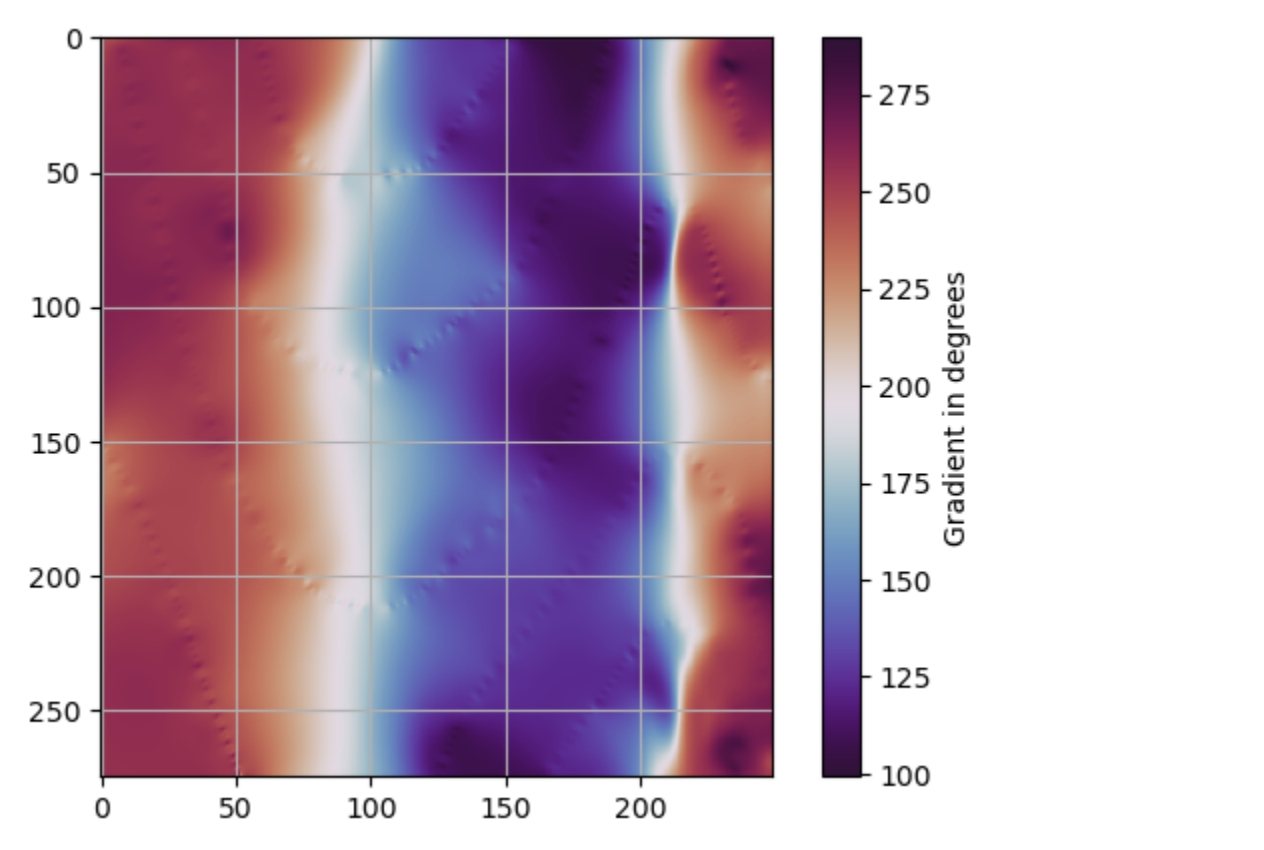
6.8.8 Calculate aspect

The aspect can be calculated using `calculate_aspect(..)`.

[9]:
```
aspect = gg.raster.calculate_aspect(raster)
```

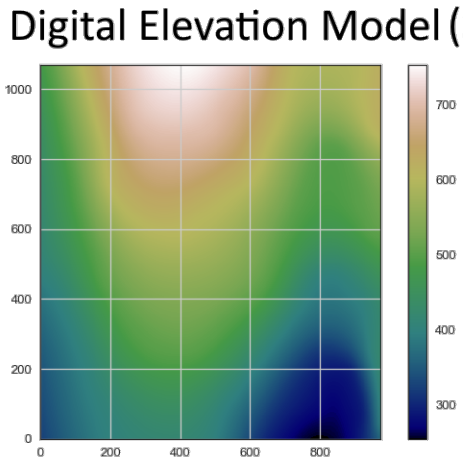
6.8.9 Plotting hillshades

[10]:
```
im = plt.imshow(aspect, cmap='twilight_shifted')
plt.grid()
cbar = plt.colorbar(im)
cbar.set_label('Gradient in degrees')
```



6.9 08 Sampling Interfaces and Orientations from Raster

Interfaces can be directly sampled from a digital elevation model or subsurface raster/mesh while orientations have to be sampled from the digital elevation model/raster, the slope for the dipping angle and the aspect for the azimuth.



Sampled Orientations

	X	Y	Z	geometry	formation	dip	azimuth	polarity
0	498.10	747.09	639.61	POINT (498.099 747.088)	Test	27.52	149.04	1
1	402.68	1040.05	747.58	POINT (402.685 1040.048)	Test	11.25	178.61	1
2	54.08	385.02	397.73	POINT (54.080 385.024)	Test	25.00	249.13	1
3	152.61	679.71	543.43	POINT (152.607 679.705)	Test	47.91	252.85	1
4	533.62	1054.60	723.82	POINT (533.623 1054.601)	Test	23.76	125.68	1
5	882.15	921.22	572.99	POINT (882.147 921.224)	Test	35.66	233.80	1
6	839.61	710.91	490.91	POINT (839.614 710.914)	Test	5.73	240.20	1
7	442.41	76.81	429.44	POINT (442.409 76.807)	Test	20.54	138.88	1
8	288.13	737.22	647.40	POINT (288.130 737.217)	Test	22.81	218.87	1
9	38.71	692.95	448.73	POINT (38.707 692.954)	Test	34.77	262.12	1

6.9.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/08_sampling_interfaces_orientations_from_raster/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="08_sampling_interfaces_
↪orientations_from_raster.zip", dirpath=file_path)
```

```
Downloading file '08_sampling_interfaces_orientations_from_raster.zip' from 'https://
↪rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F08_sampling_interfaces_
↪orientations_from_raster.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_
↪started\tutorial\data\08_sampling_interfaces_orientations_from_raster'.
```

6.9.2 Loading raster data

[3]:
```
import rasterio

raster = rasterio.open(file_path + 'raster.tif')
```

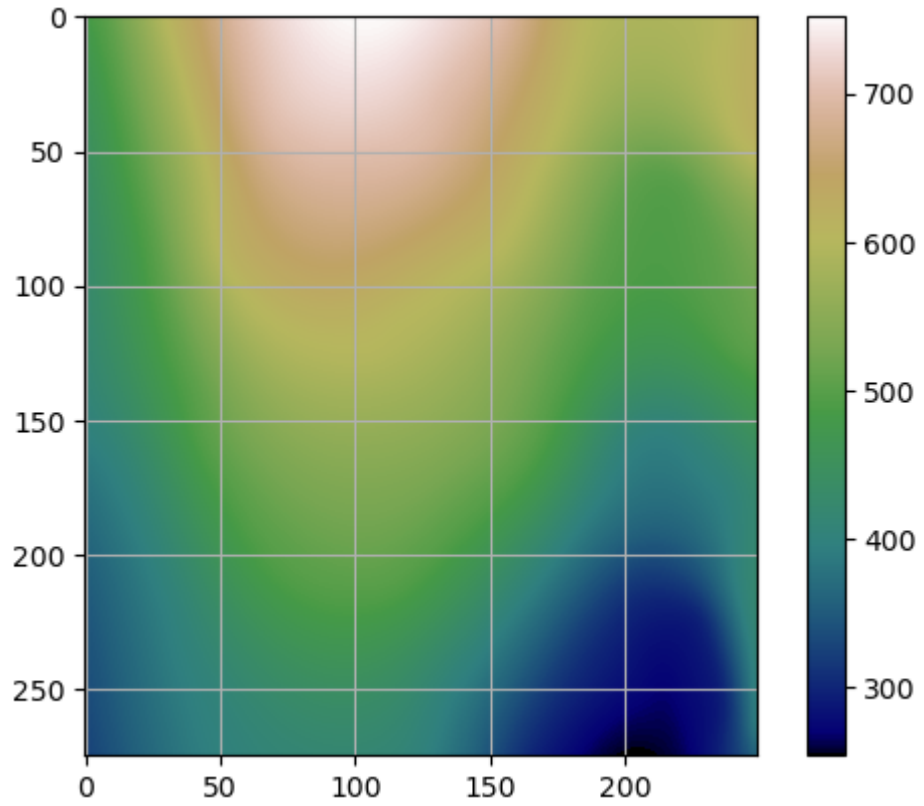
[4]:
```
raster.bounds
```

[4]:
```
BoundingBox(left=-4.440892098500626e-16, bottom=1.3642420526593924e-12, right=971.
↪002897605575, top=1068.1031873661339)
```

6.9.3 Plotting the raster

[5]:
```
import matplotlib.pyplot as plt

im = plt.imshow(raster.read(1), cmap='gist_earth')
plt.grid()
plt.colorbar(im)
```

[5]:
```
<matplotlib.colorbar.Colorbar at 0x2aaf1e504c0>
```



6.9.4 Sampling Interfaces from Raster

Sampling by lists

Sampling interfaces from a raster given lists of X and Y coordinates is done using `sample_interfaces(..)`.

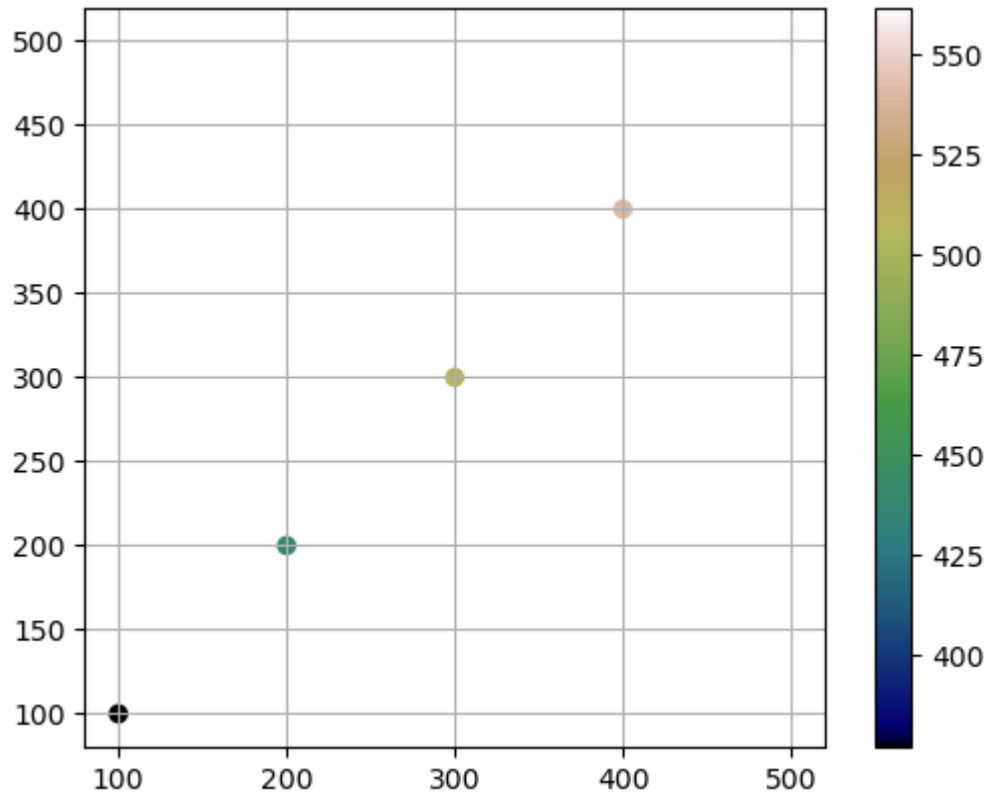
[6]:
```
point_x = [100, 200, 300, 400, 500]
point_y = [100, 200, 300, 400, 500]

gdf = gg.raster.sample_interfaces(raster=raster,
                                  point_x=point_x,
                                  point_y=point_y,
                                  formation='Test',
                                  crs='EPSG:4326')

gdf
```

[6]:

	X	Y	Z	geometry	formation
0	100.00	100.00	376.72	POINT (100.00000 100.00000)	Test
1	200.00	200.00	435.22	POINT (200.00000 200.00000)	Test
2	300.00	300.00	505.29	POINT (300.00000 300.00000)	Test
3	400.00	400.00	540.68	POINT (400.00000 400.00000)	Test
4	500.00	500.00	561.65	POINT (500.00000 500.00000)	Test

Plotting the data

[7]:
```
gdf.plot(column='Z', cmap='gist_earth', aspect='equal', legend=True)
plt.grid()
```



Sampling by arrays

Sampling interfaces from an array given lists of X and Y coordinates.

[8]:
```
point_x = [100, 200, 300, 400 ,500]
point_y = [100, 200, 300, 400 ,500]

gdf = gg.raster.sample_interfaces(raster=raster.read(1),
                                  point_x=point_x,
                                  point_y=point_y,
                                  extent=[0, 972, 0 ,1069],
                                  formation='Test',
                                  crs='EPSG:4326')

gdf
```

[8]:

	X	Y	Z	geometry	formation
0	100.00	100.00	378.80	POINT (100.00000 100.00000)	Test
1	200.00	200.00	435.22	POINT (200.00000 200.00000)	Test
2	300.00	300.00	505.29	POINT (300.00000 300.00000)	Test

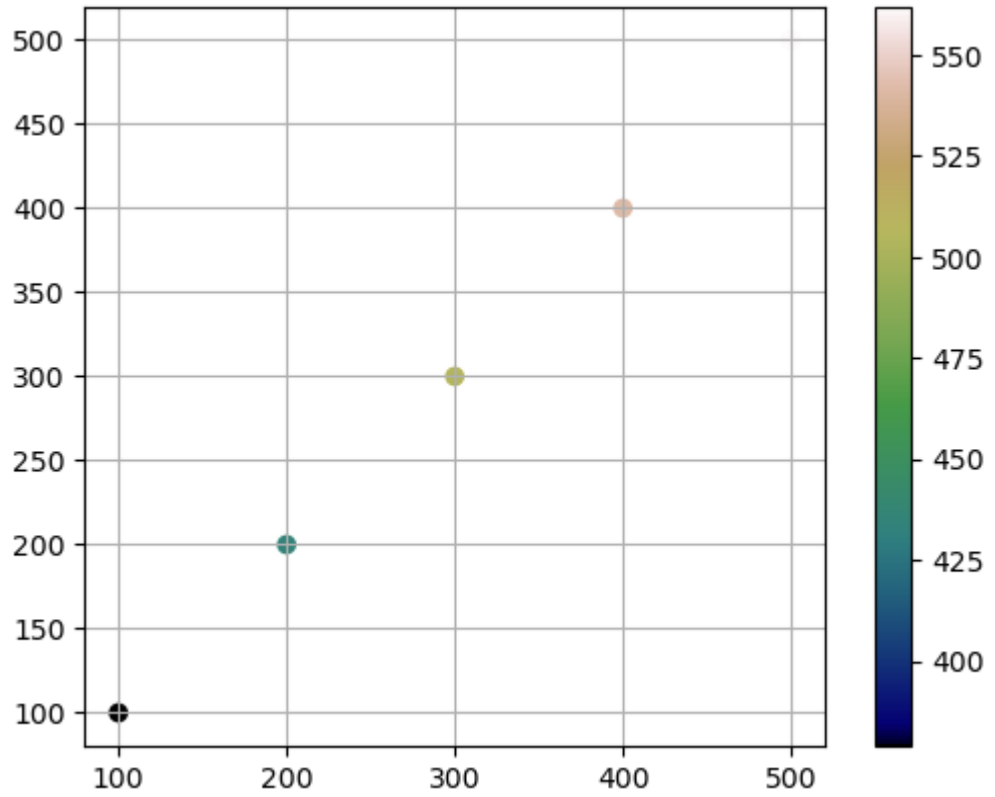
(continues on next page)

(continued from previous page)

```
3 400.00 400.00 541.74  POINT (400.00000 400.00000)      Test
4 500.00 500.00 562.02  POINT (500.00000 500.00000)      Test
```

Plotting the data

[9]:
```
gdf.plot(column='Z', cmap='gist_earth', aspect='equal', legend=True)
plt.grid()
```



Sampling Randomly from rasterio object

Sampling interfaces randomly from a raster.

[10]:
```
gdf = gg.raster.sample_interfaces(raster=raster,
                                  random_samples=10,
                                  formation='Test',
                                  crs='EPSG:4326')

gdf
```

[10]:
```
       X       Y      Z                       geometry formation
0 199.86  496.09 520.81   POINT (199.85519 496.09330)      Test
1 511.85  763.18 643.24   POINT (511.85172 763.17686)      Test
2 643.35  886.38 620.91   POINT (643.35122 886.37867)      Test
```

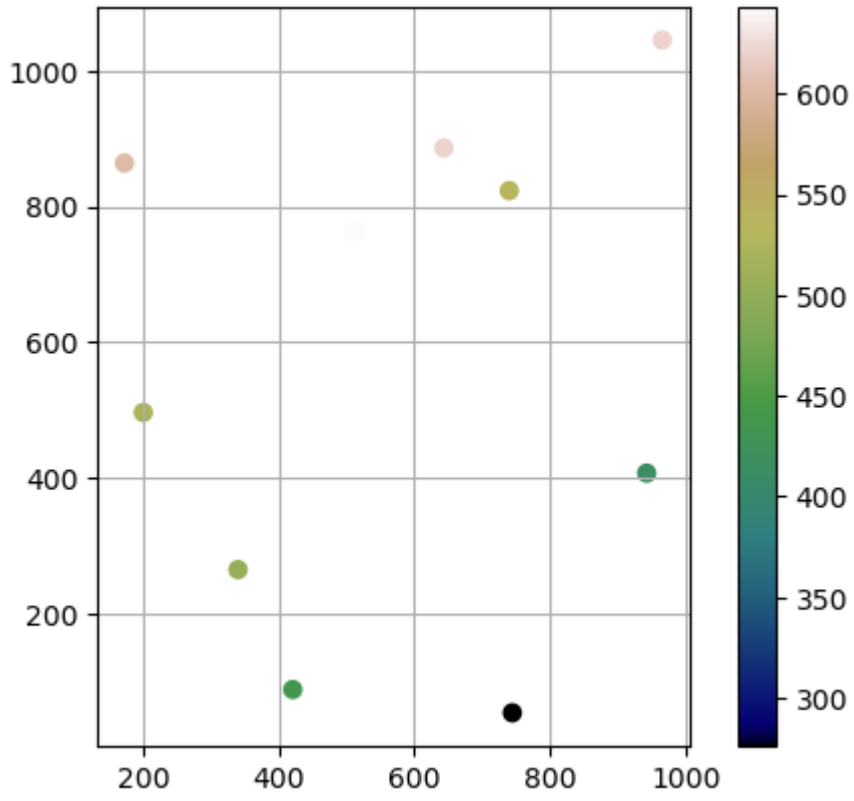
(continues on next page)

(continued from previous page)

```
3 965.65 1045.58 620.59  POINT (965.65475 1045.58279)      Test
4 172.10  864.05 602.61   POINT (172.10151 864.05005)      Test
5 744.41   52.90 276.00    POINT (744.40762 52.90476)      Test
6 739.94  823.36 533.44   POINT (739.93577 823.35867)      Test
7 942.70  406.58 414.82   POINT (942.69846 406.58069)      Test
8 420.50   87.25 436.52    POINT (420.50070 87.24755)      Test
9 339.68  264.21 503.23   POINT (339.68451 264.21229)      Test
```

Plotting the data

[11]:
```
gdf.plot(column='Z', cmap='gist_earth', aspect='equal', legend=True)
plt.grid()
```



Sampling Randomly from Array

Sampling interfaces randomly from an array.

[12]:
```
gdf = gg.raster.sample_interfaces(raster=raster.read(1),
                                  random_samples=10,
                                  extent=[0, 972, 0 ,1069],
                                  formation='Test',
                                  crs='EPSG:4326')
```

(continues on next page)

(continued from previous page)

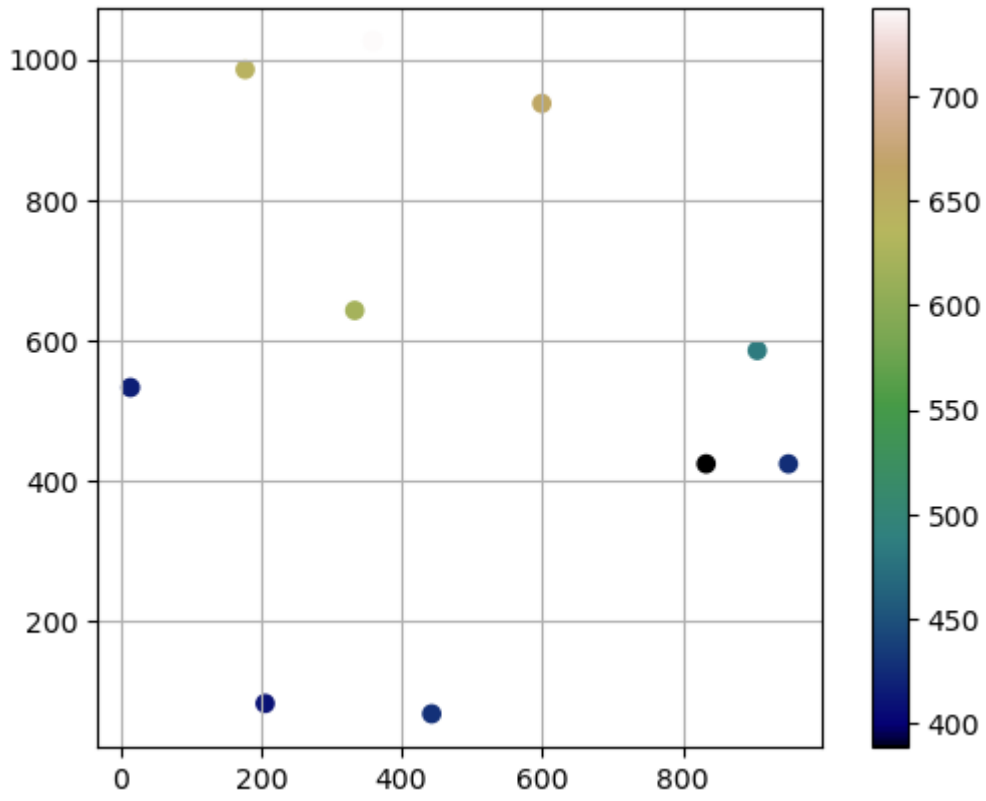
```
gdf
```

[12]:

	X	Y	Z	geometry	formation
0	950.27	424.37	426.03	POINT (950.26670 424.37299)	Test
1	205.91	83.67	410.35	POINT (205.90842 83.67224)	Test
2	833.19	424.42	388.64	POINT (833.18890 424.42058)	Test
3	442.79	68.90	427.47	POINT (442.79487 68.90223)	Test
4	333.34	642.61	620.60	POINT (333.33986 642.60911)	Test
5	905.91	585.65	484.48	POINT (905.90705 585.64586)	Test
6	359.39	1025.74	742.35	POINT (359.39233 1025.74463)	Test
7	599.42	936.93	660.12	POINT (599.41731 936.92645)	Test
8	14.28	532.99	417.31	POINT (14.28131 532.98620)	Test
9	177.50	985.25	643.90	POINT (177.49500 985.24662)	Test

Plotting the data

[13]:
```
gdf.plot(column='Z', cmap='gist_earth', aspect='equal', legend=True)
plt.grid()
```



6.9.5 Sampling Orientations from Raster

Sampling from lists

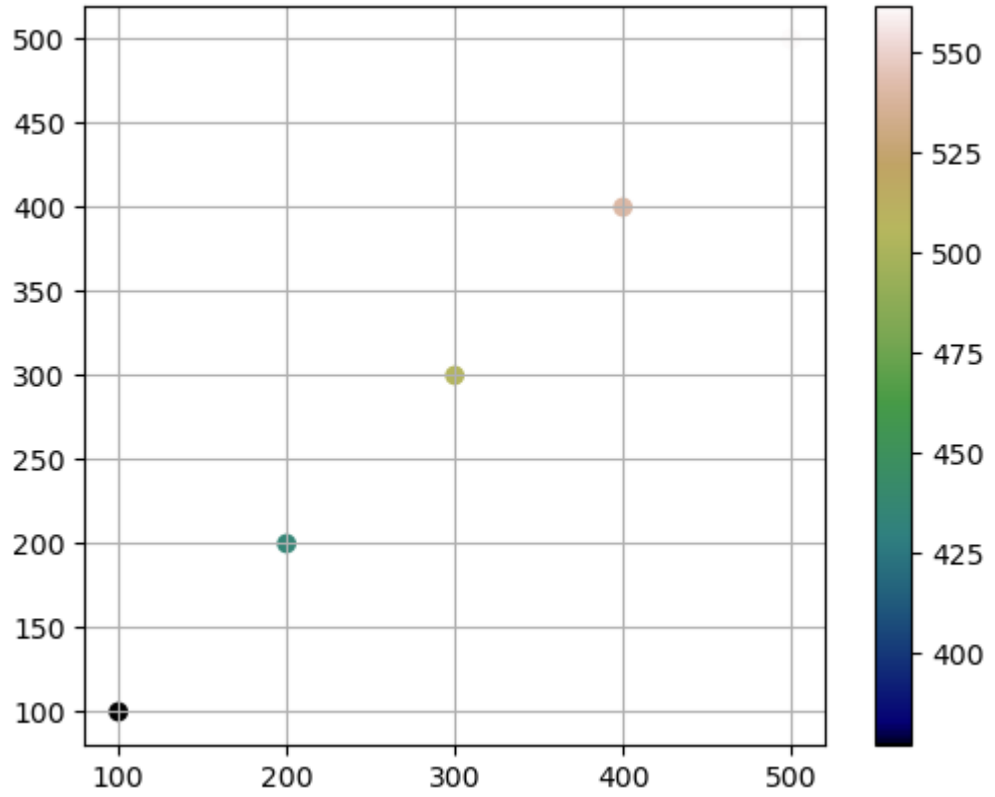
Sampling orientations from a raster given lists of X and Y coordinates.

[14]:
```
point_x = [100, 200, 300, 400 ,500]
point_y = [100, 200, 300, 400 ,500]

gdf = gg.raster.sample_orientations(raster=raster,
                                    point_x=point_x,
                                    point_y=point_y,
                                    formation='Test',
                                    crs='EPSG:4326')

gdf
```

[14]:
```
       X      Y      Z                    geometry formation   dip  azimuth  \
0 100.00 100.00 376.72  POINT (100.00000 100.00000)      Test 23.77   255.93
1 200.00 200.00 435.22  POINT (200.00000 200.00000)      Test 30.46   239.55
2 300.00 300.00 505.29  POINT (300.00000 300.00000)      Test 17.34   222.68
3 400.00 400.00 540.68  POINT (400.00000 400.00000)      Test 15.51   178.84
4 500.00 500.00 561.65  POINT (500.00000 500.00000)      Test 19.26   145.55

   polarity
0         1
1         1
2         1
3         1
4         1
```

Plotting the data

[15]:
```
gdf.plot(column='Z', cmap='gist_earth', aspect='equal', legend=True)
plt.grid()
```



Sampling from arrays

Sampling orientations from an array given lists of X and Y coordinates.

[16]:
```
point_x = [100, 200, 300, 400 ,500]
point_y = [100, 200, 300, 400 ,500]

gdf = gg.raster.sample_orientations(raster=raster.read(1),
                                    point_x=point_x,
                                    point_y=point_y,
                                    extent=[0, 972, 0 ,1069],
                                    formation='Test',
                                    crs='EPSG:4326')


gdf
```

[16]:
```
       X      Y      Z                    geometry formation   dip  azimuth  \
0 100.00 100.00 378.80  POINT (100.00000 100.00000)      Test 23.75   255.93
1 200.00 200.00 435.22  POINT (200.00000 200.00000)      Test 30.44   239.55
2 300.00 300.00 505.29  POINT (300.00000 300.00000)      Test 17.32   222.68
3 400.00 400.00 541.74  POINT (400.00000 400.00000)      Test 15.50   178.84
4 500.00 500.00 562.02  POINT (500.00000 500.00000)      Test 19.24   145.55

   polarity
0         1
```

(continues on next page)

(continued from previous page)

```
1          1
2          1
3          1
4          1
```

Sampling randomly from rasterio object

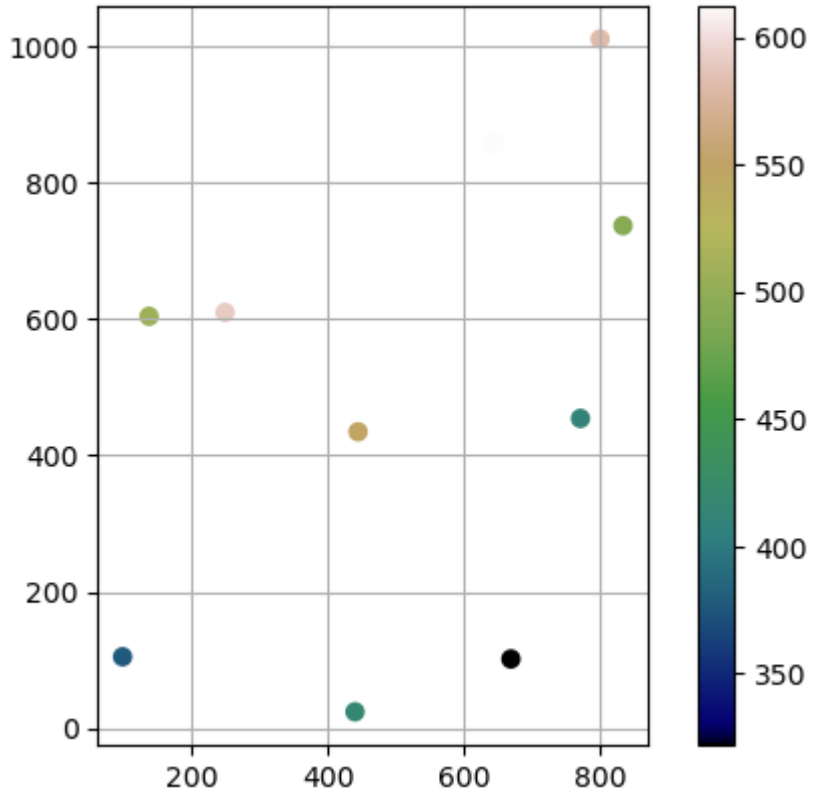
Sampling orientations randomly from a raster.

[17]:
```
gdf = gg.raster.sample_orientations(raster=raster,
                                    random_samples=10,
                                    formation='Test',
                                    crs='EPSG:4326')

gdf
```

[17]:
```
       X       Y      Z                      geometry formation   dip  \
0 770.72  454.44 411.17   POINT (770.72317 454.43623)      Test 35.50
1  98.85  104.73 377.13    POINT (98.85090 104.72719)      Test 24.00
2 439.88   23.92 417.44    POINT (439.88303 23.91500)      Test 15.43
3 444.39  434.85 548.88   POINT (444.39082 434.84986)      Test 16.92
4 642.46  857.77 612.54   POINT (642.45819 857.77092)      Test 33.90
5 668.68  101.45 321.65   POINT (668.68137 101.45438)      Test 34.24
6 833.16  737.23 491.34   POINT (833.16382 737.23242)      Test  3.94
7 137.75  604.27 506.55   POINT (137.75113 604.27271)      Test 39.04
8 249.23  610.05 590.12   POINT (249.22976 610.04522)      Test 30.48
9 799.49 1011.20 581.62  POINT (799.49348 1011.19910)      Test 11.49

   azimuth  polarity
0   126.05         1
1   256.35         1
2   138.91         1
3   165.67         1
4   115.23         1
5   122.96         1
6   242.61         1
7   252.36         1
8   224.83         1
9   170.42         1
```

Plotting the data

[18]:
```
gdf.plot(column='Z', cmap='gist_earth', aspect='equal', legend=True)
plt.grid()
```



Sample randomly from array

Sampling orientations randomly from an array.

[19]:
```
gdf = gg.raster.sample_orientations(raster=raster.read(1),
                                    random_samples=10,
                                    extent=[0, 972, 0 ,1069],
                                    formation='Test',
                                    crs='EPSG:4326')

gdf
```

[19]:
```
       X       Y      Z                       geometry formation   dip  \
0 433.83  627.61 606.19   POINT (433.82765 627.60890)      Test 19.29
1 732.25 1007.74 594.51  POINT (732.25432 1007.73581)      Test 21.30
2 849.26  539.88 444.47   POINT (849.25852 539.87992)      Test 29.99
3 506.54  297.89 499.94   POINT (506.54294 297.89069)      Test 22.06
4 482.80  959.46 715.72   POINT (482.79546 959.45977)      Test 21.73
5 914.62  832.23 543.36   POINT (914.61626 832.23124)      Test 45.25
6  16.88  805.99 447.03    POINT (16.88303 805.98712)      Test 37.63
7  97.01  896.66 534.58    POINT (97.00884 896.65803)      Test 48.06
8 528.77  325.33 500.61   POINT (528.77290 325.33217)      Test 20.33
9 635.98  435.47 500.45   POINT (635.98363 435.47230)      Test 29.20

   azimuth  polarity
```

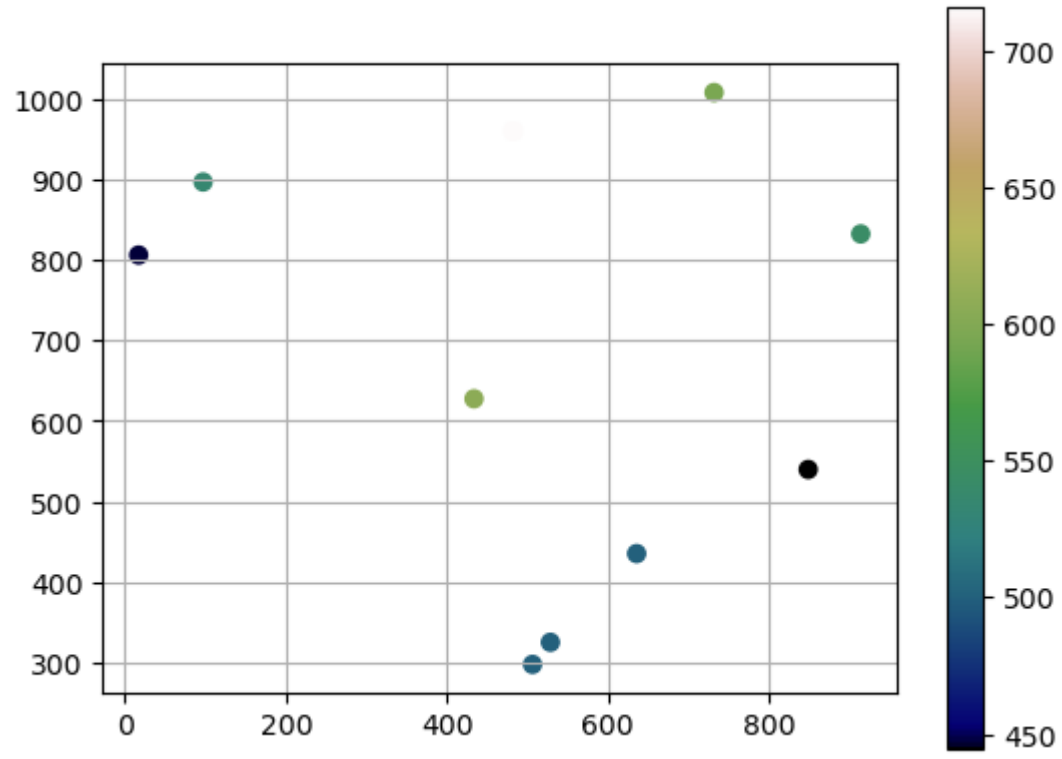
(continues on next page)

(continued from previous page)

```
0   144.83          1
1   109.13          1
2   196.23          1
3   134.76          1
4   129.97          1
5   231.65          1
6   260.69          1
7   257.25          1
8   138.42          1
9   118.66          1
```

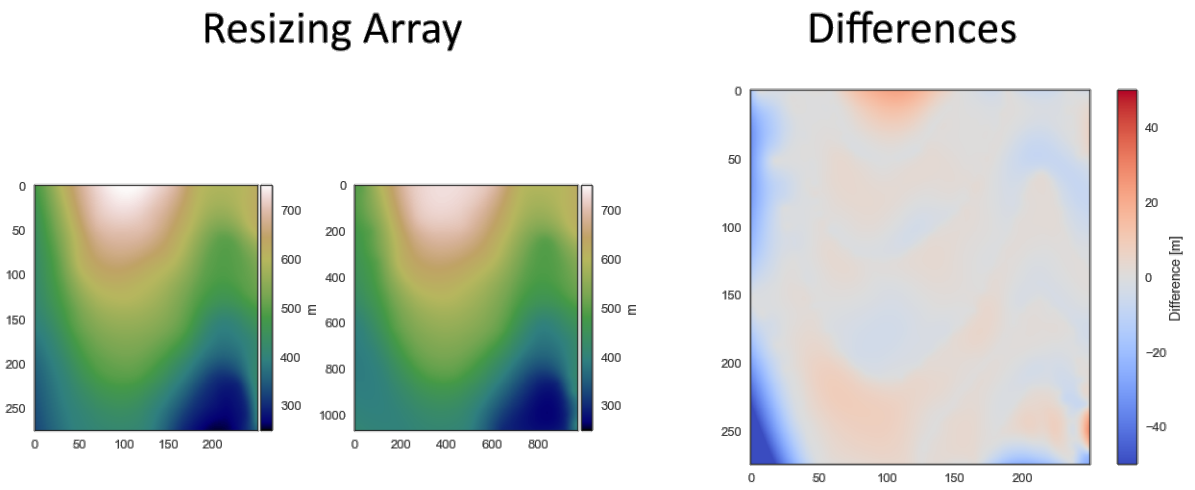
Plotting the data

[20]:
```
gdf.plot(column='Z', cmap='gist_earth', aspect='equal', legend=True)
plt.grid()
```



6.10 09 Raster Operations in GemGIS

Several smaller raster operations needed for the functionality of GemGIS are implemented. This includes calculating the difference of arrays and rasterio objects (`calculate_difference`).



6.10.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/09_raster_operations_gemgis/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="09_raster_operations_gemgis.zip
↪", dirpath=file_path)
```

```
Downloading file '09_raster_operations_gemgis.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F09_raster_operations_gemgis.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\tutorial\data\09_raster_operations_
↪gemgis'.
```

6.10.2 Loading the raster data

The rasters to perform the raster operations are being loaded with rasterio and NumPy.

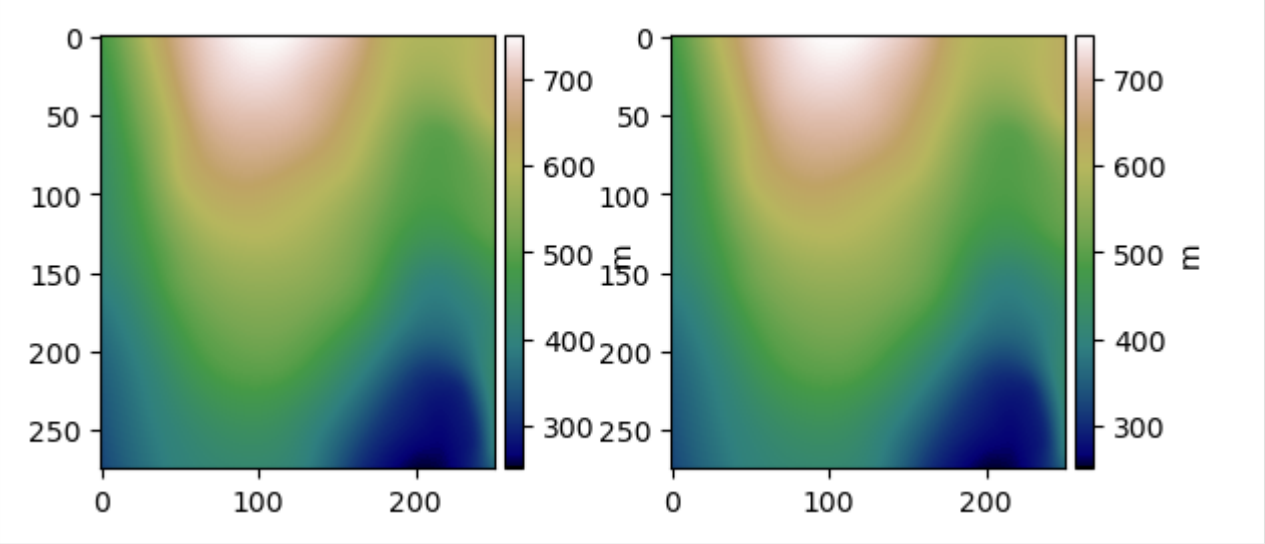
[3]:
```
import rasterio
import numpy as np

array = np.load(file_path + 'array.npy')

raster1 = rasterio.open(file_path + 'raster.tif')
raster2 = rasterio.open(file_path + 'raster.tif')
```

6.10.3 Calculating Differences

Plotting Raster Data

As the loaded rasters are identical, the same figures are plotted.

[4]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, (ax1, ax2) = plt.subplots(1,2)
plt.subplots_adjust(wspace=0.35)

im1 = ax1.imshow(raster1.read(1), cmap='gist_earth', vmin=250, vmax=750)
divider = make_axes_locatable(ax1)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im1, cax=cax)
cbar.set_label('m')


im2 = ax2.imshow(raster2.read(1), cmap='gist_earth', vmin=250, vmax=750)
divider = make_axes_locatable(ax2)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im2, cax=cax)
cbar.set_label('m')
```



Calculate difference between rasterio objects

The difference between two rasterio objects can easily be calculated using the function `calculate_difference`. The result will be an array. In this case, the result is zero as the arrays are identical.

[5]:
```
diff = gg.raster.calculate_difference(raster1=raster1,
                                      raster2=raster2)
diff
```

[5]:
```
array([[[0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.],
        [0., 0., 0., ..., 0., 0., 0.]]], dtype=float32)
```

Calculate difference between arrays of the same size

The difference between two arrays with the same size can easily be calculated using the function `calculate_difference`. The result will be an array. In this case, the result is not zero as 10 meters were subtracted from the second raster.

[6]:
```
diff = gg.raster.calculate_difference(raster1=raster1.read(1),
                                      raster2=raster2.read(1)-10)
diff
```

[6]:
```
array([[10., 10., 10., ..., 10., 10., 10.],
       [10., 10., 10., ..., 10., 10., 10.],
       [10., 10., 10., ..., 10., 10., 10.],
       ...,
       [10., 10., 10., ..., 10., 10., 10.],
       [10., 10., 10., ..., 10., 10., 10.],
       [10., 10., 10., ..., 10., 10., 10.]], dtype=float32)
```

Calculate difference of arrays of different size

The difference between two arrays that do not have the same size can easily be calculated using the function `calculate_difference`. The result will be an array. In this case, the result is not zero.

Printing the shapes of the arrays.

[7]:
```
print(array.shape)
```

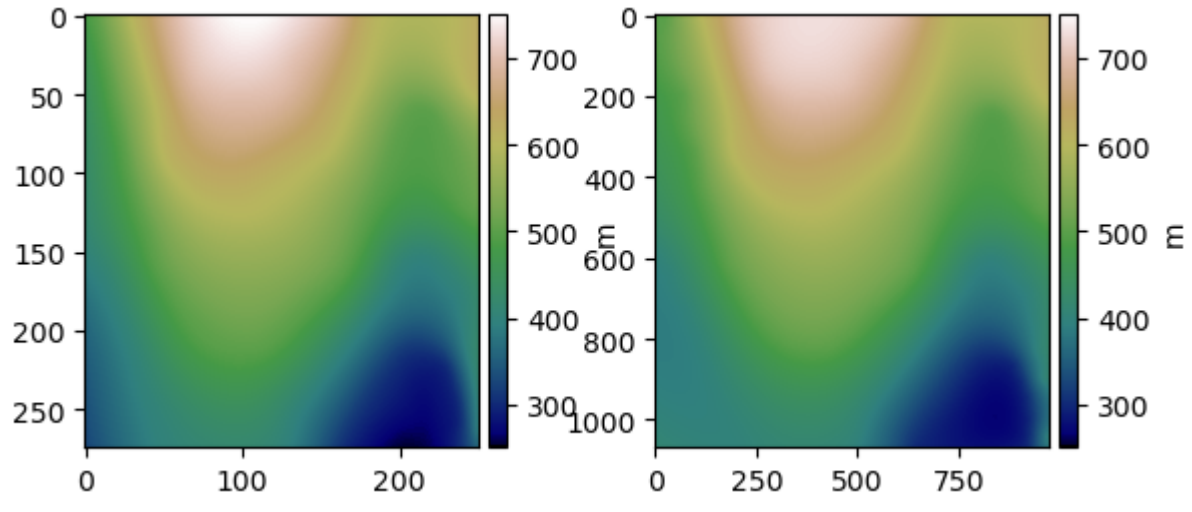
```
(1069, 972)
```

[8]:
```
print(raster1.shape)
```

```
(275, 250)
```

Plotting the arrays.

[9]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, (ax1, ax2) = plt.subplots(1,2)
plt.subplots_adjust(wspace=0.35)

im1 = ax1.imshow(raster1.read(1), cmap='gist_earth', vmin=250, vmax=750)
divider = make_axes_locatable(ax1)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im1, cax=cax)
cbar.set_label('m')


im2 = ax2.imshow(np.flipud(array), cmap='gist_earth', vmin=250, vmax=750)
divider = make_axes_locatable(ax2)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im2, cax=cax)
cbar.set_label('m')
```



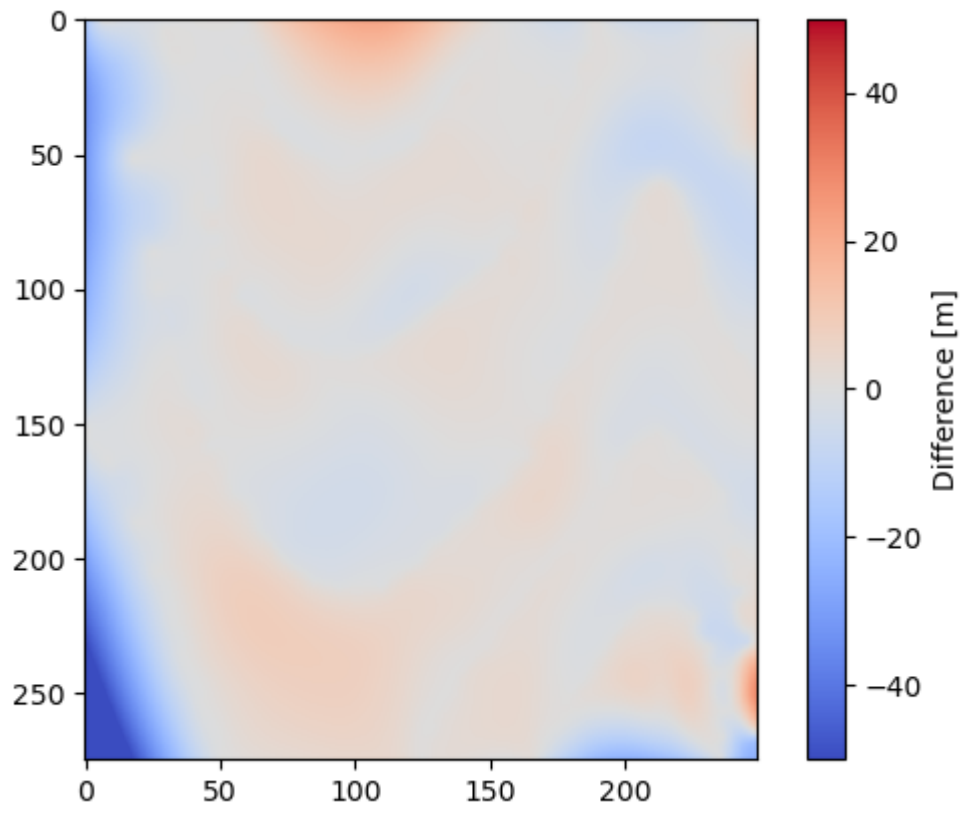
Calculating the difference. The array is turned so that the right values are substracted from the original raster.

[10]:
```
diff = gg.raster.calculate_difference(raster1=raster1.read(1),
                                      raster2=array,
                                      flip_array=True)

diff
```

[10]:
```
array([[-16.41678396, -13.61172709, -10.87035877, ...,  -3.19775272,
         -3.22781911,  -3.10479498],
       [-16.47500144, -13.72951014, -11.06531114, ...,  -2.5296072 ,
         -2.54858561,  -2.41534018],
       [-16.65319032, -13.99706557, -11.45171536, ...,  -1.86637421,
         -1.8699199 ,  -1.72295418],
       ...,
       [-84.66600704, -82.48561266, -80.28644375, ..., -22.60844144,
        -22.88386594, -22.15305095],
       [-85.70802569, -83.51980015, -81.31276338, ..., -23.11606161,
        -23.39106379, -22.70976508],
       [-86.71828282, -84.52284361, -82.30832183, ..., -23.66986812,
        -23.95510436, -23.32702576]])
```

Plotting the difference. The difference is caused by a different interpolation algorithm that was used to create the array.

[11]:
```
im = plt.imshow(diff, cmap='coolwarm', vmin=-50, vmax=50)
cbar = plt.colorbar(im)
cbar.set_label('Difference [m]')
```



6.10.4 Resizing a raster by extent

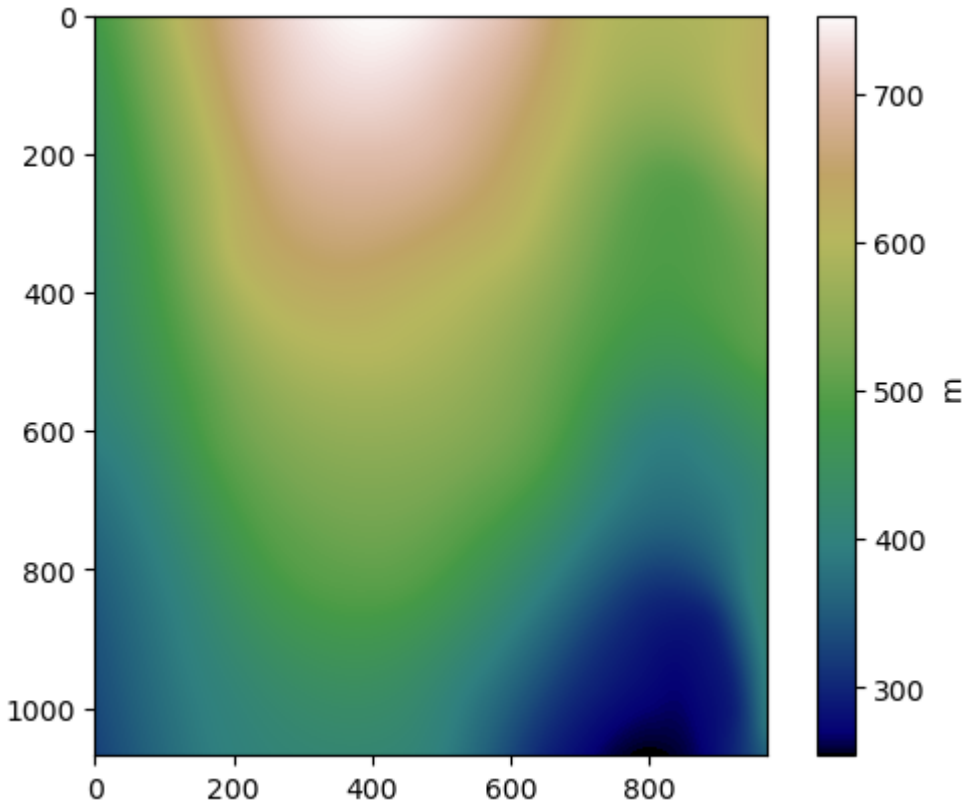
We can easily resize an array by providing a width and height for the returning array.

[12]:
```
array_resized = gg.raster.resize_raster(raster=raster1,
                                        width=972,
                                        height=1069)
array_resized.shape
```

[12]:
```
(1069, 972)
```

Plotting the new array.

[13]:
```
im = plt.imshow(array_resized, cmap='gist_earth')
cbar = plt.colorbar(im)
cbar.set_label('m')
```



6.10.5 Resize raster by array

A raster can also be resized to fit the size of another array.

Printing the shapes of the arrays.

[14]:
```
print(raster1.shape)
```

```
(275, 250)
```

[15]:
```
print(array.shape)
```

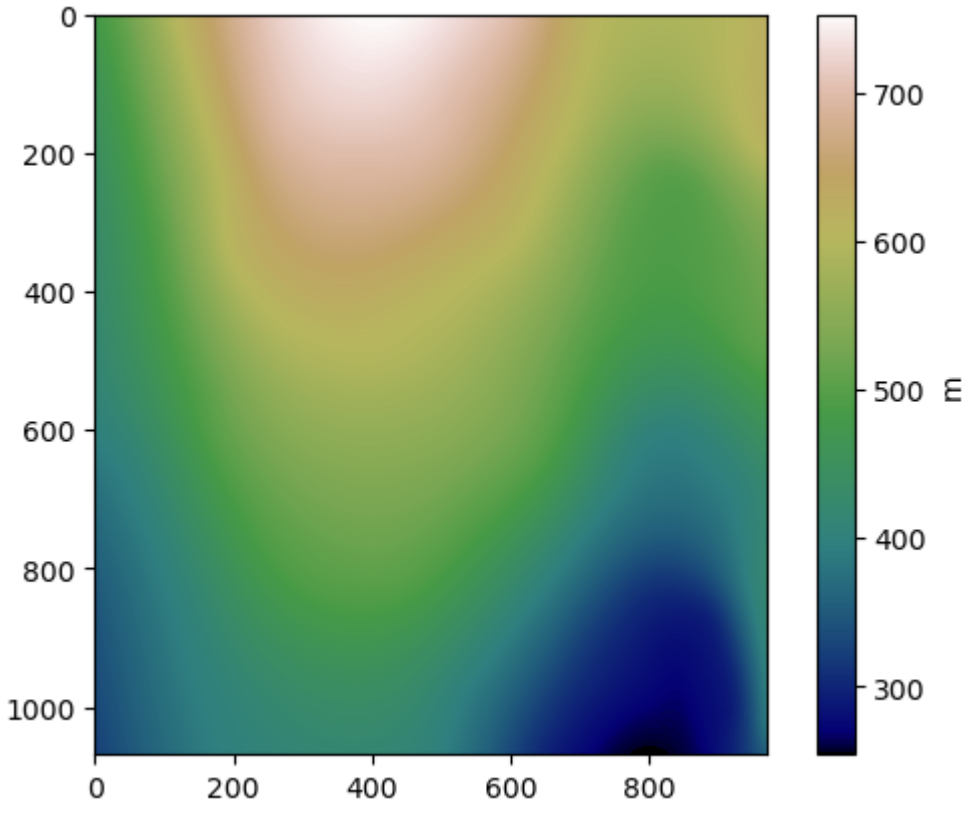
```
(1069, 972)
```

Resizing the raster to the size of the array.

[16]:
```
array_resized = gg.raster.resize_by_array(raster=raster1,
                                          array=array)

print(array_resized.shape)
```

```
(1069, 972)
```

[17]:
```
im = plt.imshow(array_resized, cmap='gist_earth')
cbar = plt.colorbar(im)
cbar.set_label('m')
```



6.10.6 Saving array as tiff

Arrays can easily be saved as rasters using `save_as_tiff`.

[18]:
```
gg.raster.save_as_tiff(raster=array,
                       path= file_path + 'output_raster.tif',
                       extent=[0,972,0,1069],
                       crs='EPSG:4326',
                       overwrite_file=True)
```

```
Raster successfully saved
```

6.10.7 Merging rasters

Several different rasterio objects/raster files can be merged using the `merge_tiles(..)` function. Before that, a list of file paths and and a list of loaded rasterio objects needs to be created first.

Rasters downloaded from https://www.tim-online.nrw.de/tim-online2/

Creating List of Filepaths

A list of filepaths with all raster files can be created using `create_filepaths(..)`.

[19]:
```
paths = gg.raster.create_filepaths(dirpath=file_path, search_criteria = 'tile*')
paths
```

[19]:
```
['C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_292000_294000_5626000_5628000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_292000_294000_5628000_5630000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_292000_294000_5630000_5632000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_294000_296000_5626000_5628000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_294000_296000_5628000_5630000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_294000_296000_5630000_5632000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_296000_298000_5626000_5628000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_296000_298000_5628000_5630000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\09_
↪raster_operations_gemgis\\tile_296000_298000_5630000_5632000.tif']
```

Creating List of Rasterio Objects

[20]:
```
raster_objects = gg.raster.create_src_list(dirpath=file_path, search_criteria = 'tile*')
raster_objects
```

[20]:
```
[<open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_292000_294000_5626000_5628000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_292000_294000_5628000_5630000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_292000_294000_5630000_5632000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_294000_296000_5626000_5628000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_294000_296000_5628000_5630000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_294000_296000_5630000_5632000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_296000_298000_5626000_5628000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_296000_298000_5628000_5630000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_296000_298000_5630000_5632000.tif' mode=
↪'r'>]
```

[21]:
```
raster_objects = gg.raster.create_src_list(filepaths=paths)
raster_objects
```

[21]:
```
[<open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_292000_294000_5626000_5628000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_292000_294000_5628000_5630000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_292000_294000_5630000_5632000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_294000_296000_5626000_5628000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_294000_296000_5628000_5630000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_294000_296000_5630000_5632000.tif' mode=
↪'r'>,
```

(continues on next page)

(continued from previous page)

```
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_296000_298000_5626000_5628000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_296000_298000_5628000_5630000.tif' mode=
↪'r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/09_raster_operations_gemgis/tile_296000_298000_5630000_5632000.tif' mode=
↪'r'>]
```

Merging Rasters

The loaded list of rasters can now be merged using `merge_tiles(..)`. The result will the array that is created and the transform associated with the array.

[22]:
```
mosaic, transform = gg.raster.merge_tiles(src_files=raster_objects)
```

[23]:
```
type(mosaic)
```

[23]:
```
numpy.ndarray
```

[24]:
```
type(transform)
```

[24]:
```
affine.Affine
```

[25]:
```
mosaic
```

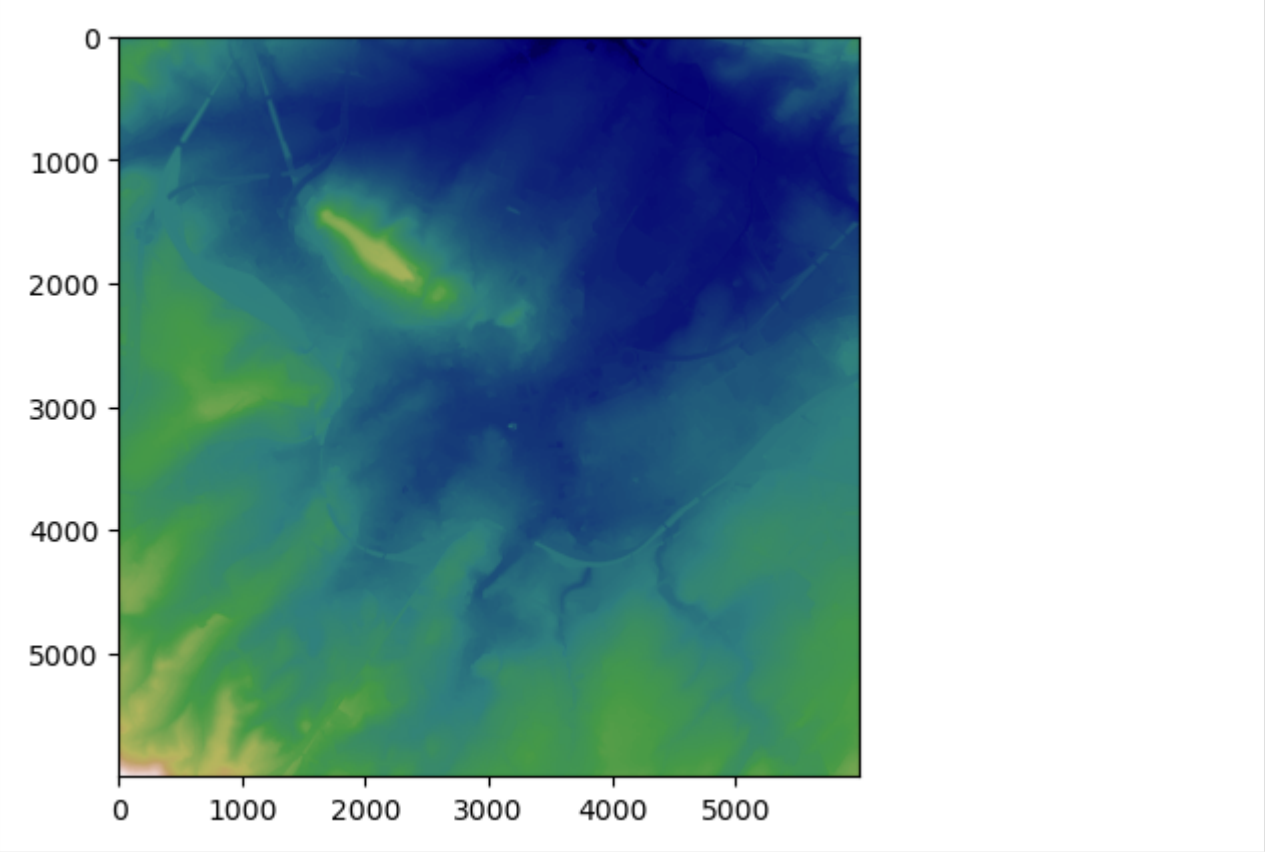
[25]:
```
array([[200.72, 200.73, 200.72, ..., 204.42, 204.45, 204.45],
       [200.74, 200.74, 200.75, ..., 204.43, 204.44, 204.48],
       [200.76, 200.76, 200.76, ..., 204.42, 204.48, 204.5 ],
       ...,
       [329.15, 328.86, 328.74, ..., 242.45, 242.38, 242.28],
       [329.29, 329.06, 328.87, ..., 242.45, 242.39, 242.31],
       [329.47, 329.3 , 329.09, ..., 242.42, 242.37, 242.32]],
      dtype=float32)
```

[26]:
```
transform
```

[26]:
```
Affine(1.0, 0.0, 292000.0,
       0.0, -1.0, 5632000.0)
```

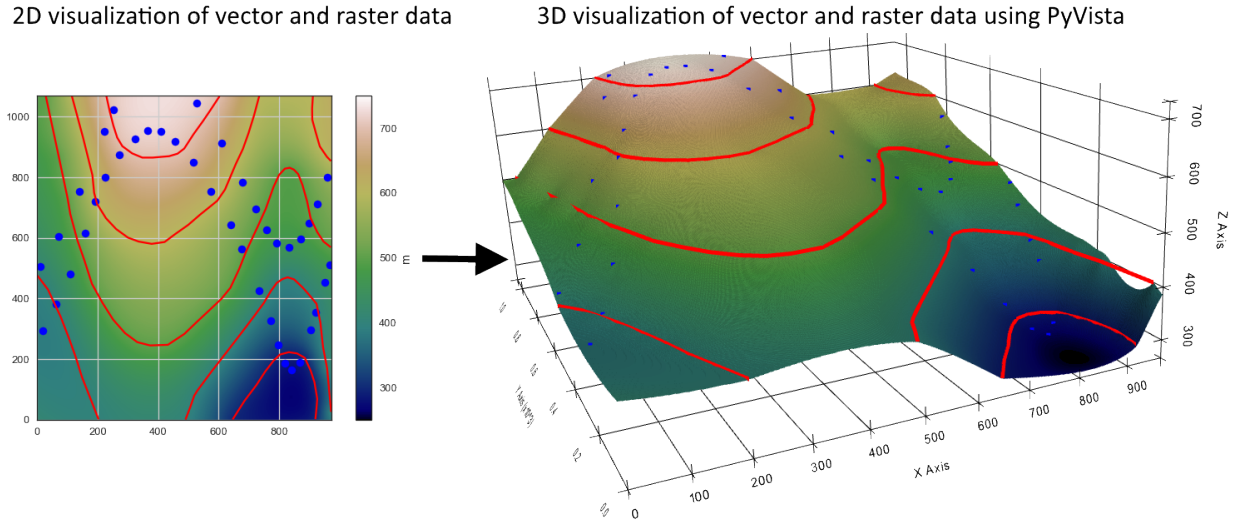
[27]:
```
plt.imshow(mosaic, cmap='gist_earth')
```

[27]:
```
<matplotlib.image.AxesImage at 0x29685b71cf0>
```



6.11 10 Visualizing Spatial Data with PyVista

Spatial Data can be displayed using the PyVista package. This includes point data, line data and rasters. Data will usually be returned as PolyData datasets or Grids so that the user has the full flexibility of plotting the data with PyVista.



6.11.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/10_visualizing_data_with_pyvista/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="10_visualizing_data_with_
↪pyvista.zip", dirpath=file_path)
```

6.11.2 Visualizing Contour Lines with PyVista

Loading Data

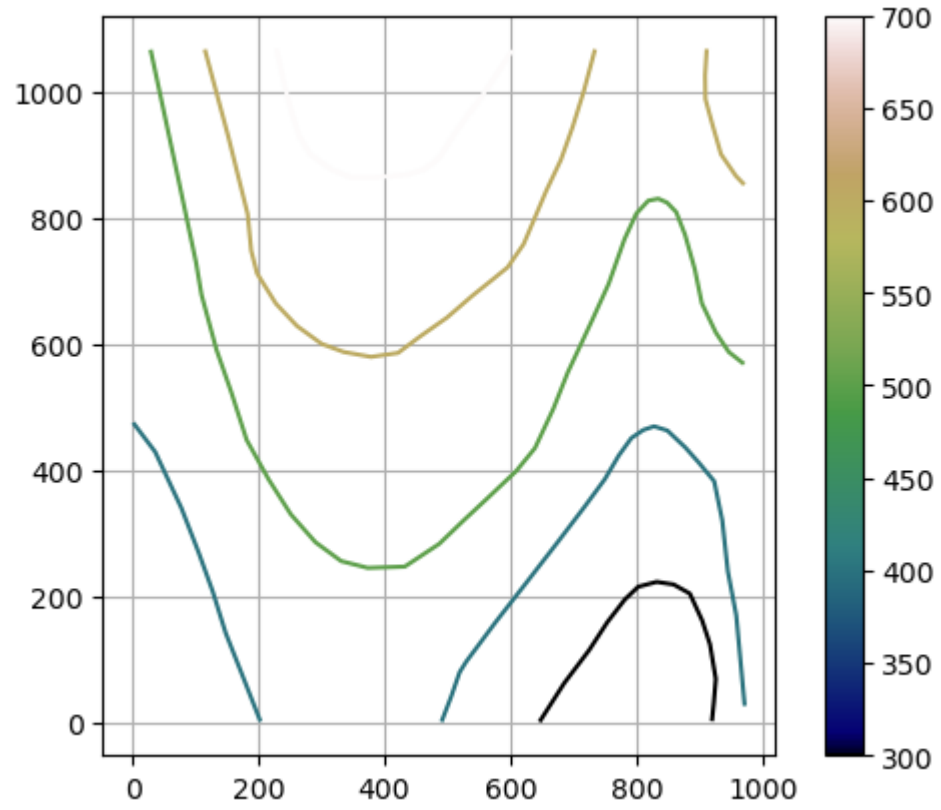
The contour lines are loaded as Shapely LineStrings within a GeoDataFrame.

[3]:
```
import pyvista as pv
import geopandas as gpd

contours = gpd.read_file(file_path + 'topo.shp')
contours.head()
```

[3]:
```
     id    Z                                           geometry
0  None  400  LINESTRING (0.74088 475.44101, 35.62873 429.24...
1  None  300  LINESTRING (645.96500 0.52496, 685.14093 61.86...
2  None  400  LINESTRING (490.29223 0.52496, 505.75641 40.73...
3  None  600  LINESTRING (911.43347 1068.58451, 908.85610 10...
4  None  700  LINESTRING (228.43207 1068.58451, 239.77247 10...
```

Plotting Data

[4]:
```
import matplotlib.pyplot as plt

contours.plot(aspect='equal', column='Z', cmap='gist_earth', legend=True)
plt.grid()
```



Extracting the vertices of the contour lines for the plotting with PyVista

A PolyData dataset of the contour lines can be created using `create_lines_3d(..)`.

[5]:
```
lines = gg.visualization.create_lines_3d_polydata(gdf=contours)
lines
```

[5]:
```
PolyData (0x1e86aaaf100)
  N Cells:  7
  N Points: 121
  N Strips: 0
  X Bounds: 7.409e-01, 9.717e+02
  Y Bounds: 5.250e-01, 1.069e+03
  Z Bounds: 3.000e+02, 7.000e+02
  N Arrays: 0
```

[6]:
```
type(lines)
```

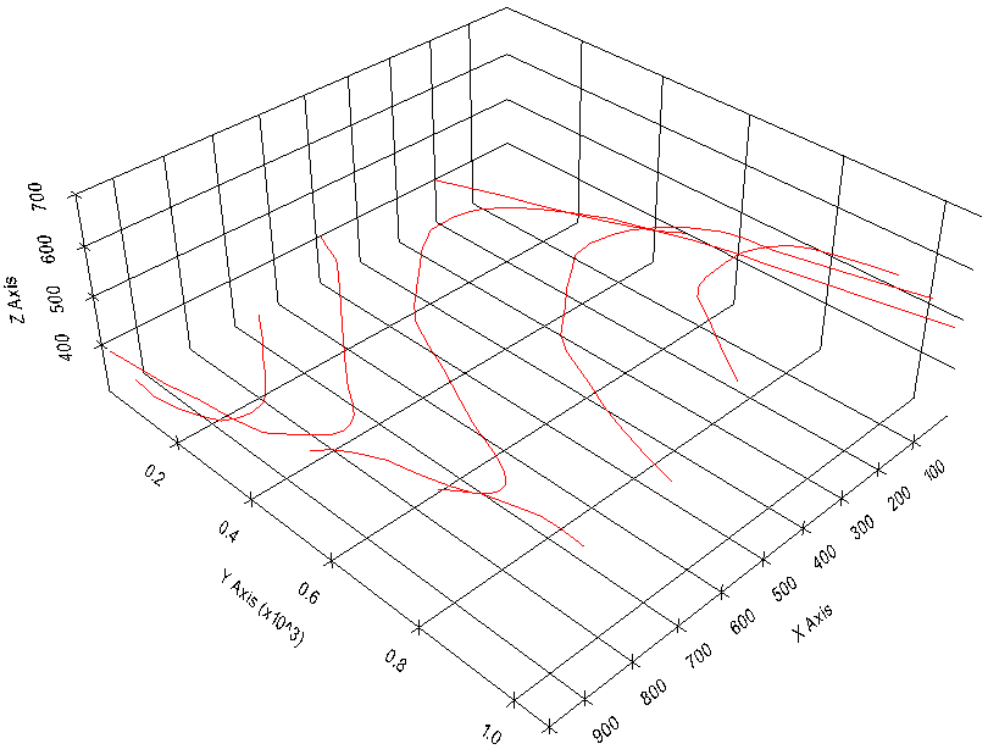
[6]:
```
pyvista.core.pointset.PolyData
```

Plotting the Lines with PyVista

The result can then be plotted with PyVista by creating a new Plotter and adding the lines as mesh.

[7]:
```
p = pv.Plotter()

p.add_mesh(mesh=lines, color='red')

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis_test\lib\site-packages\pyvista\jupyter\notebook.
↪py:60: UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.11.3 Visualizing Points with PyVista

Load Data

The points are loaded as Shapely Points within a GeoDataFrame.

[8]:
```
import pyvista as pv
import geopandas as gpd
import gemgis as gg

points = gpd.read_file(file_path + 'interfaces_points.shp')
points.head()
```

[8]:
```
     id formation                     geometry
0  None       Ton   POINT (19.15013 293.31349)
1  None       Ton   POINT (61.93437 381.45933)
2  None       Ton  POINT (109.35786 480.94557)
3  None       Ton  POINT (157.81230 615.99943)
4  None       Ton  POINT (191.31803 719.09398)
```

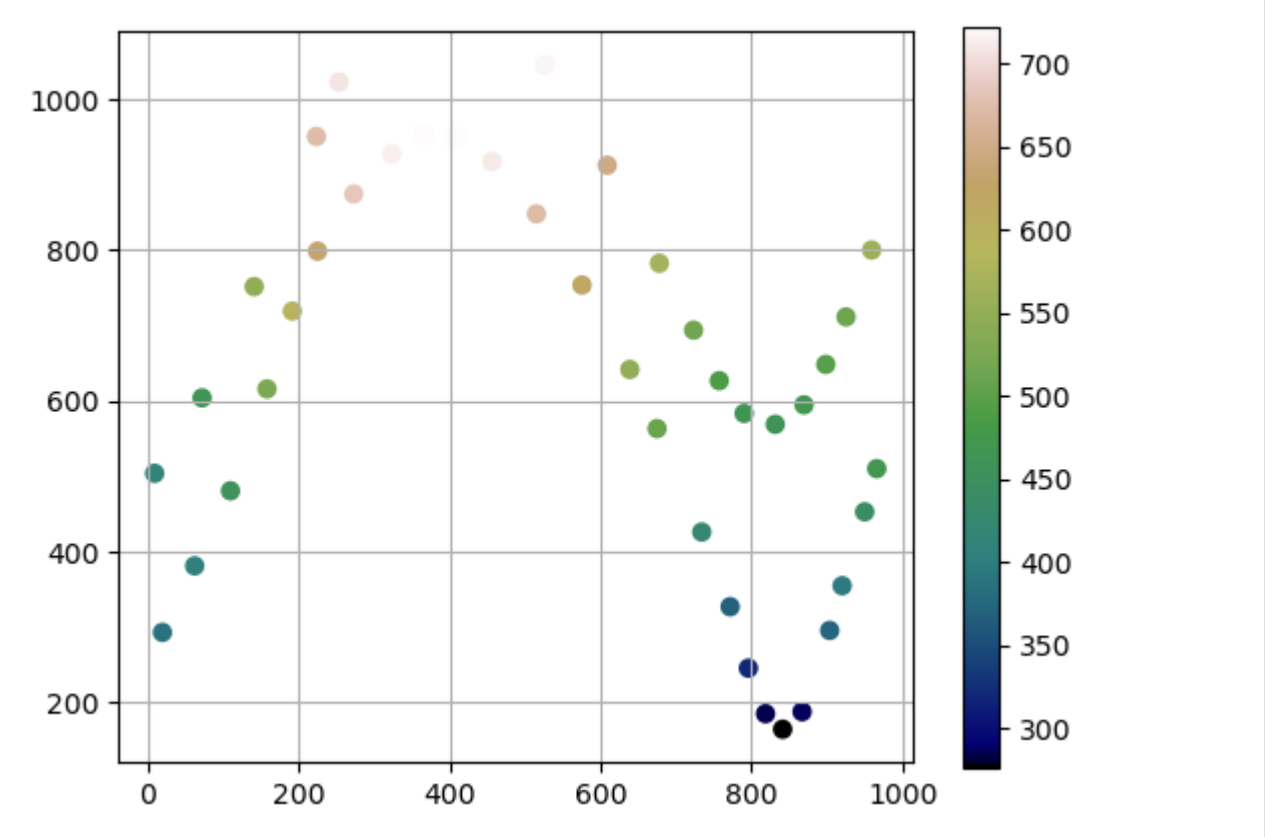
Loading DEM and extract Z values

In order to plot the points at the correct Z-position, the values have to be extracted from a DEM.

[9]:
```
import rasterio

dem = rasterio.open(file_path + 'topo.tif')

points_xyz = gg.vector.extract_xyz(gdf=points,
                                   dem=dem)
```

Plotting Data

[10]:
```
import matplotlib.pyplot as plt

points_xyz.plot(aspect='equal', column='Z', cmap='gist_earth', legend=True)
plt.grid()
```



Extracting the vertices of the contour lines for the plotting with PyVista

A PolyData dataset containing the point information can be created using `create_points_3d(..)`.

[11]:
```
points_mesh = gg.visualization.create_points_3d(gdf=points_xyz)
points_mesh
```

[11]:
```
PolyData (0x1e8698e3e80)
  N Cells:  41
  N Points: 41
  N Strips: 0
  X Bounds: 8.841e+00, 9.661e+02
  Y Bounds: 1.650e+02, 1.045e+03
  Z Bounds: 2.769e+02, 7.220e+02
  N Arrays: 0
```

[12]:
```
type(points_mesh)
```

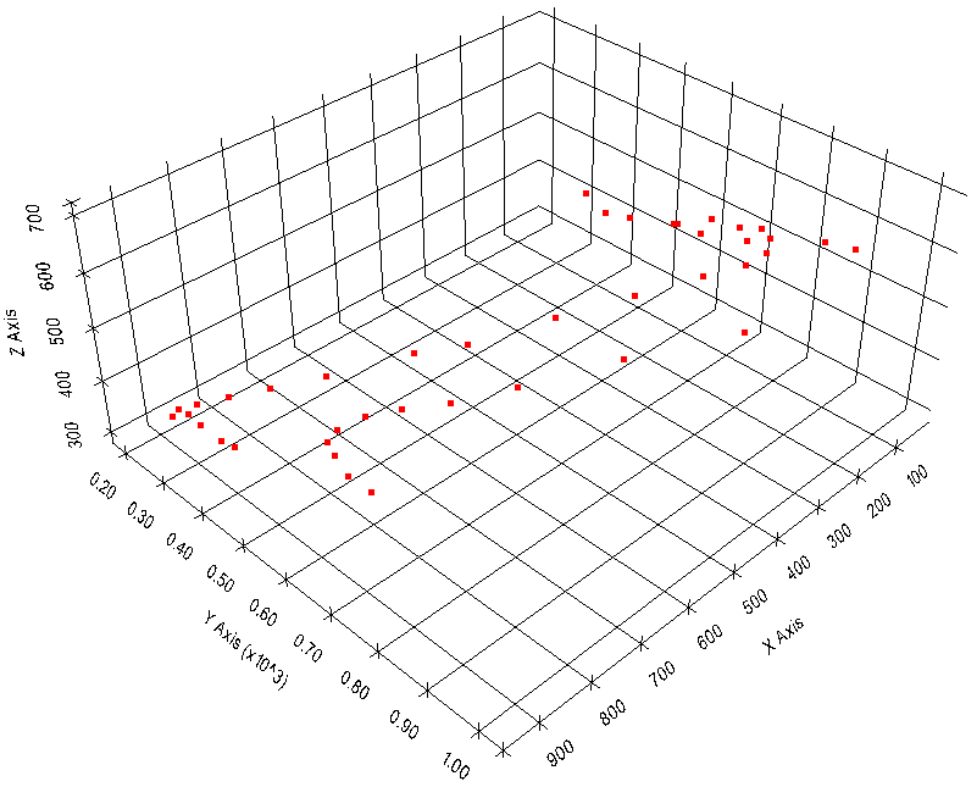
[12]:
```
pyvista.core.pointset.PolyData
```

Plotting the Points with PyVista

The result can then be plotted with PyVista by creating a new Plotter and adding the points as mesh.

[13]:
```
p = pv.Plotter()

p.add_mesh(mesh=points_mesh, color='red')

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis_test\lib\site-packages\pyvista\jupyter\notebook.
↪py:60: UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.11.4 Visualizing the DEM with PyVista

Loading Data

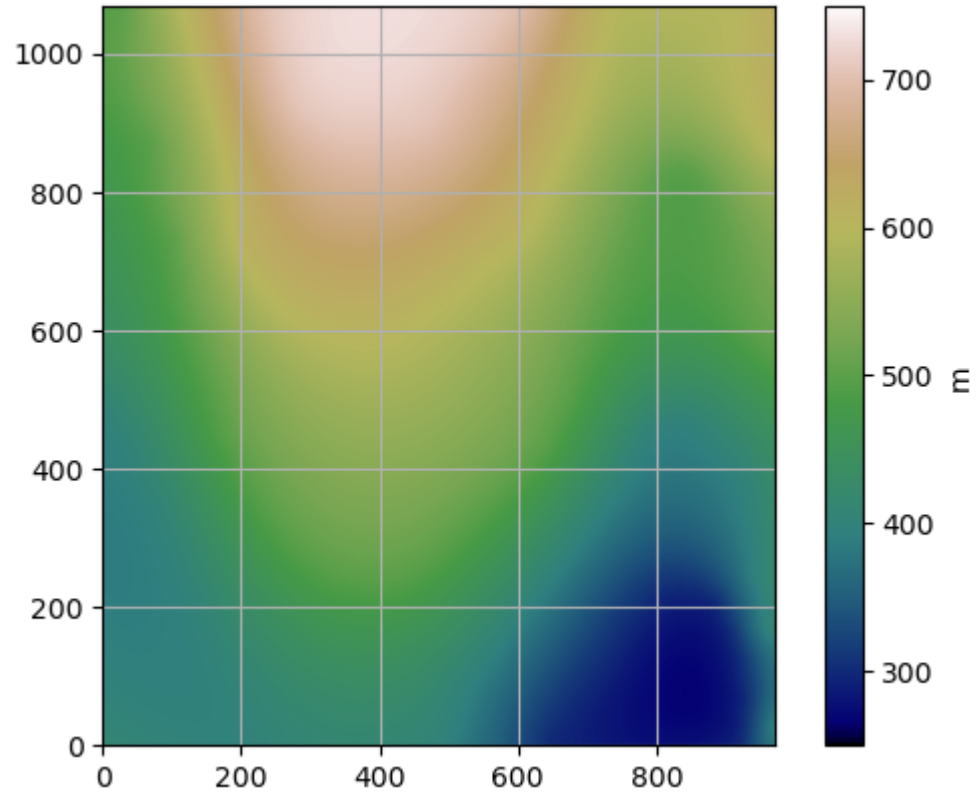
The DEM is loaded as Rasterio object using rasterio.

[14]:
```
import rasterio
import gemgis as gg
import pyvista as pv
import numpy as np

dem = rasterio.open(file_path + 'topo.tif')

dem.read(1)
```

[14]:
```
array([[499.90110024, 499.86421238, 499.82858152, ..., 625.37307284,
        625.78164892, 626.18920124],
       [499.53566482, 499.49887905, 499.4633306 , ..., 625.17315916,
        625.58165735, 625.98912699],
       [499.18752484, 499.15158818, 499.11692808, ..., 624.97739453,
        625.38574125, 625.79305697],
       ...,
       [411.5023835 , 411.37335931, 411.24503355, ..., 384.8252337 ,
        386.21293421, 387.56684012],
       [411.66101945, 411.5316941 , 411.40306465, ..., 384.4299191 ,
        385.80964238, 387.15718098],
       [411.82014581, 411.69052091, 411.56158939, ..., 384.04962954,
        385.42140506, 386.76248969]])
```

Plotting the Data

[15]:
```
import matplotlib.pyplot as plt

im =plt.imshow(dem.read(1), cmap='gist_earth', vmin=250, vmax=750, extent=[0,972,0,1069])
cbar = plt.colorbar(im)
cbar.set_label('m')
plt.grid()
```



Converting the Rasterio object or NumPy Array into a Structured Grid

A StructuredGrid containing the DEM information can be created using `create_dem_3d(..)`.

[16]:
```
grid = gg.visualization.create_dem_3d(dem=np.flipud(dem.read(1)), extent=[0,972,0,1069])

grid
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis_test\lib\site-packages\pyvista\utilities\helpers.
↪py:507: UserWarning: Points is not a float type. This can cause issues when␣
↪transforming or applying filters. Casting to ``np.float32``. Disable this by passing␣
↪``force_float=False``.
  warnings.warn(
```

[16]:
```
StructuredGrid (0x1e871337640)
  N Cells:  1037028
  N Points: 1039068
  X Bounds: 0.000e+00, 9.710e+02
  Y Bounds: 0.000e+00, 1.068e+03
  Z Bounds: 2.650e+02, 7.300e+02
  Dimensions:       1069, 972, 1
  N Arrays: 1
```

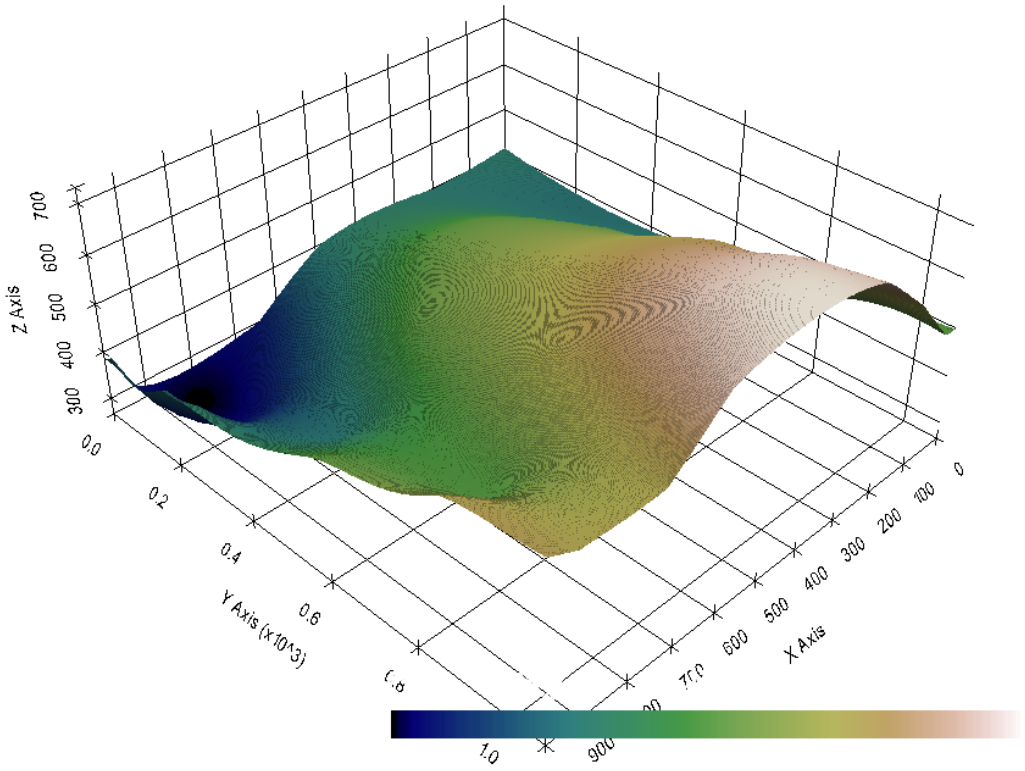
[17]:
```
type(grid)
```

[17]:
```
pyvista.core.pointset.StructuredGrid
```

Plotting the Structured Grid with PyVista

The elevation information of the DEM, also called `scalars` are stored in the NumPy array `grid["Elevation"]`. The grid can be plotted by creating a new PyVista Plotter and adding the grid as mesh to the plotter.

[18]:
```
p = pv.Plotter()

p.add_mesh(mesh=grid, scalars=grid["Elevation"], cmap='gist_earth')

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.11.5 Combining everything

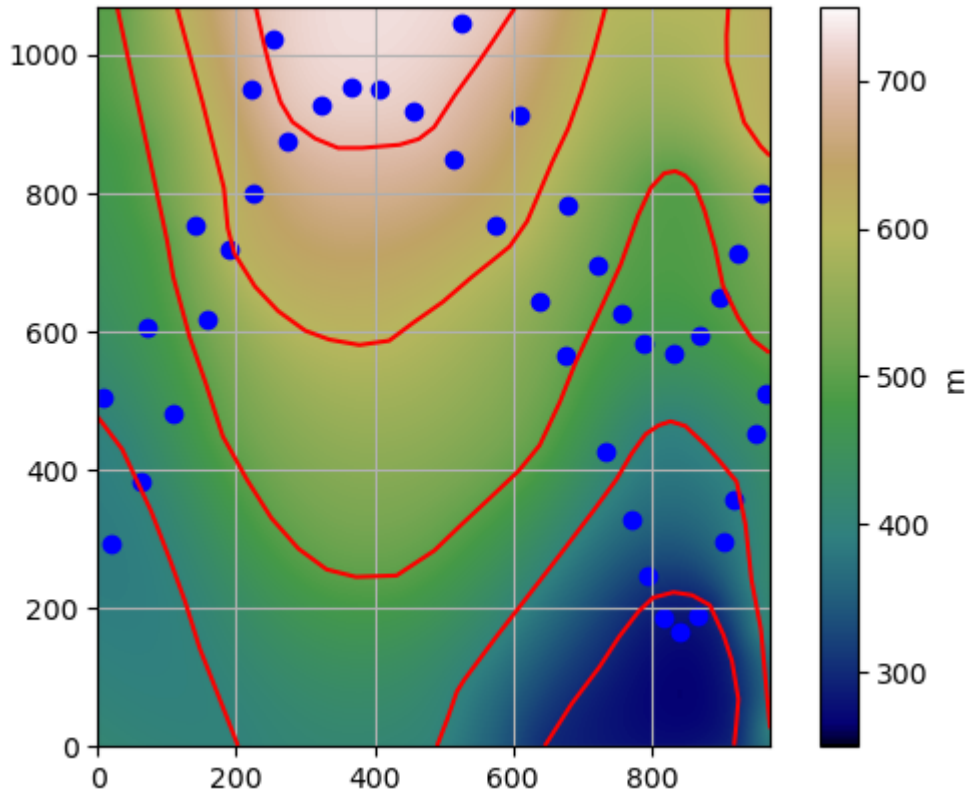
All datasets can also be plotted in the same plot.

[19]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1,1)
points_xyz.plot(ax=ax, aspect='equal', color='blue')
contours.plot(ax=ax,aspect='equal', color='red')
```

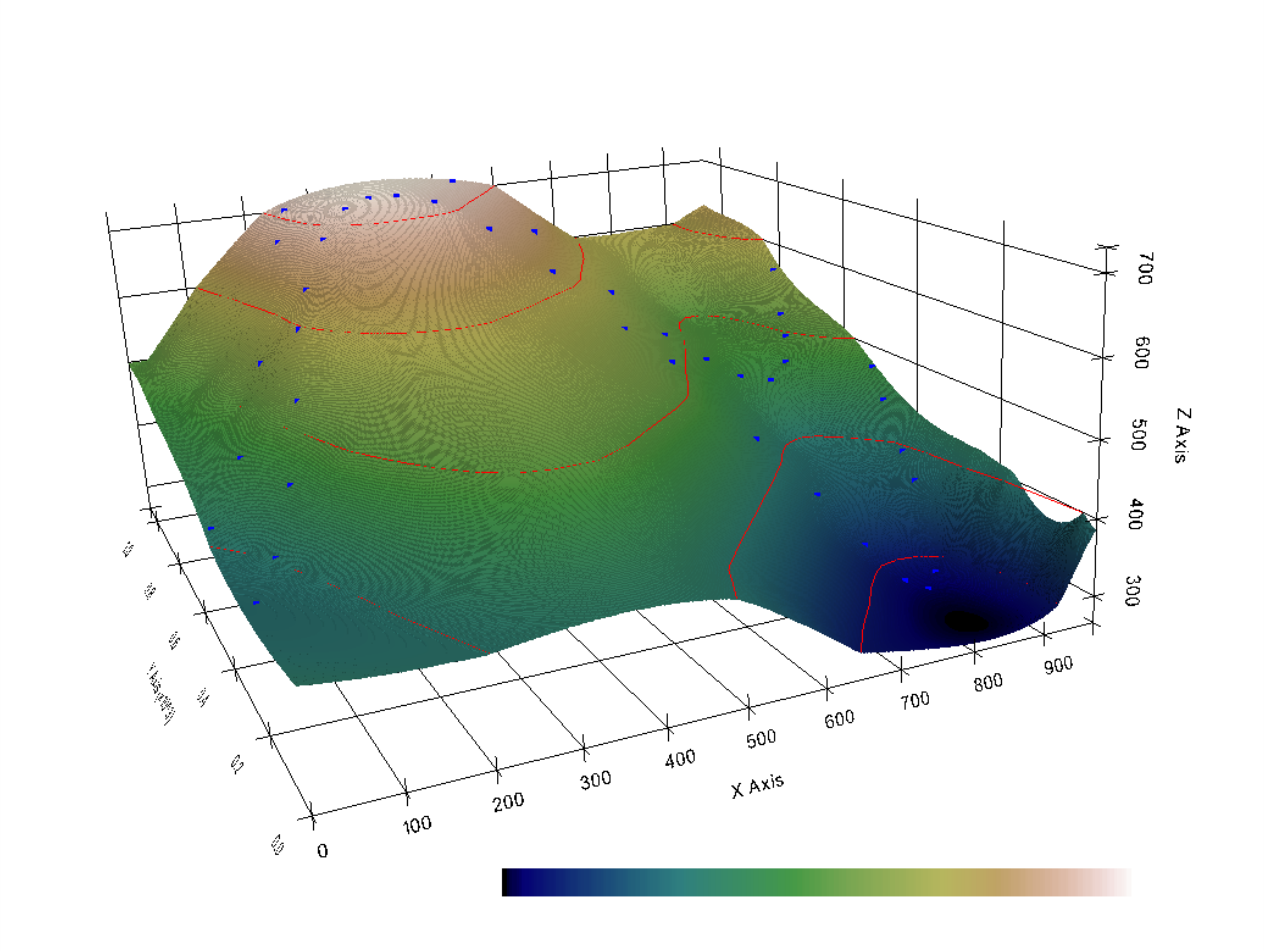
(continues on next page)

(continued from previous page)

```
im =plt.imshow(dem.read(1), cmap='gist_earth', vmin=250, vmax=750, extent=[0,972,0,1069])
cbar = plt.colorbar(im)
cbar.set_label('m')

plt.grid()
```

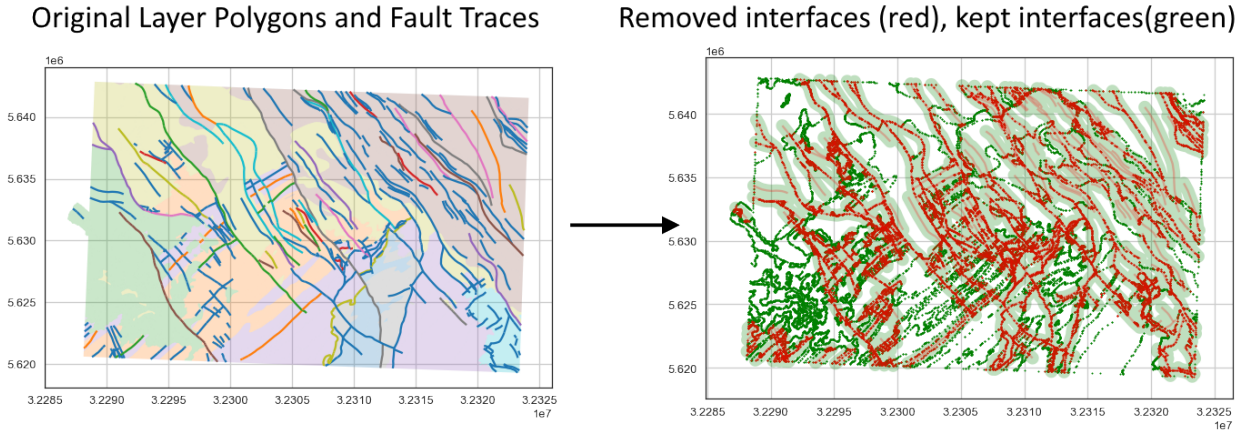


[20]:
```
p = pv.Plotter()

p.add_mesh(mesh=lines, color='red')

p.add_mesh(mesh=points_mesh, color='blue')

p.add_mesh(mesh=grid, scalars=grid["Elevation"], cmap='gist_earth')

p.camera_position =[(-283.285811675846, -1597.1397046051004, 1155.542325449192),
                    (577.9371599370799, 495.3480261506809, 381.7124055285182),
                    (0.17313457304419916, 0.27814381639313923, 0.9448070898437746)]

p.set_background('white')
p.show_grid(color='black')
p.show()
```



6.12 11 Removing Interface Points within Fault Buffers

The following will present how vertices of LineStrings stored as GeoDataFrames can be removed if they share the same points with the trace of a fault. The LineStrings in questions usually represent layer boundaries on geological maps. If vertices of these boundaries are sharing the same coordinates as the fault vertices, the interpolation in `GemPy`or a subsequent uncertainty analysis may not work properly.



Source: Geological Map 1:50,000, Geological Survey NRW

6.12.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/11_removing_interfaces_within_fault_buffers/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="11_removing_interfaces_within_
↪fault_buffers.zip", dirpath=file_path)
```

```
Downloading file '11_removing_interfaces_within_fault_buffers.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F11_removing_interfaces_within_
↪fault_buffers.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\
↪tutorial\data\11_removing_interfaces_within_fault_buffers'.
```

6.12.2 Loading Interface points and fault data

For this tutorial, we are using layer boundaries and fault traces from the Aachen area, Germany. Both data sets are saved as shape files and will be loaded with GeoPandas. The interface data is saved as a Polygon but LineStrings can be extracted from that.

The data used for GemGIS was provided by the Geological Survey NRW (GD NRW). It will be used under Datenlizenz Deutschland – Namensnennung – Version 2.0 (https://www.govdata.de/dl-de/by-2-0) with © Geowissenschaftliche Daten: IS GK 50, Projektgebiet Ballungsraum Aachen (2020).

Loading Data

The faults and interfaces are loaded using GeoPandas and converted to `EPSG:4647`.

[3]:
```
import geopandas as gpd

faults = gpd.read_file(file_path + 'GK50_Tektonik.shp')
faults = faults.to_crs('EPSG:4647')
faults.head()
```

[3]:

	ID	NAME	LEGENDE	\
0	90006	-	Abschiebung	
1	90185	Aachener Überschiebung	Auf- oder Überschiebung	
2	90196	Breinigerberg Überschiebung	Auf- oder Überschiebung	
3	90196	Breinigerberg Überschiebung	Auf- oder Überschiebung	
4	90196	Breinigerberg Überschiebung	Auf- oder Überschiebung	

	TYP	BEDECKUNG	\
0	Abschiebung (su)	keine quartäre Bedeckung	
1	Aufschiebung (sa)	keine quartäre Bedeckung	

(continues on next page)

(continued from previous page)

```
2  Aufschiebung (sa)  keine quartäre Bedeckung
3  Aufschiebung (sa)  keine quartäre Bedeckung
4  Aufschiebung (sa)  keine quartäre Bedeckung

                           HYDRAULIK              AKTIVITAET  \
0                       nicht bekannt  tektonisch nicht aktiv
1  hydraulisch wirksam - wasserführend  tektonisch nicht aktiv
2  hydraulisch wirksam - wasserführend  tektonisch nicht aktiv
3  hydraulisch wirksam - wasserführend  tektonisch nicht aktiv
4  hydraulisch wirksam - wasserführend  tektonisch nicht aktiv

                       BEDEUTUNG  BEARBEITUN  AENDERUNGS  AKTUALISIE  \
0  lokale bis regionale Bedeutung  2010-07-29  2010-07-29  2019-09-26
1         überregionale Bedeutung  2010-07-29  2010-07-29  2019-09-26
2  lokale bis regionale Bedeutung  2010-07-29  2010-07-29  2019-09-26
3  lokale bis regionale Bedeutung  2010-07-29  2010-07-29  2019-09-26
4  lokale bis regionale Bedeutung  2010-07-29  2010-07-29  2019-09-26

   SHAPE_Leng                                           geometry
0      236.48  LINESTRING (32322728.046 5638651.481, 32322623...
1      403.92  LINESTRING (32297864.303 5632557.868, 32297856...
2      184.94  LINESTRING (32300812.497 5620357.365, 32300932...
3     5738.56  LINESTRING (32301454.461 5620679.837, 32301589...
4      713.29  LINESTRING (32306882.447 5625319.830, 32306899...
```

[4]:
```
interfaces = gpd.read_file(file_path + 'GeologicalMapAachen.shp')
print(interfaces.crs)
interfaces.head()
```

```
epsg:4647
```

[4]:
```
   OBJECTID SYSTEM2     SERIE2 SSERIE2     STUFE2 SSTUFE2     SYSTEM1  \
0       131   Devon  Oberdevon       -  Famennium       -    Devonian
1       132       -          -       -          -       -     Neogene
2       133       -          -       -          -       -     Neogene
3       134       -          -       -          -       -  Cretaceous
4       135   Devon  Oberdevon       -  Famennium       -    Devonian

        SERIE1      SSERIE1     STUFE1  ... SGRUPPE SSYMBOL  \
0    Oberdevon            -   Frasnium  ...       -  dfrs+f
1      MiozÃ¤n  ObermiozÃ¤n          -  ...       -    mii7
2      MiozÃ¤n  ObermiozÃ¤n          -  ...       -    mii7
3   Oberkreide            -  Santonium  ...       -   krsah
4  Mitteldevon            -   Givetium  ...       -    dgfk

                       EINHEIT1                                    EINHEIT2  \
0  Frasnes- und Famenne-Schiefer                                           -
1                Inden-Formation                                           -
2                Inden-Formation                                           -
3               Aachen-Formation  Aachen-Formation, Hergenrath-Subformation
4                     Massenkalk                                           -

    SSY_GSY          GE_GG                                          GRUTEXT  \
```

(continues on next page)

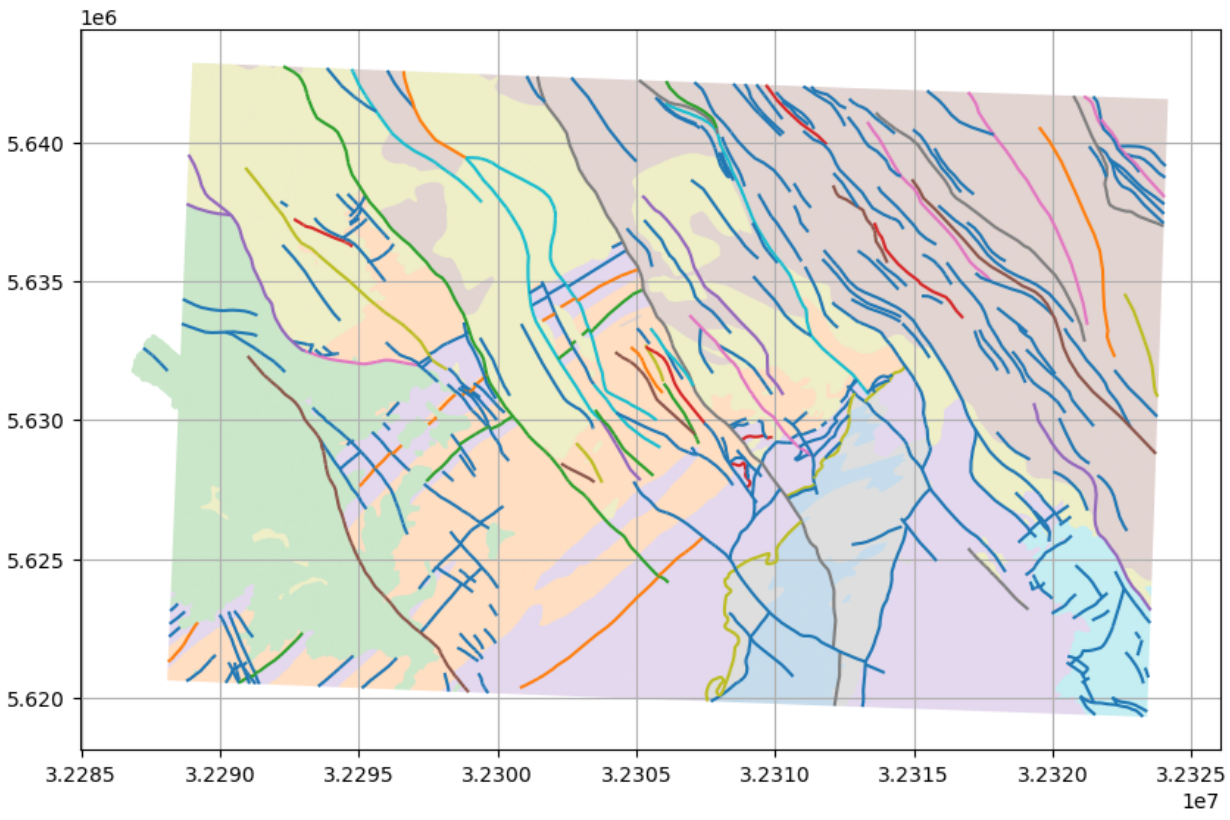
(continued from previous page)

```
0  dfrs+f,T 45403002130.00  Tonstein; untergeordnet Kalkmergel- bis Mergel...
1   mii7,s4  7000001110.00  Fein- bis Mittelsand; untergeordnet Schluff un...
2   mii7,s4  7000001110.00  Fein- bis Mittelsand; untergeordnet Schluff un...
3  krsah,ut 17400001115.00  Schluff und Ton; untergeordnet Fein- bis Mitte...
4   dgfk,KD 46800002015.00                          Kalkstein und Dolomitstein

   SHAPE_Leng SHAPE_Area                                           geometry
0      459.12   12043.31  POLYGON ((32299083.709 5631034.983, 32299164.0...
1     4409.72  487296.28  POLYGON ((32317556.700 5633860.692, 32317497.2...
2     7644.70 2347096.98  POLYGON ((32310428.554 5638385.230, 32310400.4...
3    19078.93 2340702.57  POLYGON ((32291096.918 5622295.519, 32290989.7...
4     4615.89  405124.98  POLYGON ((32307344.796 5625848.944, 32307389.6...

[5 rows x 22 columns]
```

Plotting the Input Data

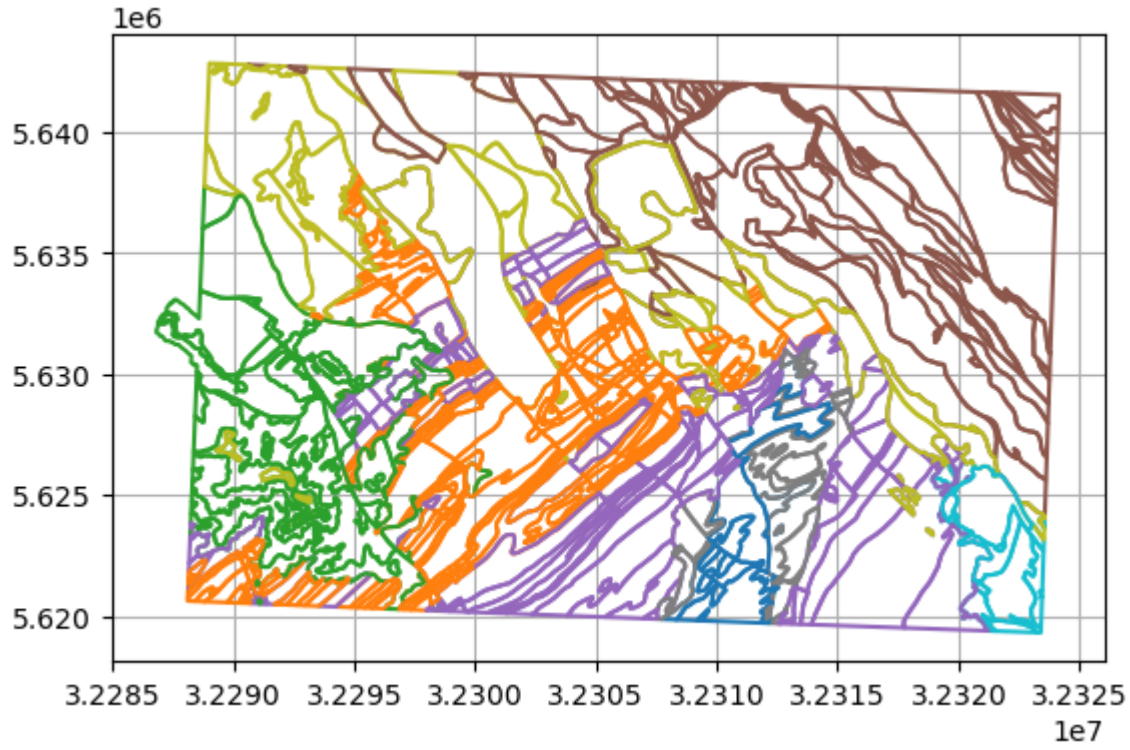
The data can be plotted using the built-in GeoPandas plotting function.

[5]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(figsize=(10,10))
faults.plot(ax=ax, aspect='equal', column='NAME')
interfaces.plot(ax=ax, aspect='equal', column = 'SYSTEM1', alpha = 0.25)
plt.grid()
```



Exploding Polygons to LineStrings to show the different mapped layers.

[6]:
```
gg.vector.explode_polygons(interfaces).plot(column='SYSTEM1')
plt.grid()
```



6.12.3 Removing interfaces within fault buffers

The removal of interfaces from within fault buffers consists of multiple steps:

- Creating a buffer around a fault
- Creating buffers for all faults
- Subtracting Geometry objects from another
- Removing object within buffer
- Removing objects within buffer
- Removing all interfaces from all faults

Creating a buffer around a fault

A buffer needs to be created around a fault so that all vertices of interfaces can be deleted from this buffer. A section (782) of the Sandgewand Fault will be taken to demonstrate the feature.

Original fault LineString.

[7]:
```
faults.loc[782].geometry
```

[7]:

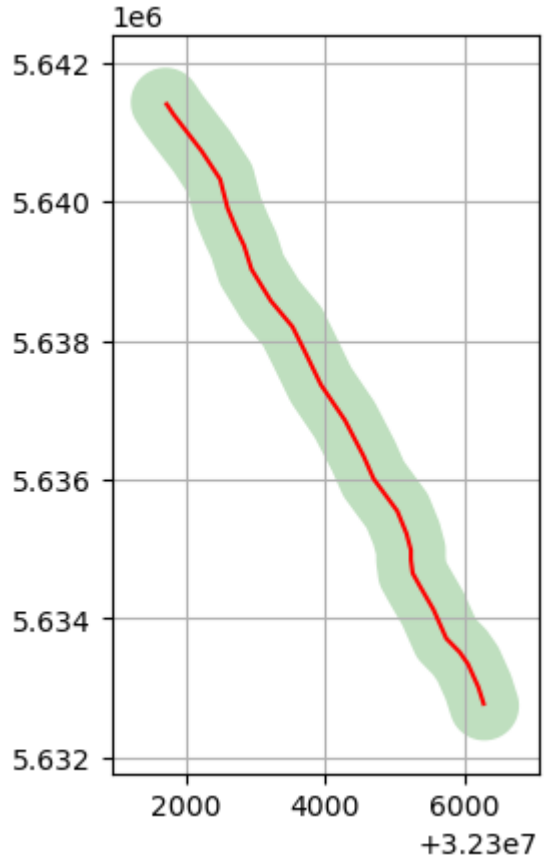
Buffer created around LineString.

[8]:
```
buffered_fault = gg.vector.create_buffer(geom_object=faults.loc[782].geometry,
                                         distance=500)
buffered_fault
```

[8]:

The buffer and the original LineString can also be plotted to illustrate the buffering.

[9]:
```
fig, ax = plt.subplots(1,1)

gpd.GeoDataFrame(geometry=[faults.loc[782].geometry]).plot(ax=ax, aspect='equal', color=
↪'red')
gpd.GeoDataFrame(geometry=[buffered_fault]).plot(ax=ax, aspect='equal', color='green',␣
↪alpha=0.25)

plt.grid()
```

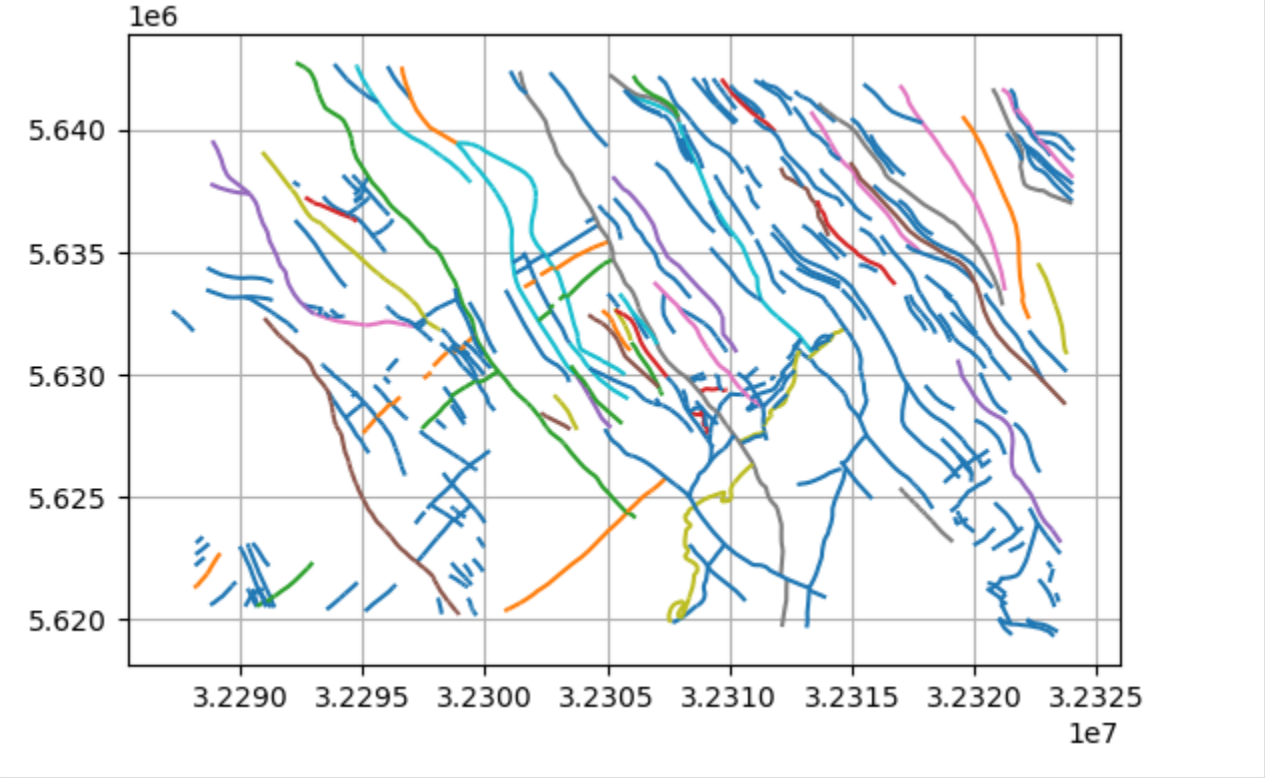


Creating Buffers around all faults

To avoid creating too many single buffers to check whether an interface is within a buffer or not, one single buffer for all faults can be created. It will be returned as Shapely Multipolygon.

Plotting all faults in the Aachen area.

[10]:
```
faults.plot(aspect='equal', column='NAME')
plt.grid()
```



Creating a buffer of 500 m around all faults.

[11]:
```
polygon = gg.vector.create_unified_buffer(geom_object=faults,
                                          distance=500)

polygon
```

[11]:

[12]:
```
type(polygon)
```

[12]:
```
shapely.geometry.multipolygon.MultiPolygon
```

Subtracting Geometry Objects from another

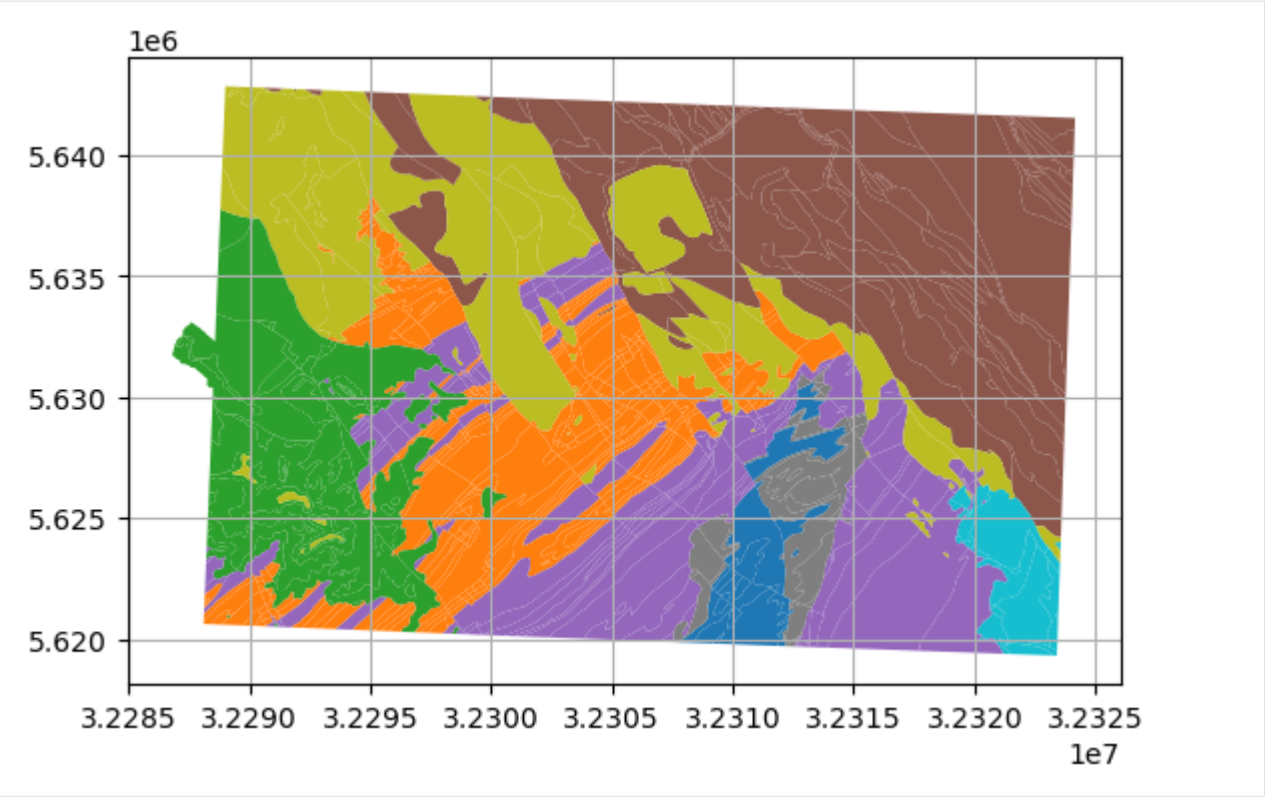
The next step is to subtract the vertices of an interface polygon, linestring or points from the fault bufffer. A layer boundary sharing vertices with the Sandgewand fault will be used for that (198). The polygons will be exploded to LineStrings and MultiLineStrings first so that no artifacts will remain.

[13]:
```
interfaces_ls = gg.vector.explode_polygons(gdf=interfaces)
interfaces_ls.head()
```

[13]:
```
   OBJECTID SYSTEM2     SERIE2 SSERIE2     STUFE2 SSTUFE2     SYSTEM1  \
0       131   Devon  Oberdevon       -  Famennium       -    Devonian
1       132       -          -       -          -       -     Neogene
2       133       -          -       -          -       -     Neogene
3       134       -          -       -          -       -  Cretaceous
4       135   Devon  Oberdevon       -  Famennium       -    Devonian

        SERIE1      SSERIE1     STUFE1  ... SGRUPPE SSYMBOL  \
0    Oberdevon            -   Frasnium  ...       -  dfrs+f
1      MiozÃ¤n  ObermiozÃ¤n          -  ...       -    mii7
2      MiozÃ¤n  ObermiozÃ¤n          -  ...       -    mii7
3   Oberkreide            -  Santonium  ...       -   krsah
4  Mitteldevon            -   Givetium  ...       -    dgfk

                      EINHEIT1                                    EINHEIT2  \
0  Frasnes- und Famenne-Schiefer                                         -
1               Inden-Formation                                          -
2               Inden-Formation                                          -
3              Aachen-Formation  Aachen-Formation, Hergenrath-Subformation
4                    Massenkalk                                         -

    SSY_GSY          GE_GG                                        GRUTEXT  \
0  dfrs+f,T 45403002130.00  Tonstein; untergeordnet Kalkmergel- bis Mergel...
1   mii7,s4  7000001110.00  Fein- bis Mittelsand; untergeordnet Schluff un...
2   mii7,s4  7000001110.00  Fein- bis Mittelsand; untergeordnet Schluff un...
3  krsah,ut 17400001115.00  Schluff und Ton; untergeordnet Fein- bis Mitte...
4   dgfk,KD 46800002015.00                       Kalkstein und Dolomitstein

   SHAPE_Leng SHAPE_Area                                          geometry
0      459.12   12043.31  LINESTRING (32299083.709 5631034.983, 32299164...
1     4409.72  487296.28  LINESTRING (32317556.700 5633860.692, 32317497...
2     7644.70 2347096.98  LINESTRING (32310428.554 5638385.230, 32310400...
3    19078.93 2340702.57  MULTILINESTRING ((32291096.918 5622295.519, 32...
4     4615.89  405124.98  LINESTRING (32307344.796 5625848.944, 32307389...

[5 rows x 22 columns]
```

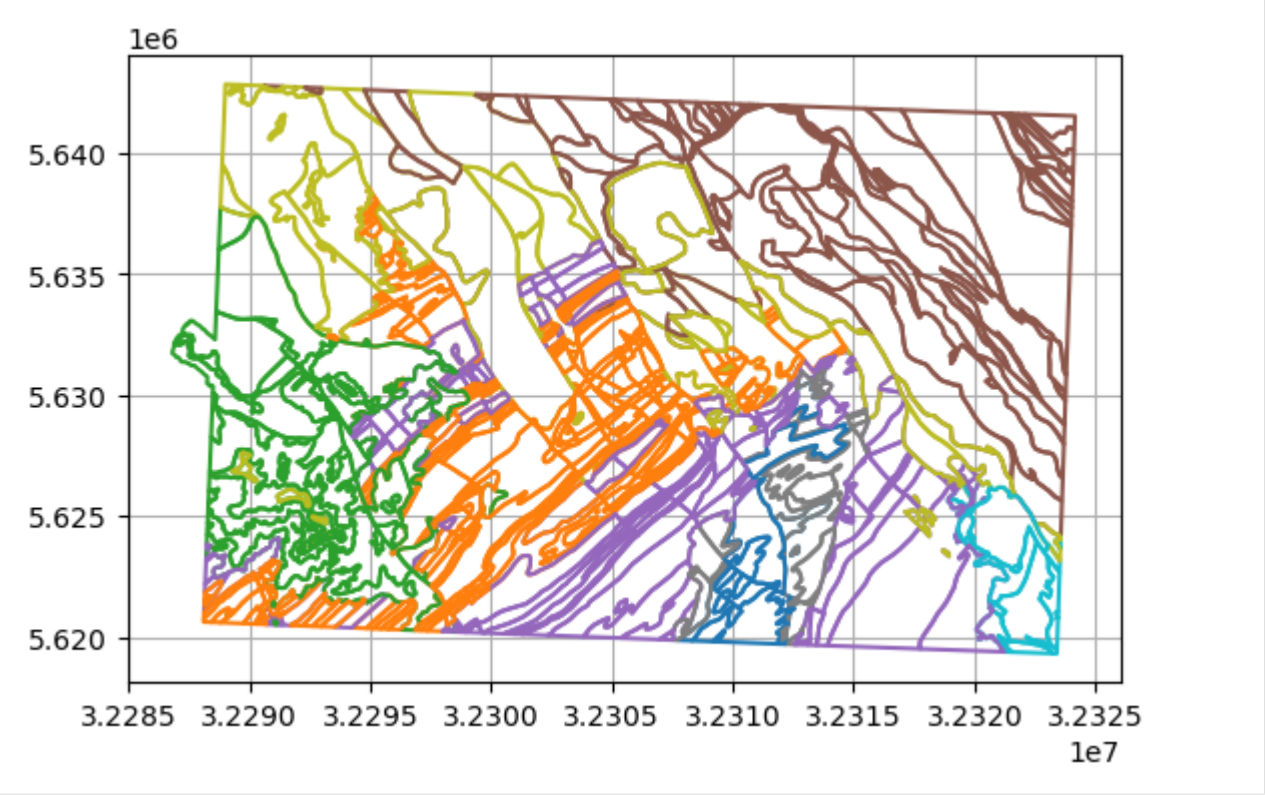
Original Polygon Data.

[14]:
```
interfaces.plot(column='SYSTEM1')
plt.grid()
```



LineStrings and MultiLineStrings after exploding the polygons.

[15]:
```
interfaces_ls.plot(column='SYSTEM1')
plt.grid()
```



LineString that will be buffered.

[16]:
```
interfaces_ls.loc[198].geometry
```

[16]:

Buffer for the LineString.

[17]:
```
buffered_fault
```

[17]:

Subtracting the geometry type from each other.

[18]:
```
result_geom_object = gg.vector.subtract_geom_objects(geom_object1=interfaces_ls.loc[198].
↪geometry,
                                                     geom_object2=buffered_fault)

result_geom_object
```

[18]:

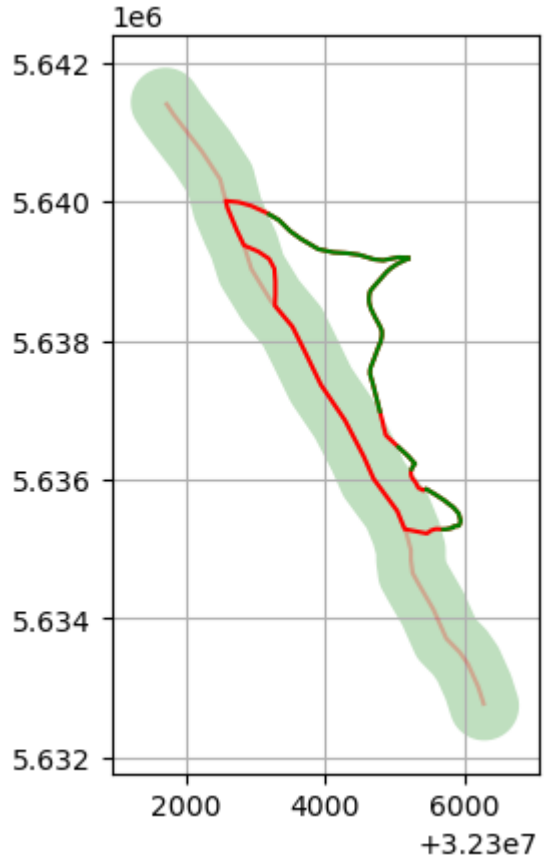
Plotting the removed interfaces (red) and the kept interfaces (green).

[19]:
```
fig, ax = plt.subplots(1,1)
gpd.GeoDataFrame(geometry=[interfaces_ls.loc[198].geometry]).plot(ax=ax, aspect='equal',␣
↪color='red')

gpd.GeoDataFrame(geometry=[result_geom_object]).plot(ax=ax, aspect='equal', color='green
↪')

gpd.GeoDataFrame(geometry=[buffered_fault]).plot(ax=ax, aspect='equal', color='green',␣
↪alpha=0.25)
```

(continues on next page)

(continued from previous page)

```
gpd.GeoDataFrame(geometry=[faults.loc[782].geometry]).plot(ax=ax, aspect='equal', color=
↪'red', alpha=0.25)

plt.grid()
```



Performing the subtraction for the entire fault polygon.

[20]:
```
result_geom_object = gg.vector.subtract_geom_objects(geom_object1=interfaces_ls.loc[198].
↪geometry,
                                                     geom_object2=polygon)

result_geom_object
```

[20]:

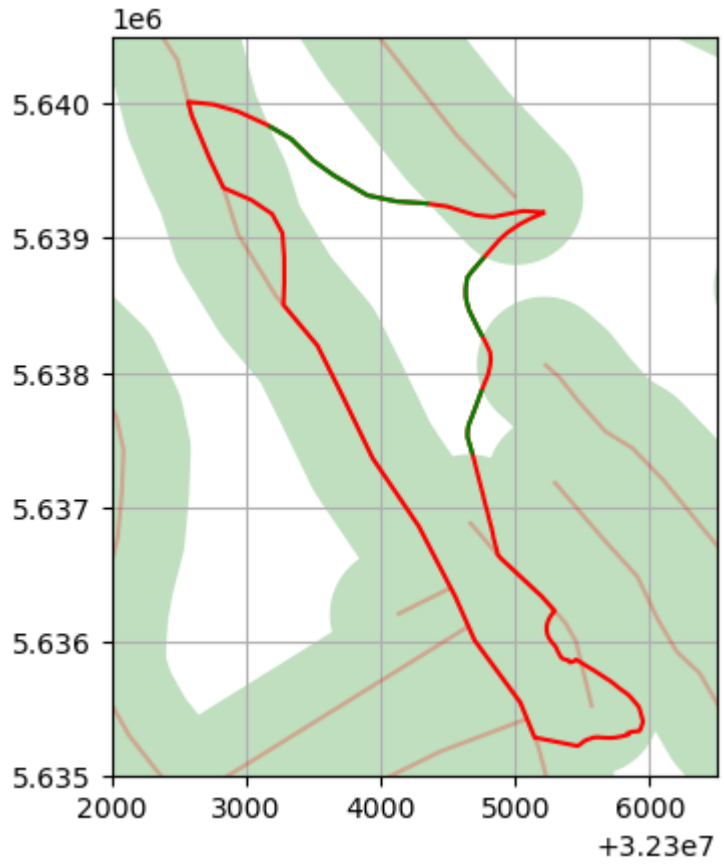
Plotting the removed interfaces (red) and the kept interfaces (green) when substracting it from the entire fault buffer polygon.

[21]:
```
fig, ax = plt.subplots(1,1)
gpd.GeoDataFrame(geometry=[interfaces_ls.loc[198].geometry]).plot(ax=ax, aspect='equal',␣
↪color='red')

gpd.GeoDataFrame(geometry=[result_geom_object]).plot(ax=ax, aspect='equal', color='green
↪')
```

(continues on next page)

(continued from previous page)

```
gpd.GeoDataFrame(geometry=[polygon]).plot(ax=ax, aspect='equal', color='green', alpha=0.
↪25)
faults.plot(ax=ax, aspect='equal', color='red', alpha=0.25)
plt.xlim(32302000, 32306500)
plt.ylim(5.635e6, 5.6405e6)
plt.grid()
```



Removing object within buffer

The two previous steps of creating a buffer and removing the objects within the buffer can be combined in the function `remove_object_within_buffer(..)`.

Displaying a second layer around the same fault.

[22]:
```
interfaces_ls.loc[374].geometry
```

[22]:

Buffer for the same fault as before.

[23]:
```
buffered_fault
```

[23]:

Removing the object from within the buffer.

[24]:
```
outside, inside = gg.vector.remove_object_within_buffer(buffer_object=faults.loc[782].
↪geometry,
                                                        buffered_object=interfaces_ls.
↪loc[374].geometry,
                                                        distance=500)
```

Interfaces that were not removed from the layer.

[25]:
```
outside
```

[25]:

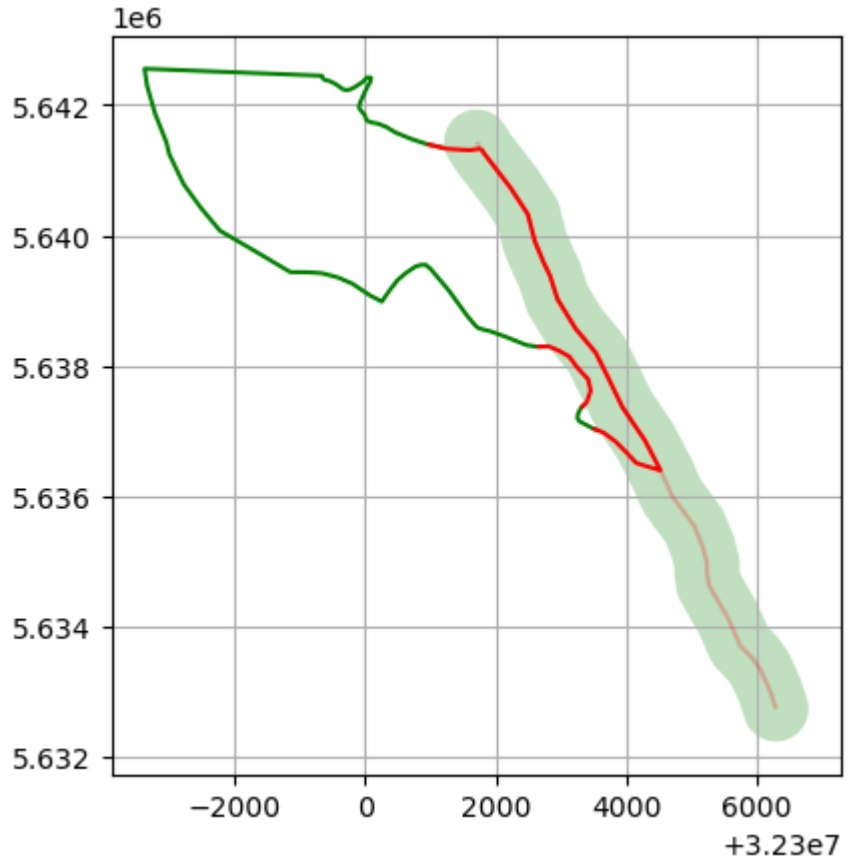
Interfaces that were removed from the layer.

[26]:
```
inside
```

[26]:

Plotting the interfaces that were removed (red) and the interfaces that were kept (green).

[27]:
```
fig, ax = plt.subplots(1,1)
gpd.GeoDataFrame(geometry=[outside]).plot(ax=ax, aspect='equal', color='green')

gpd.GeoDataFrame(geometry=[inside]).plot(ax=ax, aspect='equal', color='red')

gpd.GeoDataFrame(geometry=[buffered_fault]).plot(ax=ax, aspect='equal', color='green',␣
↪alpha=0.25)
gpd.GeoDataFrame(geometry=[faults.loc[782].geometry]).plot(ax=ax, aspect='equal', color=
↪'red', alpha=0.25)

plt.grid()
```



Performing the same operation with the entire fault polygon

[28]:
```
outside, inside = gg.vector.remove_object_within_buffer(buffer_object=polygon,
                                                         buffered_object=interfaces_ls.
↪loc[374].geometry,
                                                         distance=None,
                                                         buffer=False)
```

The resulting interfaces that are left are less than before as other faults also contributed to the buffering.

[29]:
```
outside
```

[29]:

More interfaces were removed as other faults also contributed to the buffering.

[30]:
```
inside
```

[30]:

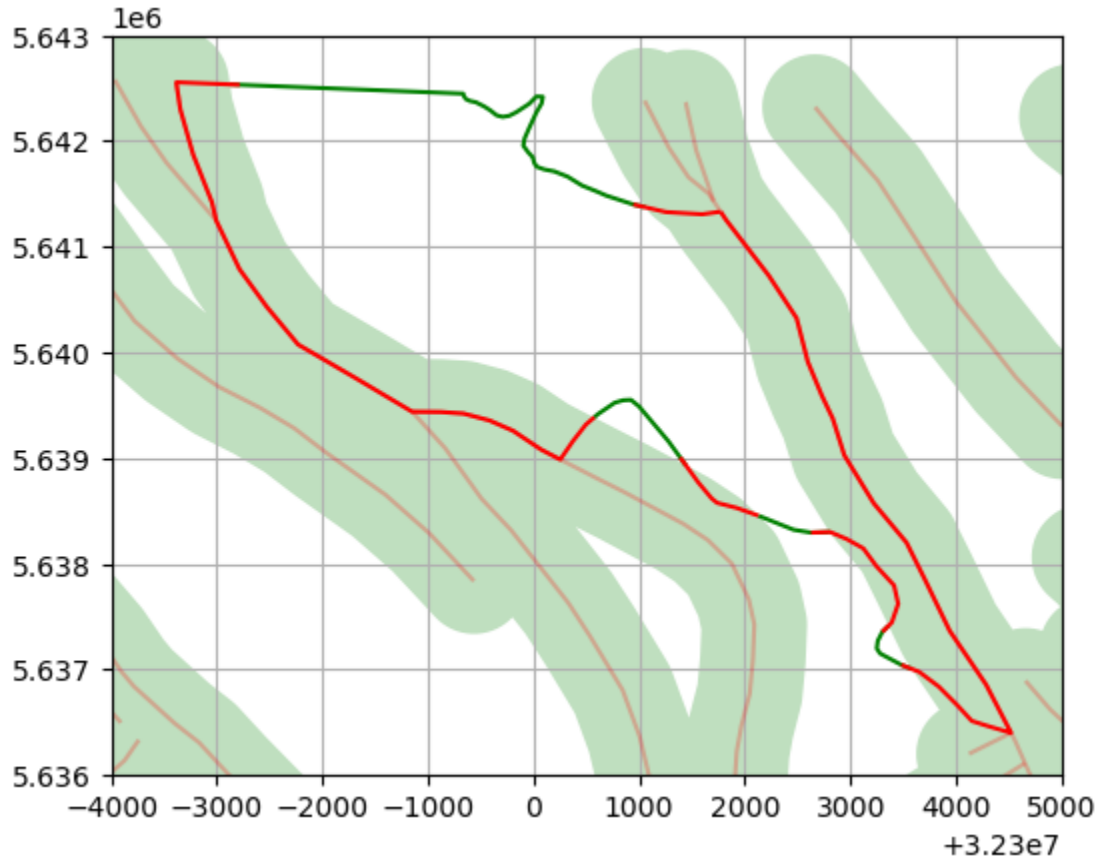
Plotting the interfaces that were removed (red) and the interfaces that were kept (green).

[31]:
```
fig, ax = plt.subplots(1,1)
gpd.GeoDataFrame(geometry=[outside]).plot(ax=ax, aspect='equal', color='green')

gpd.GeoDataFrame(geometry=[inside]).plot(ax=ax, aspect='equal', color='red')
```

(continues on next page)

(continued from previous page)

```
gpd.GeoDataFrame(geometry=[polygon]).plot(ax=ax, aspect='equal', color='green', alpha=0.
↪25)
faults.plot(ax=ax, aspect='equal', color='red', alpha=0.25)

plt.xlim(32296000, 32305000)
plt.ylim(5.636e6, 5.643e6)
plt.grid()
```



Removing objects within buffer

The next step is to remove multiple objects from the fault buffer polygon at the same time.

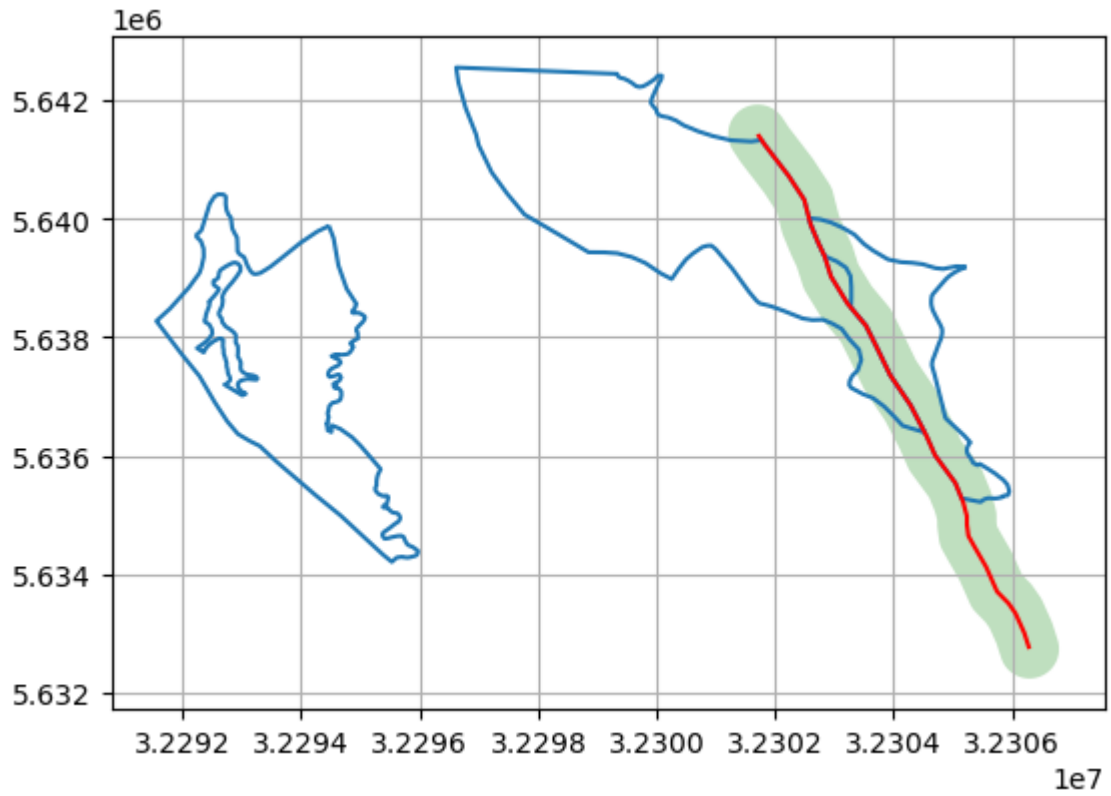
Therefore, we create a GeoDataFrame with interfaces objects and use once only one fault and then all faults.

[32]:
```
interfaces_gdf = gpd.GeoDataFrame(geometry=[interfaces_ls.loc[374].geometry,
                                            interfaces_ls.loc[198].geometry,
                                            interfaces_ls.loc[800].geometry])
interfaces_gdf
```

[32]:
```
                                            geometry
0  LINESTRING (32301771.153 5641329.303, 32301821...
1  LINESTRING (32305222.632 5639189.905, 32305218...
2  MULTILINESTRING ((32294465.330 5639887.282, 32...
```

Plotting the interfaces and the fault including its buffer. Two layers are touching the fault, the third one does not.

[33]:
```
fig, ax = plt.subplots(1,1)

interfaces_gdf.plot(ax=ax, aspect='equal')
gpd.GeoDataFrame(geometry=[gg.vector.create_buffer(faults.loc[782].geometry, 500)]).
↪plot(ax=ax, aspect='equal', color='green', alpha=0.25)
gpd.GeoDataFrame(geometry=[faults.loc[782].geometry]).plot(ax=ax, aspect='equal', color=
↪'red')

plt.grid()
```



Removing the interfaces from the fault buffer.

[34]:
```
gdf_out, gdf_in = gg.vector.remove_objects_within_buffer(buffer_object=faults.loc[782].
↪geometry,
                                                         buffered_objects_gdf=interfaces_
↪gdf,
                                                         distance=500,
                                                         return_gdfs=True)
```

Resulting GeoDataFrame containing the geometry objects that remained outside the fault buffers.

[35]:
```
gdf_out
```

[35]:
```
                                             geometry
0  MULTILINESTRING ((32303580.528 5637004.216, 32...
1  MULTILINESTRING ((32305222.632 5639189.905, 32...
```

(continues on next page)

(continued from previous page)

```
2  MULTILINESTRING ((32294465.330 5639887.282, 32...
```

Resulting GeoDataFrame containing the geometry objects that were within the buffer. The last LineString is empty since the third layer did not touch the fault.

[36]:
```
gdf_in
```

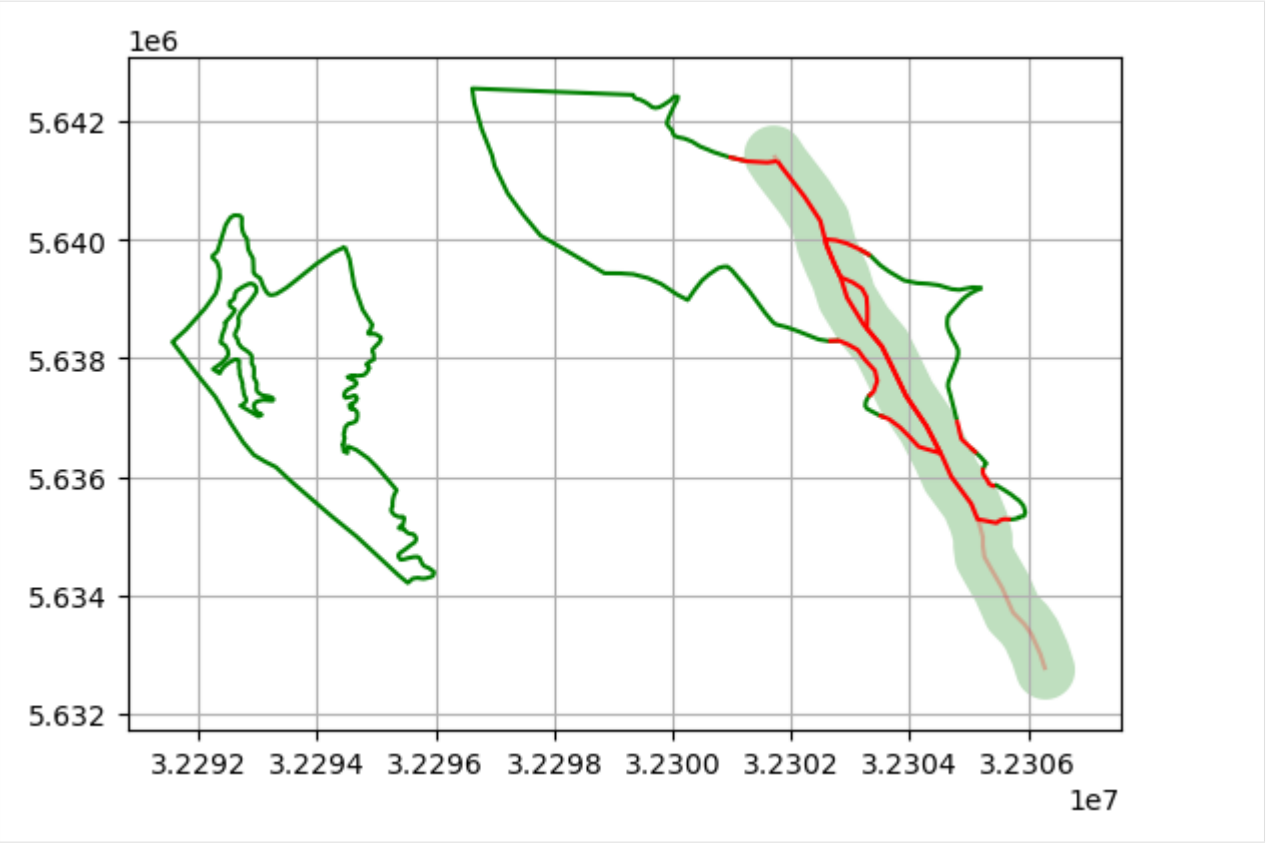
[36]:
```
                                          geometry
0  MULTILINESTRING ((32301771.153 5641329.303, 32...
1  MULTILINESTRING ((32304795.577 5636961.054, 32...
2                                LINESTRING Z EMPTY
```

Plotting the vertices that were kept (green) and removed (red).

[37]:
```
faults_gdf = gpd.GeoDataFrame(geometry=[faults.loc[782].geometry])
faults_gdf
```

[37]:
```
                                          geometry
0  LINESTRING (32301705.663 5641428.471, 32301705...
```

[38]:
```
fig, ax = plt.subplots(1,1)

gdf_out.plot(ax=ax, aspect='equal', color='green')
gdf_in[~gdf_in.is_empty].plot(ax=ax, aspect='equal', color='red')
faults_gdf.plot(ax=ax, aspect='equal', color='red', alpha=0.25)
gpd.GeoDataFrame(geometry=[gg.vector.create_buffer(faults.loc[782].geometry, 500)]).
↪plot(ax=ax, aspect='equal', color='green', alpha=0.25)
plt.grid()
```



Performing the same operation for the entire fault buffer polyon.

[39]:
```
gdf_out, gdf_in = gg.vector.remove_objects_within_buffer(buffer_object=polygon,
                                                         buffered_objects_gdf=interfaces_
↪gdf,
                                                         distance=None,
                                                         return_gdfs=True,
                                                         buffer=False)
```

Resulting GeoDataFrame containing the geometry objects that remained outside the fault buffers.

[40]:
```
gdf_out
```

[40]:
```
                                             geometry
0  MULTILINESTRING ((32303580.528 5637004.216, 32...
1  MULTILINESTRING ((32304771.851 5638858.254, 32...
2  MULTILINESTRING ((32291916.587 5638630.583, 32...
```

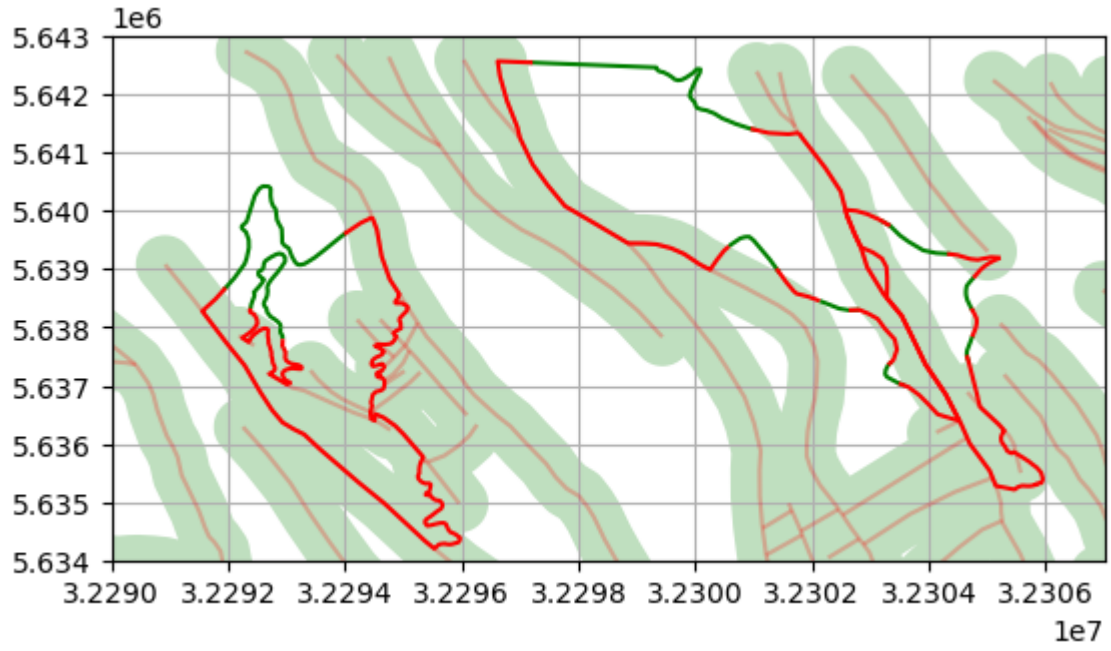
Resulting GeoDataFrame containing the geometry objects that were within the buffer.

[41]:
```
gdf_in
```

[41]:
```
                                             geometry
0  MULTILINESTRING ((32301771.153 5641329.303, 32...
1  MULTILINESTRING ((32305222.632 5639189.905, 32...
2  MULTILINESTRING ((32294465.330 5639887.282, 32...
```

Plotting the vertices that were kept (green) and removed (red).

[42]:
```
fig, ax = plt.subplots(1,1)

gdf_out.plot(ax=ax, aspect='equal', color='green')
gdf_in[~gdf_in.is_empty].plot(ax=ax, aspect='equal', color='red')
faults.plot(ax=ax, aspect='equal', color='red', alpha=0.25)
gpd.GeoDataFrame(geometry=[polygon]).plot(ax=ax, aspect='equal', color='green', alpha=0.
↪25)

plt.xlim(32290000, 32307000)
plt.ylim(5.634e6, 5.643e6)
plt.grid()
```



Removing all Interfaces from all Faults

The functions above can now be generalized to the function `remove_interfaces_within_fault_buffers(..)` were the loaded fault GeoDataFrame and the loaded polygon GeoDataFrame are loaded.

[43]:
```
gdf_out, gdf_in = gg.vector.remove_interfaces_within_fault_buffers(fault_gdf=faults,
                                                                   interfaces_
↪gdf=interfaces,
                                                                   distance=250)
```

The GeoDataFrame with all its column and Shapely points that remained outside a fault polygon.

[44]:
```
gdf_out.head()
```

[44]:

	OBJECTID	SYSTEM2	SERIE2	SSERIE2	STUFE2	SSTUFE2	SYSTEM1	SERIE1	\
0	132	-	-	-	-	-	Neogene	MiozÃ¤n	
1	132	-	-	-	-	-	Neogene	MiozÃ¤n	
2	132	-	-	-	-	-	Neogene	MiozÃ¤n	
3	132	-	-	-	-	-	Neogene	MiozÃ¤n	

(continues on next page)

(continued from previous page)

```
4       132       -      -       -      -       -  Neogene  MiozÃ¤n

       SSERIE1 STUFE1  ...         EINHEIT1 EINHEIT2  SSY_GSY         GE_GG  \
0  ObermiozÃ¤n      -  ...  Inden-Formation        -  mii7,s4 7000001110.00
1  ObermiozÃ¤n      -  ...  Inden-Formation        -  mii7,s4 7000001110.00
2  ObermiozÃ¤n      -  ...  Inden-Formation        -  mii7,s4 7000001110.00
3  ObermiozÃ¤n      -  ...  Inden-Formation        -  mii7,s4 7000001110.00
4  ObermiozÃ¤n      -  ...  Inden-Formation        -  mii7,s4 7000001110.00

                                           GRUTEXT SHAPE_Leng SHAPE_Area  \
0  Fein- bis Mittelsand; untergeordnet Schluff un...    4409.72  487296.28
1  Fein- bis Mittelsand; untergeordnet Schluff un...    4409.72  487296.28
2  Fein- bis Mittelsand; untergeordnet Schluff un...    4409.72  487296.28
3  Fein- bis Mittelsand; untergeordnet Schluff un...    4409.72  487296.28
4  Fein- bis Mittelsand; untergeordnet Schluff un...    4409.72  487296.28

                          geometry           X          Y
0  POINT (32316610.248 5635509.674) 32316610.25 5635509.67
1  POINT (32316651.533 5635494.023) 32316651.53 5635494.02
2  POINT (32316651.919 5635493.752) 32316651.92 5635493.75
3  POINT (32316811.783 5635381.554) 32316811.78 5635381.55
4  POINT (32317075.334 5635058.862) 32317075.33 5635058.86

[5 rows x 24 columns]
```

The GeoDataFrame with all its column and Shapely points that are inside a fault polygon.

[45]:
```
gdf_in.head()
```

[45]:
```
   OBJECTID SYSTEM2     SERIE2 SSERIE2     STUFE2 SSTUFE2   SYSTEM1  \
0       131   Devon  Oberdevon       -  Famennium       -  Devonian
1       131   Devon  Oberdevon       -  Famennium       -  Devonian
2       131   Devon  Oberdevon       -  Famennium       -  Devonian
3       131   Devon  Oberdevon       -  Famennium       -  Devonian
4       131   Devon  Oberdevon       -  Famennium       -  Devonian

      SERIE1 SSERIE1    STUFE1  ...                       EINHEIT1 EINHEIT2  \
0  Oberdevon       -  Frasnium  ...  Frasnes- und Famenne-Schiefer        -
1  Oberdevon       -  Frasnium  ...  Frasnes- und Famenne-Schiefer        -
2  Oberdevon       -  Frasnium  ...  Frasnes- und Famenne-Schiefer        -
3  Oberdevon       -  Frasnium  ...  Frasnes- und Famenne-Schiefer        -
4  Oberdevon       -  Frasnium  ...  Frasnes- und Famenne-Schiefer        -

    SSY_GSY          GE_GG                                          GRUTEXT  \
0  dfrs+f,T 45403002130.00  Tonstein; untergeordnet Kalkmergel- bis Mergel...
1  dfrs+f,T 45403002130.00  Tonstein; untergeordnet Kalkmergel- bis Mergel...
2  dfrs+f,T 45403002130.00  Tonstein; untergeordnet Kalkmergel- bis Mergel...
3  dfrs+f,T 45403002130.00  Tonstein; untergeordnet Kalkmergel- bis Mergel...
4  dfrs+f,T 45403002130.00  Tonstein; untergeordnet Kalkmergel- bis Mergel...

  SHAPE_Leng SHAPE_Area                          geometry           X  \
0     459.12   12043.31  POINT (32299083.709 5631034.983) 32299083.71
1     459.12   12043.31  POINT (32299164.005 5630970.066) 32299164.01
```

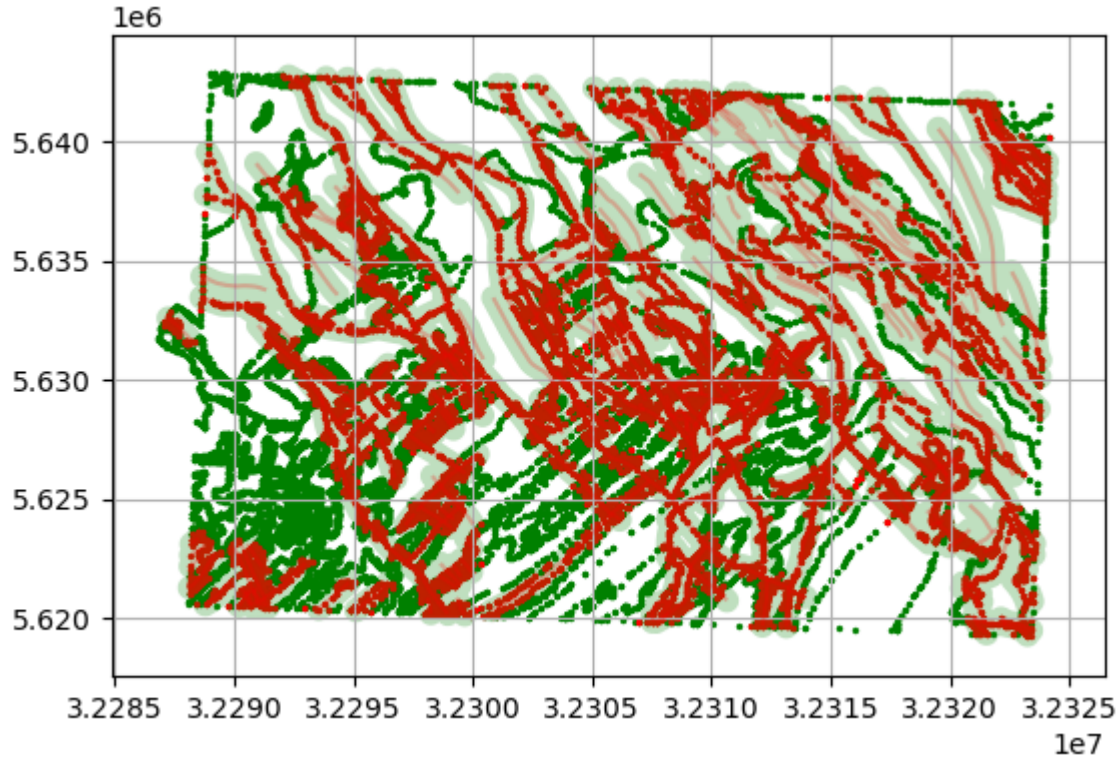
(continues on next page)

(continued from previous page)

```
2     459.12   12043.31  POINT (32299123.225 5630909.550) 32299123.23
3     459.12   12043.31  POINT (32299088.346 5630931.022) 32299088.35
4     459.12   12043.31  POINT (32298996.618 5630993.458) 32298996.62

            Y
0 5631034.98
1 5630970.07
2 5630909.55
3 5630931.02
4 5630993.46

[5 rows x 24 columns]
```

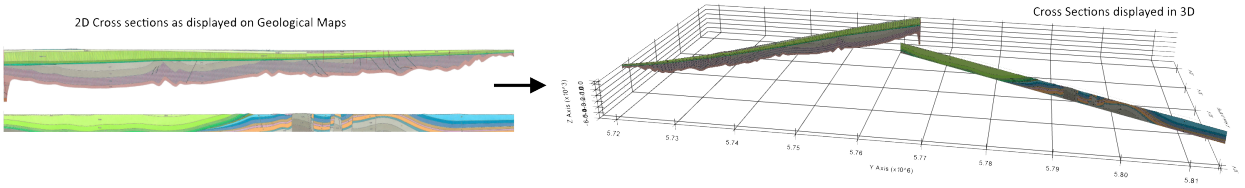
Plotting the final result of interfaces that are outside a fault polygon (green) and interfaces inside a fault polygon (red).

[46]:
```
fig, ax = plt.subplots(1,1)

gdf_out.plot(ax=ax, aspect='equal', color='green',markersize=2)
gdf_in[~gdf_in.is_empty].plot(ax=ax, aspect='equal', color='red', markersize=2)
faults.plot(ax=ax, aspect='equal', color='red', alpha=0.25)
gpd.GeoDataFrame(geometry=[polygon]).plot(ax=ax, aspect='equal', color='green', alpha=0.
↪25)

plt.grid()
```



6.13 12 Visualizing Geological Cross Sections with PyVista

Geological Cross Sections obtained from Geological Maps can be loaded and plotted with PyVista in GemGIS. The following illustrates how to get data from a Geological Map into GemGIS.

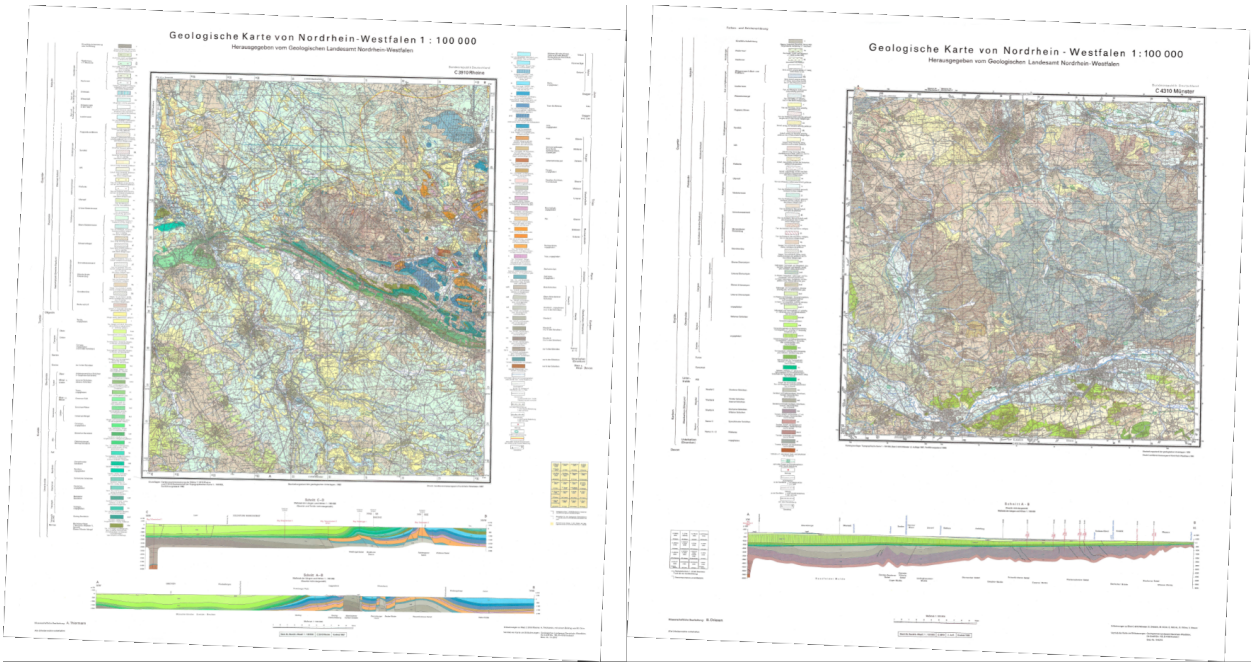


6.13.1 Data Preparation

The data used for GemGIS is obtained from *OpenDataNRW*. It will be used under Datenlizenz Deutschland – Namensnennung – Version 2.0 (https://www.govdata.de/dl-de/by-2-0) with © Geowissenschaftliche Daten: Analoges Kartenwerk der Geologischen Karte von Nordrhein-Westfalen 1:100.000 (2020).

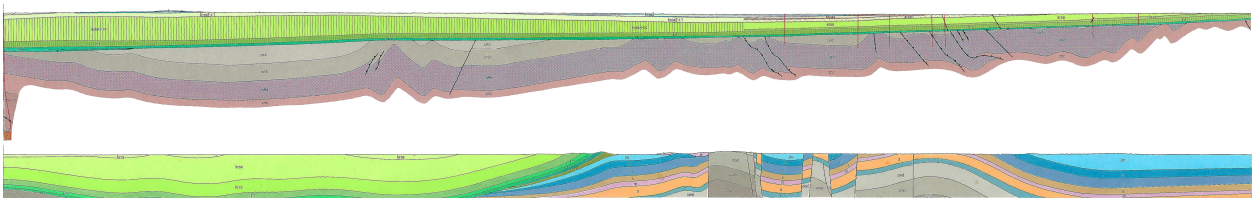
Geological Maps

For the visualization of cross sections, the geological maps `Münster` and `Rheine` were taken as shown below.



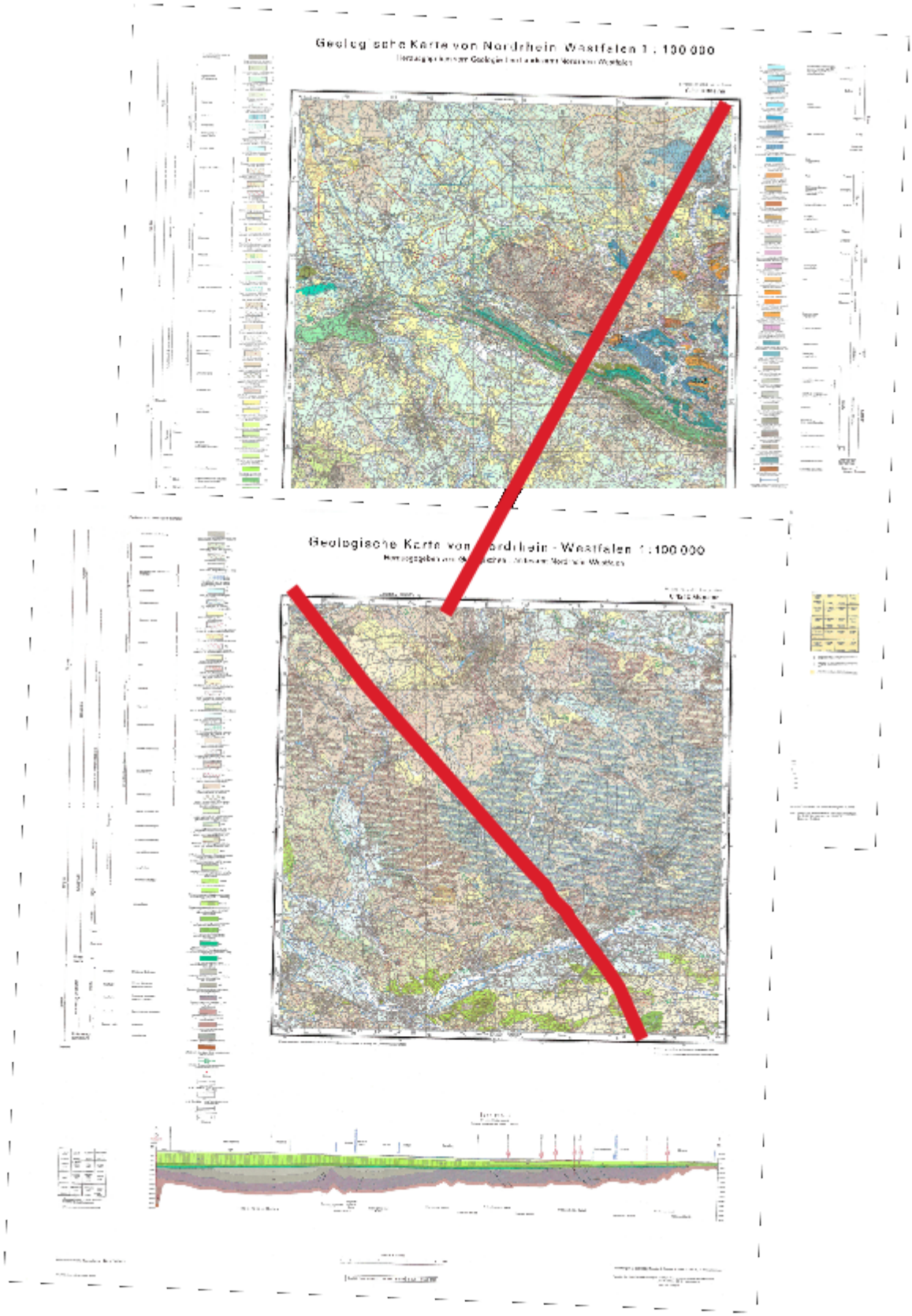
Geological Cross Section

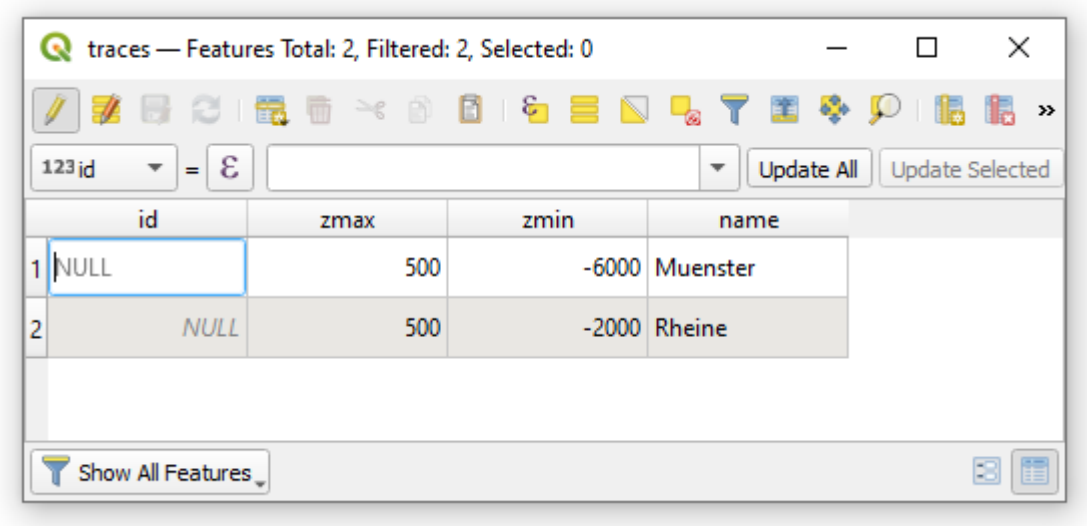
From these maps, the geological cross sections were extracted with an image editing software, tilted so that they are completely horizontal and cropped to the extent of the cross section. The axes were also cut off.



Traces of Geological Cross Sections

In order to locate the cross sections in space, the traces of the profiles need to be digitized and the vertical ranges of the cross sections need to be provided as attributes of the shape file. This can be done for example in QGIS.


Geologische Karte von Nordrhein-Westfalen 1 : 100 000
Geologische Karte von Nordrhein-Westfalen 1 : 100 000



6.13.2 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/12_visualizing_cross_sections_in_pyvista/'
```

```
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\numpy\_distributor_
↪init.py:30: UserWarning: loaded more than 1 DLL from .libs:
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\numpy\.libs\
↪libopenblas.FB5AE2TYXYH2IJRDKGDGQ3XBKLKTF43H.gfortran-win_amd64.dll
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\numpy\.libs\
↪libopenblas64__v0.3.23-246-g3d31191b-gcc_10_3_0.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="12_visualizing_cross_sections_
↪in_pyvista.zip", dirpath=file_path)
```

```
Downloading file '12_visualizing_cross_sections_in_pyvista.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F/12_visualizing_cross_sections_in_
↪pyvista.zip' to 'C:\Users\ale93371\Documents\gemgis1\docs\getting_started\tutorial\
↪data\12_visualizing_cross_sections_in_pyvista'.
```

6.13.3 Loading Data

The traces were saved as shape file and are now loaded as GeoDataFrame using GeoPandas.

[3]:
```
import geopandas as gpd

traces = gpd.read_file(file_path + 'traces.shp')
traces
```

[3]:
```
   id  zmax  zmin      name                                              geometry
0 NaN   500 -6000  Muenster  LINESTRING (32386148.890 5763304.720, 32393549...
1 NaN   500 -2000    Rheine  LINESTRING (32402275.136 5761828.501, 32431165...
```

6.13.4 Create Mesh for one cross section

A mesh can be created based on the LineStrings and the provided vertical extent using the function `create_mesh_cross_section(..)`.

[30]:
```
mesh = gg.visualization.create_mesh_from_cross_section(linestring=traces.loc[0].geometry,
                                                       zmin=traces.loc[0]['zmin'],
                                                       zmax=traces.loc[0]['zmax'])

mesh
```

[30]:
```
PolyData (0x2cb97873ee0)
  N Cells:    20
  N Points:   22
  N Strips:   0
  X Bounds:   3.239e+07, 3.242e+07
  Y Bounds:   5.717e+06, 5.763e+06
  Z Bounds:   -6.000e+03, 5.000e+02
  N Arrays:   0
```

[31]:
```
type(mesh)
```

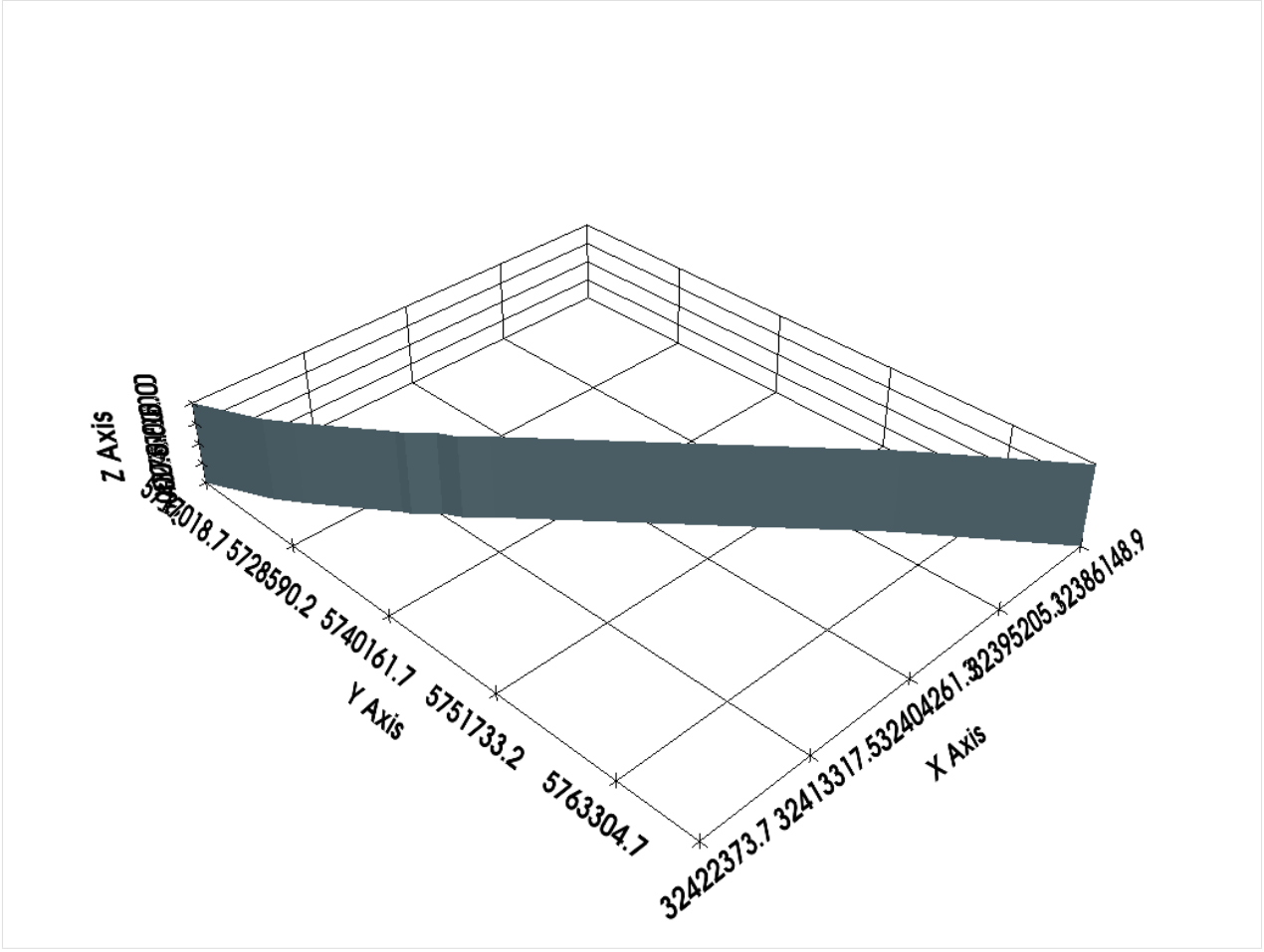
[31]:
```
pyvista.core.pointset.PolyData
```

Plotting Mesh

The mesh can now be plotted using PyVista. A new PyVista plotter is created. The cross section is correctly placed in space. However, no cross section data is yet shown on the mesh.

[32]:
```
import pyvista as pv

p = pv.Plotter()

p.add_mesh(mesh)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\pyvista\jupyter\
↪notebook.py:58: UserWarning: Failed to use notebook backend:

No module named 'trame'

Falling back to a static output.
  warnings.warn(
```



Adding texture to mesh

A texture can be applied to the mesh to display the cross section data. In this case, the image data of the cross section is used to drape it over the mesh.

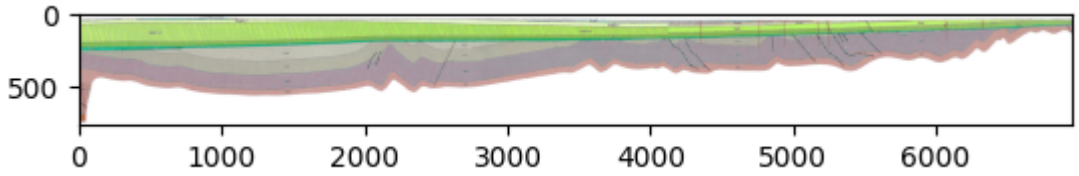
[33]:
```
import rasterio

texture = rasterio.open(file_path + 'profile1.png')
```

```
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\rasterio\__init__.py:
↪304: NotGeoreferencedWarning: Dataset has no geotransform, gcps, or rpcs. The identity␣
↪matrix will be returned.
  dataset = DatasetReader(path, driver=driver, sharing=sharing, **kwargs)
```

Plotting texture

The loaded image file contains the data of the cross section.

[34]:
```
from rasterio.plot import show

show(texture);
```

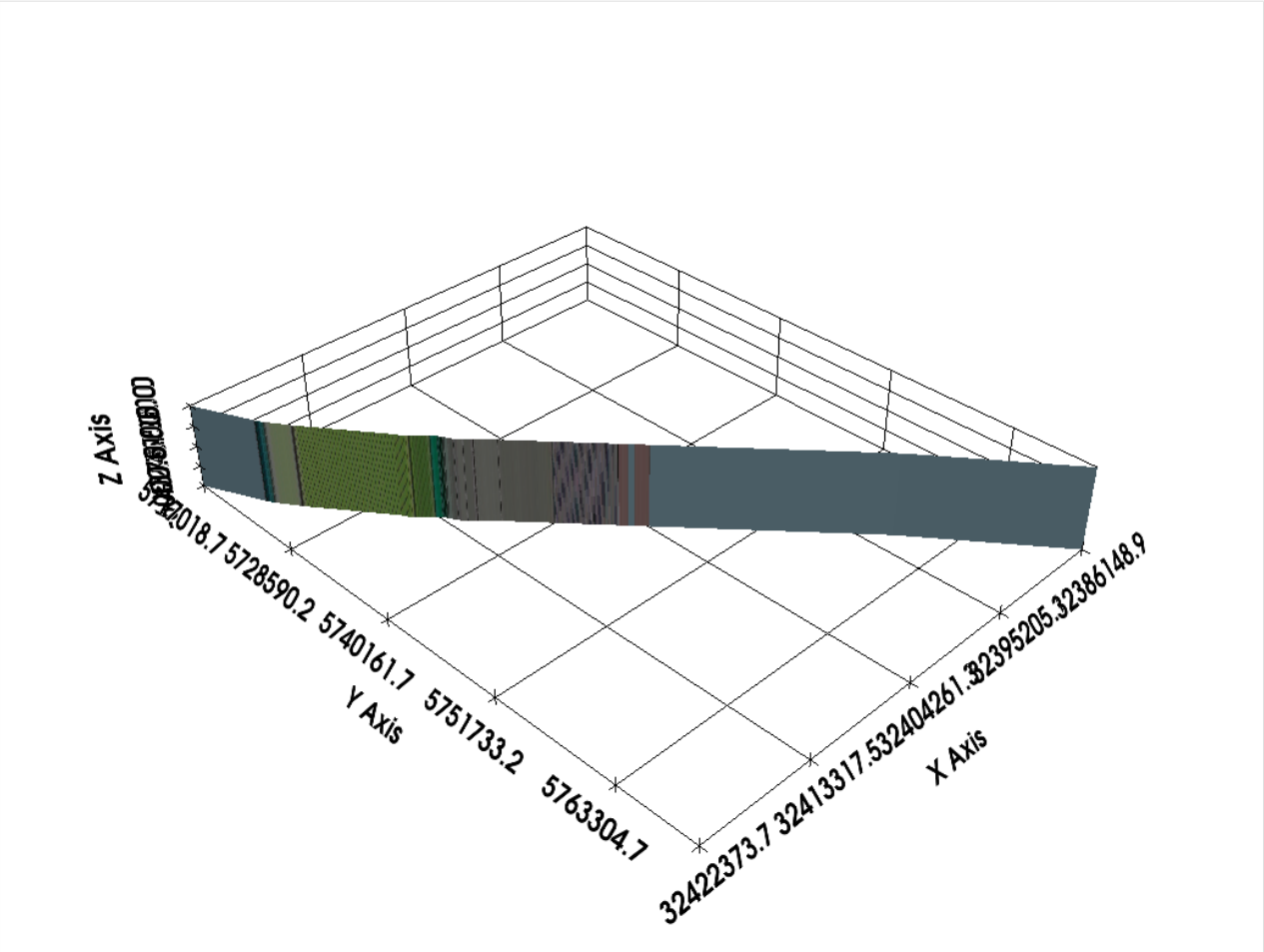


Adding texture to mesh

A PyVista texture will be created and passed to the `add_mesh(...)` function as shown below to drape the image data over the mesh.

[42]:
```
tex = pv.read_texture(file_path + 'profile1.png')

mesh.texture_map_to_plane(use_bounds=False,inplace=True)

p = pv.Plotter()
p.add_mesh(mesh)
p.add_mesh(mesh, texture=tex)


p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\pyvista\jupyter\
↪notebook.py:58: UserWarning: Failed to use notebook backend:

No module named 'trame'

Falling back to a static output.
  warnings.warn(
```



6.13.5 Creating Meshes for multiple cross sections

The functionality for one cross section can be extended to multiple cross sections. In this case, we are passing the entire GeoDataFrame containing the traces to the function `create_meshes_from_cross_sections(..)`. The result is a list of PyVista PolyData datasets.

[36]:
```
meshes = gg.visualization.create_meshes_from_cross_sections(gdf=traces)
meshes
```

[36]:
```
[PolyData (0x2cc1fb393c0)
  N Cells:    20
  N Points:   22
  N Strips:   0
  X Bounds:   3.239e+07, 3.242e+07
  Y Bounds:   5.717e+06, 5.763e+06
  Z Bounds:   -6.000e+03, 5.000e+02
  N Arrays:   0,
 PolyData (0x2cc1fb39540)
  N Cells:    2
  N Points:   4
  N Strips:   0
  X Bounds:   3.240e+07, 3.243e+07
```

(continues on next page)

(continued from previous page)

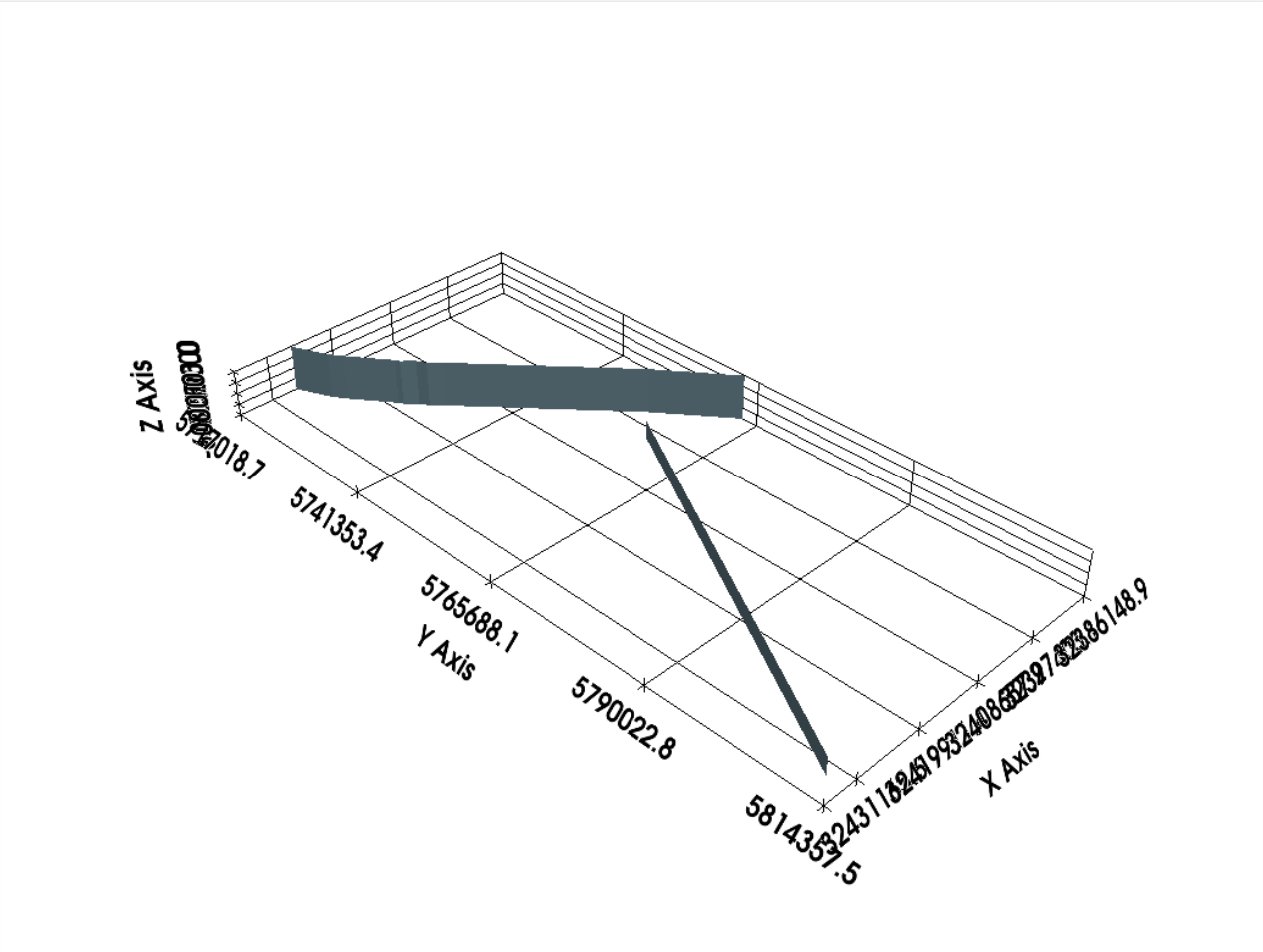
```
   Y Bounds:   5.762e+06, 5.814e+06
   Z Bounds:   -2.000e+03, 5.000e+02
   N Arrays:   0]
```

Plotting Meshes

The meshes can now be plotted using PyVista. A new PyVista plotter is created. The cross sections are correctly placed in space.

[37]:
```
import pyvista as pv

p = pv.Plotter()

for i in range(len(meshes)):
    p.add_mesh(meshes[i])

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\pyvista\jupyter\
↪notebook.py:58: UserWarning: Failed to use notebook backend:

No module named 'trame'

Falling back to a static output.
  warnings.warn(
```



Adding Texture to Meshes

As for the previous example, the textures are loaded via the path of the image files. Therefore, the function `create_file_paths` is used to create a list of file paths.

NB: The order of the traces in the passed GeoDataFrame must be identical to the order of the loaded images!

[38]:
```
source_files = gg.raster.create_filepaths(dirpath = file_path, search_criteria='profile*.
↪png')
source_files
```

[38]:
```
['C:\\Users\\ale93371\\Documents\\gemgis1\\docs\\getting_started\\tutorial\\data\\12_
↪visualizing_cross_sections_in_pyvista\\profile1.png',
 'C:\\Users\\ale93371\\Documents\\gemgis1\\docs\\getting_started\\tutorial\\data\\12_
↪visualizing_cross_sections_in_pyvista\\profile2.png']
```

A list of textures is then created from each source image file.

[39]:
```
textures = [pv.Texture(file) for file in source_files]
textures
```

[39]:
```
[Texture (0x2cb84f9da80)
  Components:   4
```

(continues on next page)

(continued from previous page)

```
  Cube Map:     False
  Dimensions:   6952, 771,
Texture (0x2cc1fb3a2c0)
  Components:   4
  Cube Map:     False
  Dimensions:   7047, 293]
```

Plotting Meshes with Textures

The cross sections are finally plotted at their true locations in space with the textures draped over them.

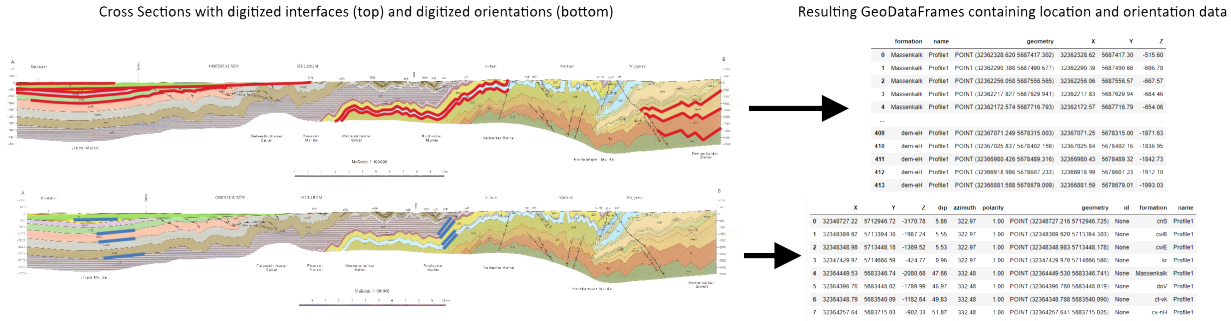
[40]:
```
import pyvista as pv

p = pv.Plotter()

for i in range(len(meshes)):
    p.add_mesh(meshes[i], texture=textures[i])

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[40], line 6
      3 p = pv.Plotter()
      5 for i in range(len(meshes)):
----> 6     p.add_mesh(meshes[i], texture=textures[i])
      8 p.show_grid(color='black')
      9 p.set_background(color='white')

File ~\Anaconda3\envs\pygeomechanical\lib\site-packages\pyvista\plotting\plotter.py:3450,
↪ in BasePlotter.add_mesh(self, mesh, color, style, scalars, clim, show_edges, edge_
↪color, point_size, line_width, opacity, flip_scalars, lighting, n_colors, interpolate_
↪before_map, cmap, label, reset_camera, scalar_bar_args, show_scalar_bar, multi_colors,␣
↪name, texture, render_points_as_spheres, render_lines_as_tubes, smooth_shading, split_
↪sharp_edges, ambient, diffuse, specular, specular_power, nan_color, nan_opacity,␣
↪culling, rgb, categories, silhouette, use_transparency, below_color, above_color,␣
↪annotations, pickable, preference, log_scale, pbr, metallic, roughness, render,␣
↪component, emissive, copy_mesh, backface_params, show_vertices, **kwargs)
   3448     raise TypeError(f'Invalid texture type ({type(texture)})')
   3449 if mesh.GetPointData().GetTCoords() is None:
-> 3450     raise ValueError(
   3451         'Input mesh does not have texture coordinates to support the texture.'
   3452     )
   3453 actor.texture = texture
   3454 # Set color to white by default when using a texture

ValueError: Input mesh does not have texture coordinates to support the texture.
```

[ ]:

6.14 13 Extracting Interface Points and Orientations from Geological Cross Sections

Geologists often work with geological maps. In addition, geological cross sections are provided with these maps to illustrate an image of the subsurface. The displayed lithological boundaries and structures can be used for geological modeling. Therefore, the interfaces and orientations of the layers have to be extracted.



The data used for GemGIS is obtained from *OpenDataNRW*. It will be used under Datenlizenz Deutschland – Namensnennung – Version 2.0 (https://www.govdata.de/dl-de/by-2-0) with © Geowissenschaftliche Daten: Analoges Kartenwerk der Geologischen Karte von Nordrhein-Westfalen 1:100.000 (2020).

6.14.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/13_extracting_interfaces_orientations_from_cross_sections/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="13_extracting_interfaces_
↪orientations_from_cross_sections.zip", dirpath=file_path)
```

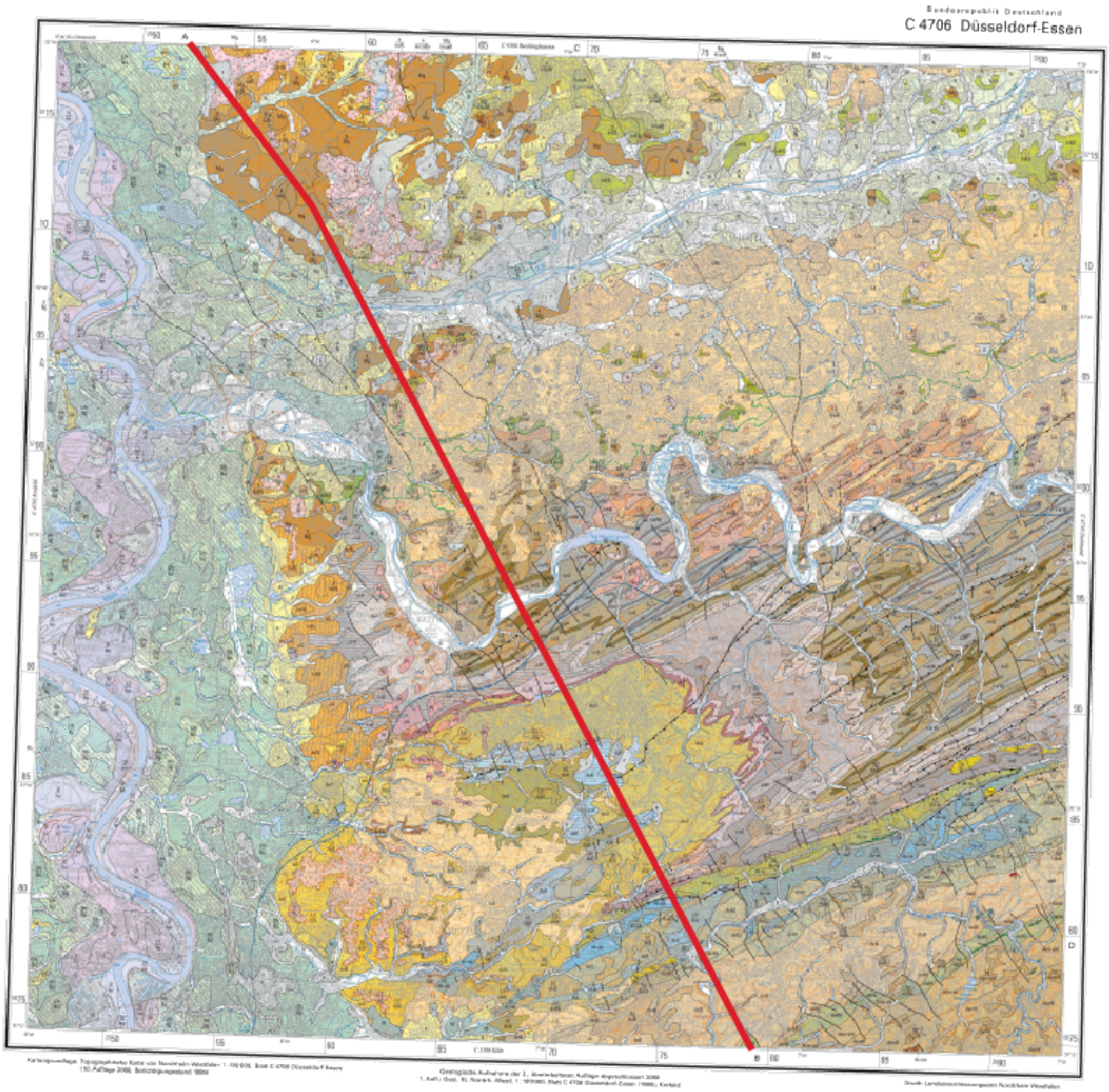
```
Downloading file '13_extracting_interfaces_orientations_from_cross_sections.zip' from
↪'https://rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F13_extracting_
↪interfaces_orientations_from_cross_sections.zip' to 'C:\Users\ale93371\Documents\
↪gemgis\docs\getting_started\tutorial\data\13_extracting_interfaces_orientations_from_
↪cross_sections'.
```

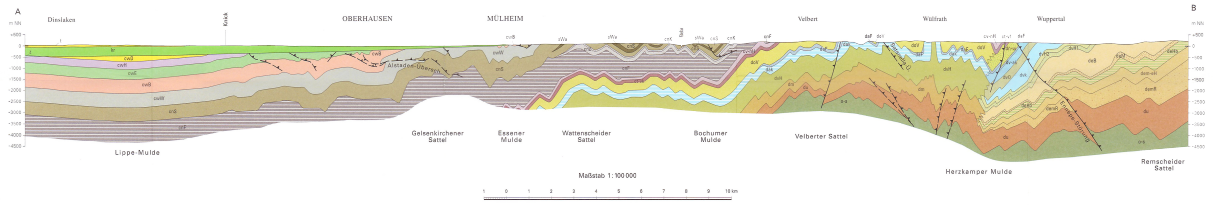
6.14.2 Extracting Interfaces from Cross Sections

Interfaces can be extracted from georeferenced maps and cross sections by digitizing the layer data as shape files. The following illustrates the data preparation steps and the extraction of coordinates in GemGIS.

Data Preparation

For the extraction of interfaces and orientation, the same principles as for maps are used. This is, that the interfaces and orientations are digitized in a GIS environment using shape files to store the layer data. The cross section used here is obtained from *OpenGeoData.NRW*. The cross section is part of the 'Analoges Kartenwerk der Geologischen Karte von Nordrhein Westfalen 1:100 000' and in particular of map sheet GK100-C4706-Düsseldorf-Essen. The data is used under Datenlizenz Deutschland Namensnennung 2.0 (https://www.govdata.de/dl-de/by-2-0).



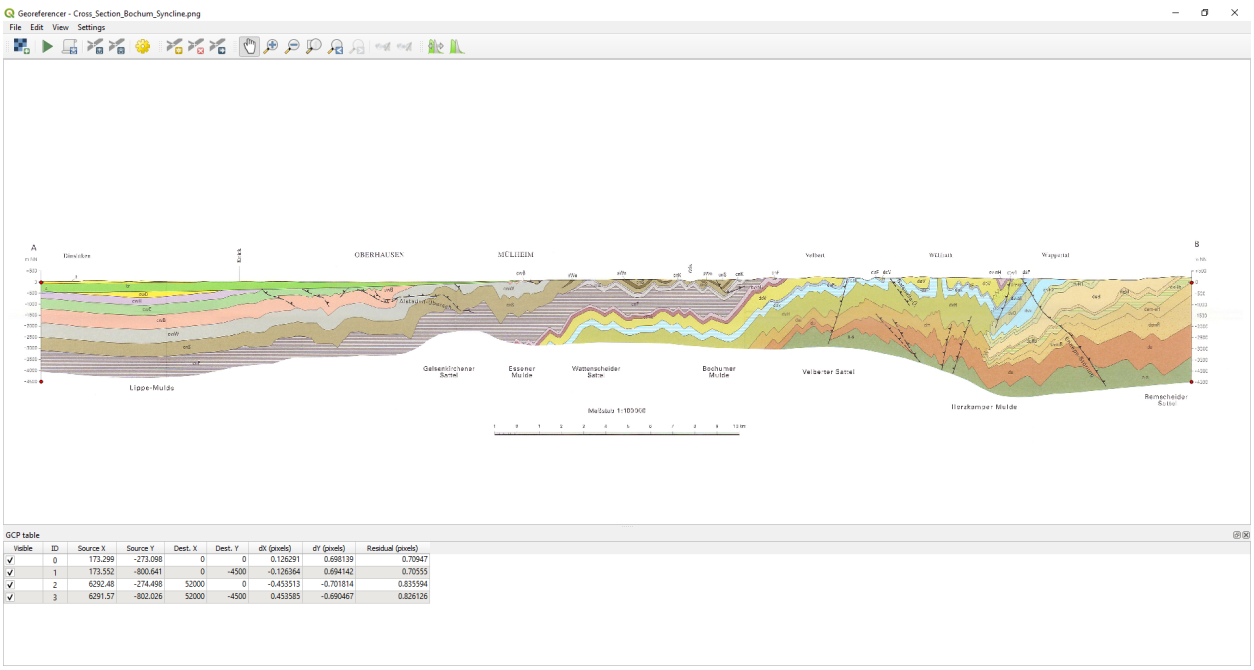


Georeference Cross Section

The first step is to georeference the cross section. A tutorial on how to georeference maps is provided on this documentation page. For the georeferencing here, we are taking four points on the two axes of the cross section. It is important that the coordinates on the left side mark the origin and are defined by X = 0. The coordinates will be:

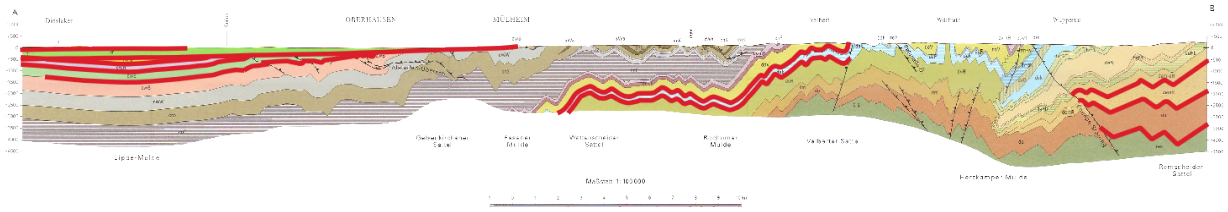
- 0,0
- 0,-4500
- 52000, 0
- 52000, -4500

NB: Ideally, you would look up the exact length of the section which is 51819.98 m but 52000 m is close enough for this demonstration.



Digitizing the layer data

In order to digitize the layer data, a new line shape file is created and the layer boundaries are digitized. The image below shows some of the digitized layers in red. This can now be read in as GeoDataFrames for further processing with GemGIS.



Loading the data

Next to the digitized interfaces, we also need to load the trace of the profile. This is shown in one of the figures above and was also saved as shape file for further processing. In addition, we are loading the raster of the cross section and the geological map for visualization purposes.

[3]:
```
import geopandas as gpd
import rasterio

interfaces = gpd.read_file(file_path + 'interfaces.shp')

section = rasterio.open(file_path + 'cross_section.tif')

gmap = rasterio.open(file_path + 'C4706_Duesseldorf_Essen_2Aufl_prs_utm32n.jpg')

trace = gpd.read_file(file_path + 'trace.shp')
```

[4]:
```
interfaces.head()
```

[4]:

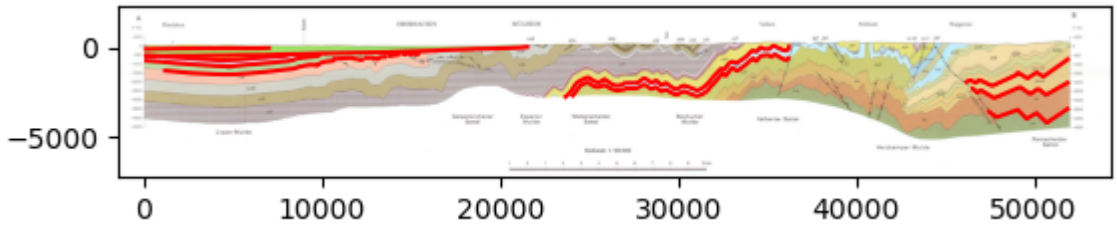
	id	formation	geometry
0	None	Massenkalk	LINESTRING (36281.025 -515.603, 36198.287 -606...
1	None	doV	LINESTRING (36262.451 161.499, 36221.926 -3.97...
2	None	t	LINESTRING (-11.512 -82.198, 1761.939 -48.737,...
3	None	kr	LINESTRING (4.757 -417.917, 561.958 -449.859, ...
4	None	z	LINESTRING (8.306 -492.448, 711.018 -478.251)

Plotting the Input data

The raster containing the image of the cross section and the interfaces can be plotted together.

[5]:
```
import matplotlib.pyplot as plt
from rasterio.plot import reshape_as_image

fig, ax = plt.subplots(1,1)
ax.imshow(reshape_as_image(section.read()), extent=[section.bounds[0],section.bounds[2],
↪section.bounds[1],section.bounds[3]])
interfaces.plot(ax=ax, aspect='equal', color='red')
```

[5]:
```
<AxesSubplot: >
```



The trace of the profile can also be plotted. The GeoDataFrame has to be converted to the matching Coordinate Reference System before plotting. In addition, the geological map is shown to visualize the location of the section on the map.

[6]:
```
trace
```

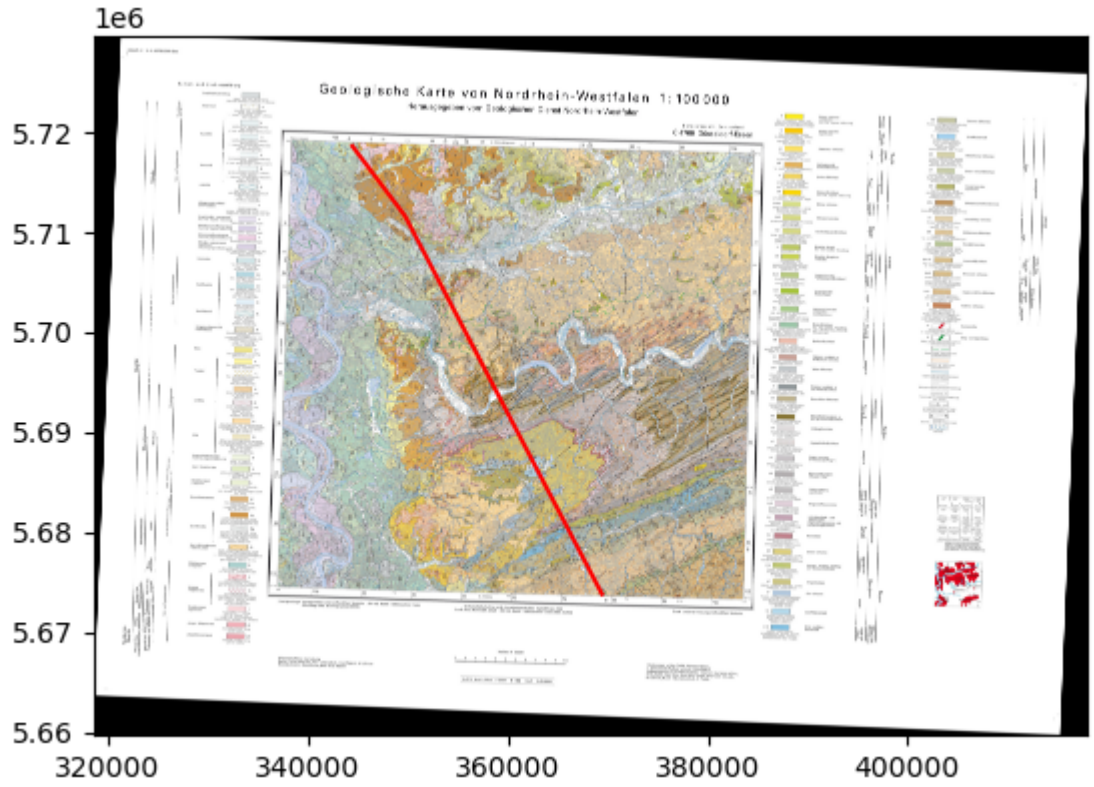
[6]:
```
     id                                               geometry
0  None  LINESTRING (32344311.102 5718801.515, 32349696...
```

[7]:
```
trace.length
```

[7]:
```
0   51819.98
dtype: float64
```

[8]:
```
fig, ax = plt.subplots(1,1)

ax.imshow(reshape_as_image(gmap.read()), extent=[gmap.bounds[0],gmap.bounds[2],gmap.
↪bounds[1],gmap.bounds[3]])
trace.to_crs('EPSG:25832').plot(ax=ax, aspect='equal', color='red')
```

[8]:
```
<AxesSubplot: >
```



Extracting the real world coordinates from the cross section

The extraction of the real world coordinates is performed in several steps:

- Extracting the striking angle of the profile
- Calculating real word coordinates for one LineString on the Cross Section
- Calculating real word coordinates for LineStrings on the Cross Section
- Extracting Coordinates from the cross section
- Extracting XYZ from the cross section

Extracting Striking angle of the profile

Here, the strike angles of a LineString with multiple vertices is shown. For straight profiles with only two vertices, the function `calculate_strike_direction_straight_linestring(..)` can be used. The calculation shows that the first part of the profile is striking towards 143 degrees to the SE and the second part with 152 degrees to the SE. This can be confirmed when looking at the map.

[9]:
```
angles = gg.vector.calculate_strike_direction_bent_linestring(linestring=trace.loc[0].
↪geometry)
angles
```

[9]:
```
[142.97370864998288, 152.47884465731576]
```

Calculating real word coordinates for one LineString on the Cross Section

The next step is to calculate the real world coordinates of one LineString located on the cross section. This function uses the previously calculated striking angle of each section of the line string and the distance of each digitized vertex to the origin to calculate the real world coordinates. This function is the extension of `calculate_coordinates_for_point_on_cross_section(..)`. The results are stored as Shapely Point objects.

[10]:
```
points = gg.vector.calculate_coordinates_for_linestring_on_cross_
↪sections(linestring=trace.loc[0].geometry,

↪interfaces=interfaces.loc[0].geometry)
points[:5]
```

[10]:
```
[<POINT (32362328.62 5687417.302)>,
 <POINT (32362290.388 5687490.677)>,
 <POINT (32362256.058 5687556.565)>,
 <POINT (32362217.827 5687629.941)>,
 <POINT (32362172.574 5687716.793)>]
```

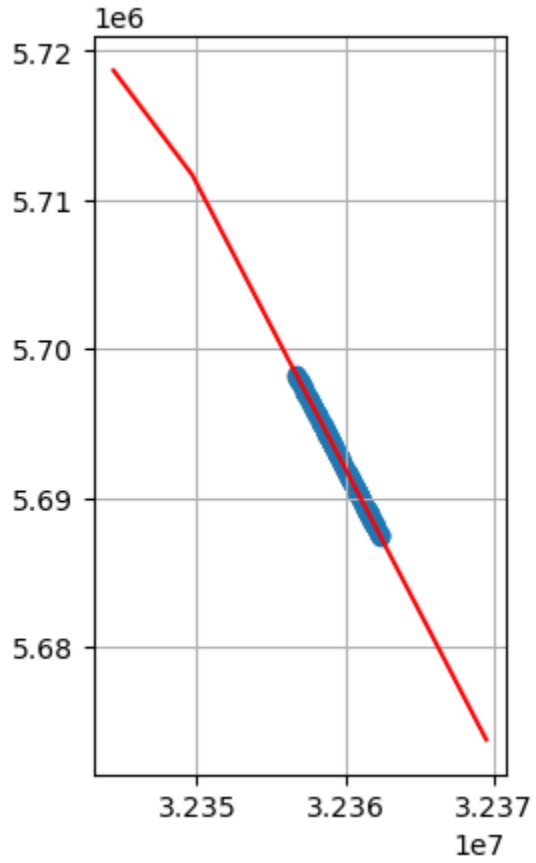
Plotting the points

The points can be converted to a GeoDataFrame and can be plotted. It can be seen that the extracted points are located exactly on the profile line/trace.

[11]:
```
points_gdf = gpd.GeoDataFrame(geometry=points)
points_gdf.head()
```

[11]:
```
                          geometry
0  POINT (32362328.620 5687417.302)
1  POINT (32362290.388 5687490.677)
2  POINT (32362256.058 5687556.565)
3  POINT (32362217.827 5687629.941)
4  POINT (32362172.574 5687716.793)
```

[12]:
```
fig, ax = plt.subplots(1,1)

trace.plot(ax=ax, aspect='equal', color='red')

points_gdf.plot(ax=ax,aspect='equal')
plt.grid()
```



Calculating real word coordinates for LineStrings on the Cross Section

The previous function can be extended to multiple LineStrings by providing a list of LineStrings.

[13]:
```
points = gg.vector.calculate_coordinates_for_linestrings_on_cross_
↪sections(linestring=trace.loc[0].geometry,
                                                                    linestring_
↪interfaces_list=interfaces.geometry.to_list())
points[:5]
```

[13]:
```
[<POINT (32362328.62 5687417.302)>,
 <POINT (32362290.388 5687490.677)>,
 <POINT (32362256.058 5687556.565)>,
 <POINT (32362217.827 5687629.941)>,
 <POINT (32362172.574 5687716.793)>]
```

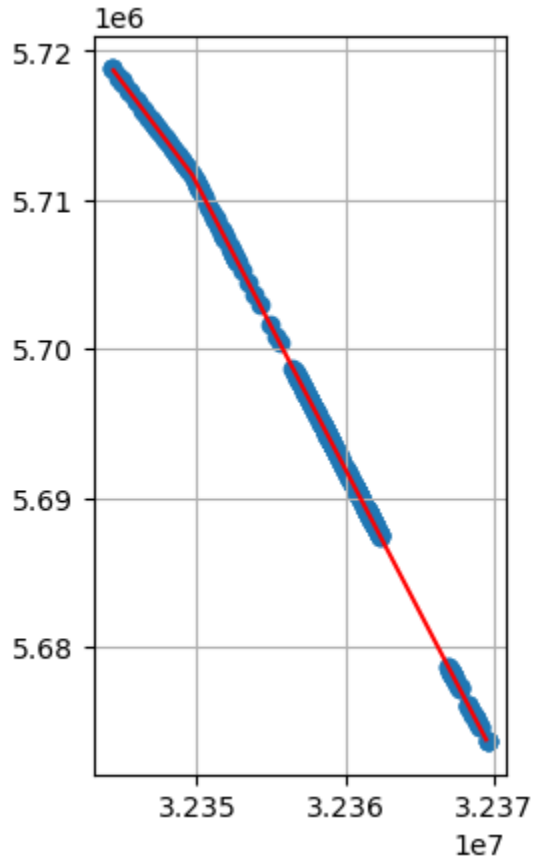
Plotting the points

The points can be converted to a GeoDataFrame and plotted. It can be seen that the extracted points are located exactly on the profile line/trace.

[14]:
```
points_gdf = gpd.GeoDataFrame(geometry=points)
points_gdf.head()
```

[14]:
```
                          geometry
0  POINT (32362328.620 5687417.302)
1  POINT (32362290.388 5687490.677)
2  POINT (32362256.058 5687556.565)
3  POINT (32362217.827 5687629.941)
4  POINT (32362172.574 5687716.793)
```

[15]:
```
fig, ax = plt.subplots(1,1)

trace.plot(ax=ax, aspect='equal', color='red')

points_gdf.plot(ax=ax,aspect='equal')
plt.grid()
```



Extracting Coordinates from the cross section

The next step is to not only extract the points in X and Y direction but also the Z component and any additional column in the original interfaces GeoDataFrame.

[16]:
```
gdf = gg.vector.extract_interfaces_coordinates_from_cross_section(linestring=trace.
↪loc[0].geometry,
                                                                  interfaces_
↪gdf=interfaces)
gdf
```

[16]:
```
      formation                          geometry           X          Y  \
0    Massenkalk  POINT (32362328.620 5687417.302) 32362328.62 5687417.30
1    Massenkalk  POINT (32362290.388 5687490.677) 32362290.39 5687490.68
2    Massenkalk  POINT (32362256.058 5687556.565) 32362256.06 5687556.57
3    Massenkalk  POINT (32362217.827 5687629.941) 32362217.83 5687629.94
4    Massenkalk  POINT (32362172.574 5687716.793) 32362172.57 5687716.79
..          ...                                ...         ...        ...
409      dem-eH  POINT (32367071.249 5678315.003) 32367071.25 5678315.00
410      dem-eH  POINT (32367025.837 5678402.159) 32367025.84 5678402.16
411      dem-eH  POINT (32366980.426 5678489.316) 32366980.43 5678489.32
412      dem-eH  POINT (32366918.986 5678607.233) 32366918.99 5678607.23
413      dem-eH  POINT (32366881.588 5678679.009) 32366881.59 5678679.01

           Z
0    -515.60
1    -606.78
2    -667.57
3    -684.46
4    -654.06
..       ...
409 -1871.63
410 -1836.95
411 -1842.73
412 -1912.10
413 -1993.03

[414 rows x 5 columns]
```

Extracting XYZ from the cross section

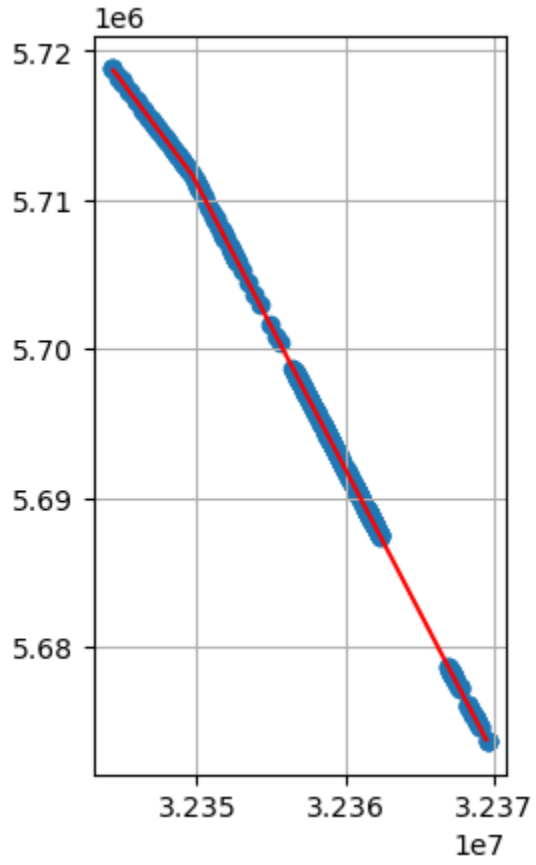
All the above steps are collected in a final function to extract the real world coordinates from the cross section. An additional column for the name of the profile needs to be provided so that coordinates for multiple profiles can be calculated.

[17]:
```
trace['name'] = 'Profile1'
interfaces['name'] = 'Profile1'
```

[18]:
```
gdf = gg.vector.extract_xyz_from_cross_sections(profile_gdf=trace,
                                                interfaces_gdf=interfaces)
gdf
```

[18]:
```
      formation      name                          geometry           X  \
0    Massenkalk  Profile1  POINT (32362328.620 5687417.302) 32362328.62
1    Massenkalk  Profile1  POINT (32362290.388 5687490.677) 32362290.39
2    Massenkalk  Profile1  POINT (32362256.058 5687556.565) 32362256.06
3    Massenkalk  Profile1  POINT (32362217.827 5687629.941) 32362217.83
4    Massenkalk  Profile1  POINT (32362172.574 5687716.793) 32362172.57
..          ...       ...                               ...         ...
409      dem-eH  Profile1  POINT (32367071.249 5678315.003) 32367071.25
410      dem-eH  Profile1  POINT (32367025.837 5678402.159) 32367025.84
411      dem-eH  Profile1  POINT (32366980.426 5678489.316) 32366980.43
412      dem-eH  Profile1  POINT (32366918.986 5678607.233) 32366918.99
413      dem-eH  Profile1  POINT (32366881.588 5678679.009) 32366881.59

             Y        Z
0   5687417.30  -515.60
1   5687490.68  -606.78
2   5687556.57  -667.57
3   5687629.94  -684.46
4   5687716.79  -654.06
..         ...      ...
409 5678315.00 -1871.63
410 5678402.16 -1836.95
411 5678489.32 -1842.73
412 5678607.23 -1912.10
413 5678679.01 -1993.03

[414 rows x 6 columns]
```

[19]:
```
fig, ax = plt.subplots(1,1)

trace.plot(ax=ax, aspect='equal', color='red')

gdf.plot(ax=ax,aspect='equal')
plt.grid()
```



6.14.3 Extracting Orientations from Cross Sections

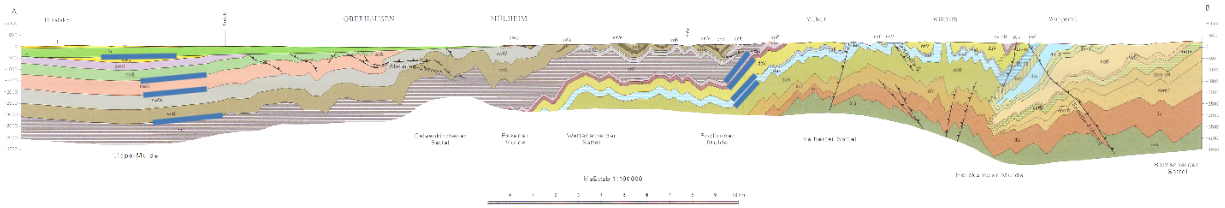
Orientations can also be extracted from cross sections using GemGIS. However, be aware that the azimuth or dipping orientation will always be the apparent angle in line with the strike of the profile.

Data Preparation

The same georeferenced section as before is needed to prepare the data.

Digitizing the Data

Orientations display the dipping/slope of a layer. For GemGIS, a straight line consisting of a start and end point needs to be digitized following the dipping of the layers as shown below.



Loading the data

Next to the digitized interfaces, we also need to load the trace of the profile. This is shown in one of the figures above and was also saved as shape file for further processing. In addition, we are loading the raster of the cross section and the geological map for visualization purposes.

[20]:
```
import geopandas as gpd
import rasterio
import gemgis as gg

orientations = gpd.read_file(file_path + 'orientations.shp')

section = rasterio.open(file_path + 'cross_section.tif')

gmap = rasterio.open(file_path + 'C4706_Duesseldorf_Essen_2Aufl_prs_utm32n.jpg')

trace = gpd.read_file(file_path + 'trace.shp')
```

[21]:
```
orientations.head()
```

[21]:

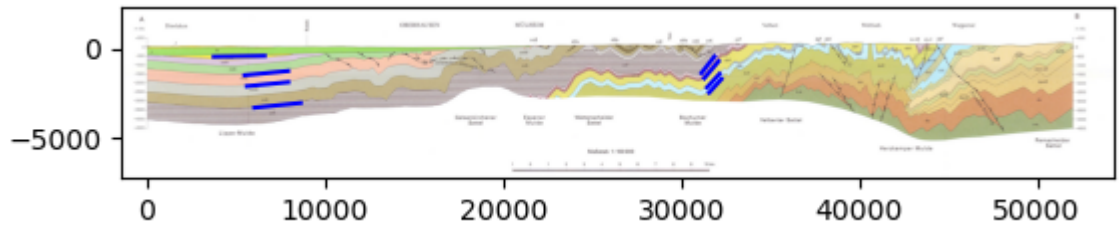
	id	formation	geometry
0	None	cnS	LINESTRING (8750.658 -3025.432, 5916.398 -3316...
1	None	cwB	LINESTRING (8055.070 -1862.659, 5490.740 -2111...
2	None	cwE	LINESTRING (8044.688 -1239.745, 5366.157 -1499...
3	None	kr	LINESTRING (6726.187 -398.810, 3632.379 -450.720)
4	None	Massenkalk	LINESTRING (32353.917 -1613.493, 31502.601 -25...

Plotting the Input data

The raster containing the image of the cross section and the interfaces can be plotted together.

[22]:
```
import matplotlib.pyplot as plt
from rasterio.plot import reshape_as_image

fig, ax = plt.subplots(1,1)
ax.imshow(reshape_as_image(section.read()), extent=[section.bounds[0],section.bounds[2],
↪section.bounds[1],section.bounds[3]])
orientations.plot(ax=ax, aspect='equal', color='blue')
```

[22]:
```
<AxesSubplot: >
```



The trace of the profile can also be plotted. The GeoDataFrame has to be converted to the matching Coordinate Reference System before plotting. In addition, the geological map is shown to visualize the location of the section on the map.

[23]:
```
trace
```

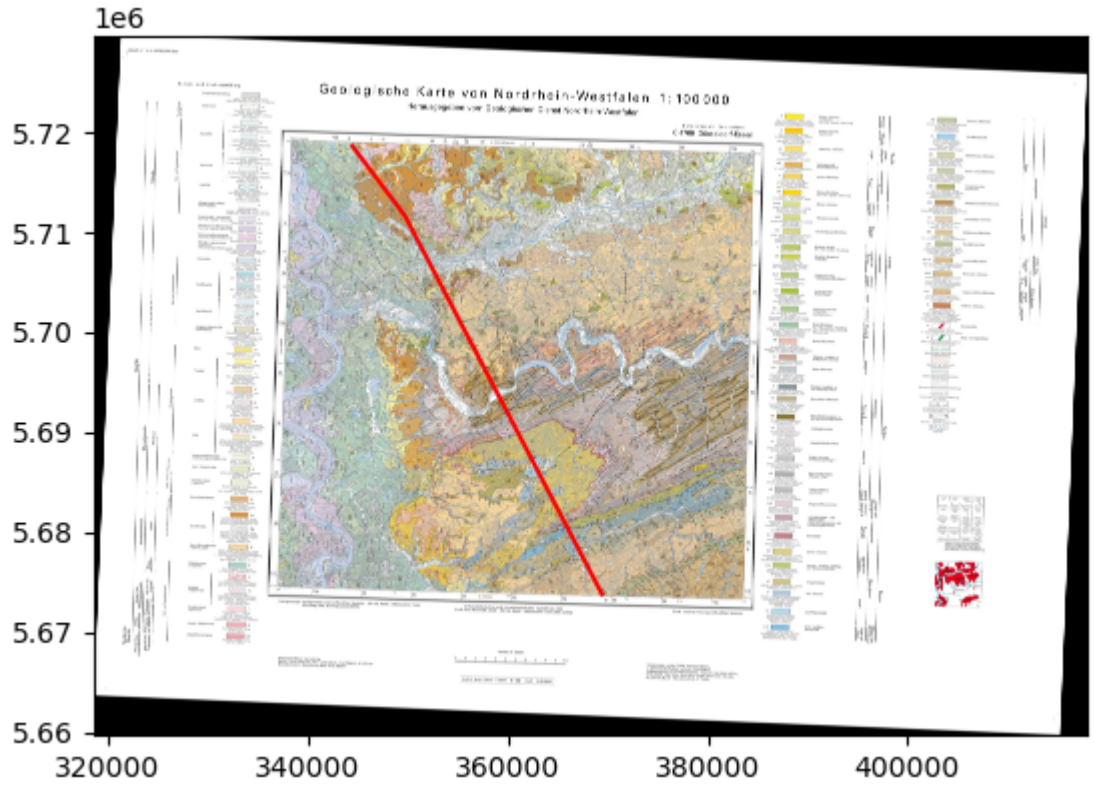
[23]:
```
     id                                                geometry
0  None  LINESTRING (32344311.102 5718801.515, 32349696...
```

[24]:
```
trace.length
```

[24]:
```
0   51819.98
dtype: float64
```

[25]:
```
fig, ax = plt.subplots(1,1)

ax.imshow(reshape_as_image(gmap.read()), extent=[gmap.bounds[0],gmap.bounds[2],gmap.
↪bounds[1],gmap.bounds[3]])
trace.to_crs('EPSG:25832').plot(ax=ax, aspect='equal', color='red')
```

[25]:
```
<AxesSubplot: >
```



Extracting the real world coordinates from the cross section

The extraction of the real world coordinates is performed in several steps:

- Extracting the striking angle of the profile
- Calculating the midpoint of one LineString
- Calculating the midpoints of LineStrings
- Calculating orientation from cross section
- Calculating orientations from cross section

Extracting Striking angle of the profile

Here, the strike angles of a LineString with multiple vertices is shown. For straight profiles with only two vertices, the function `calculate_strike_direction_straight_linestring(..)` can be used. The calculation shows that the first part of the profile is striking towards 143 degrees to the SE and the second part with 152 degrees to the SE. This can be confirmed when looking at the map.

[26]:
```
angles = gg.vector.calculate_strike_direction_bent_linestring(linestring=trace.loc[0].
↪geometry)
angles
```

[26]:
```
[142.97370864998288, 152.47884465731576]
```

Calculating the midpoint of one LineString

The location for the orientation after its extraction will be the midpoint of the digitized LineString.

[27]:
```
orientations.loc[0].geometry.wkt
```

[27]:
```
'LINESTRING (8750.657889753686 -3025.432084887312, 5916.398060484471 -3316.125400709796)'
```

[28]:
```
midpoint = gg.vector.calculate_midpoint_linestring(linestring=orientations.loc[0].
↪geometry)
midpoint.wkt
```

[28]:
```
'POINT (7333.527975119078 -3170.778742798554)'
```

Calculating the midpoints of LineStrings.

The previous function can be extended to work for a list of LineStrings or a GeoDataFrame containing LineStrings. In this case, the orientations GeoDataFrame.

[29]:
```
midpoints = gg.vector.calculate_midpoints_linestrings(linestring_gdf=orientations)
midpoints
```

[29]:
```
[<POINT (7333.528 -3170.779)>,
 <POINT (6772.905 -1987.242)>,
 <POINT (6705.423 -1369.518)>,
 <POINT (5179.283 -424.765)>,
 <POINT (31928.259 -2080.679)>,
 <POINT (31814.058 -1789.985)>,
```

(continues on next page)

(continued from previous page)

```
 <POINT (31710.239 -1182.644)>,
 <POINT (31512.983 -902.333)>]
```

Calculating Orientation from Cross Section

In order to calculate the orientation from the cross section, the real world coordinates of the midpoint of the respective LineString and its dip will be calculated. In addition, the strike of the profile will be calculated from which the azimuth of the orientation value will be derived from. As we have a bent cross section, `calculate_orientation_from_cross_section(..)` is skipped and `calculate_orientation_from_bent_cross_section(..)` will be used right away. The resulting list contains the midpoint coordinates, the depth, dip, azimuth and polarity of the orientation value.

[30]:
```
orientation = gg.vector.calculate_orientation_from_bent_cross_section(profile_
↪linestring=trace.loc[0].geometry,
                                                                      orientation_
↪linestring=orientations.loc[0].geometry)
orientation
```

[30]:
```
[<POINT (32348727.216 5712946.725)>,
 -3170.778742798554,
 5.856013585428961,
 322.9737086499829,
 1]
```

Calculating Orientations from Cross Sections

The next step is to calculate the orientations of multiple LineStrings from a cross section.

[31]:
```
orientation_gdf = gg.vector.calculate_orientations_from_cross_section(profile_
↪linestring=trace.loc[0].geometry,
                                                                      orientation_
↪linestrings=orientations)
orientation_gdf
```

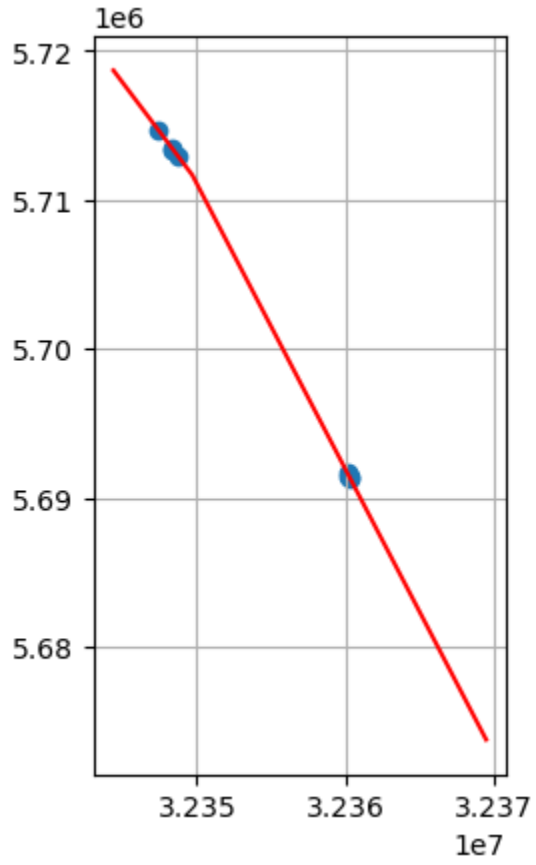
[31]:
```
            X          Y        Z   dip  azimuth  polarity  \
0 32348727.22 5712946.72 -3170.78  5.86   322.97      1.00
1 32348389.62 5713394.30 -1987.24  5.55   322.97      1.00
2 32348348.98 5713448.18 -1369.52  5.53   322.97      1.00
3 32347429.97 5714666.59  -424.77  0.96   322.97      1.00
4 32360317.31 5691277.51 -2080.68 47.66   332.48      1.00
5 32360264.54 5691378.79 -1789.99 46.97   332.48      1.00
6 32360216.57 5691470.86 -1182.64 49.83   332.48      1.00
7 32360125.42 5691645.79  -902.33 51.87   332.48      1.00

                          geometry    id   formation
0  POINT (32348727.216 5712946.725)  None         cnS
1  POINT (32348389.620 5713394.303)  None         cwB
2  POINT (32348348.983 5713448.178)  None         cwE
3  POINT (32347429.970 5714666.586)  None          kr
4  POINT (32360317.311 5691277.510)  None  Massenkalk
5  POINT (32360264.541 5691378.788)  None         doV
```

(continues on next page)

(continued from previous page)

```
6  POINT (32360216.569 5691470.859)  None       ct-vk
7  POINT (32360125.421 5691645.794)  None       cv-nH
```

[32]:
```
fig, ax = plt.subplots(1,1)

trace.plot(ax=ax, aspect='equal', color='red')

orientation_gdf.plot(ax=ax,aspect='equal')
plt.grid()
```



Extracting Orientations from Cross Sections

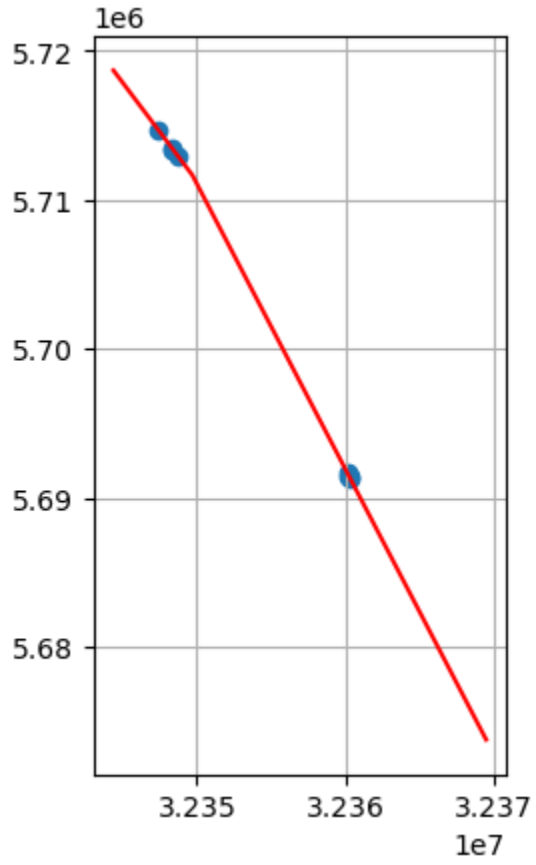
The above functions are collected in the function `extract_orientations_from_cross_section(..)` to also work for multiple profiles. Therefore, the profile name needs to be provided as well.

[33]:
```
trace['name'] = 'Profile1'
orientations['name'] = 'Profile1'
```

[34]:
```
orientation_gdf = gg.vector.extract_orientations_from_cross_sections(profile_gdf=trace,
                                                                     orientations_
↪gdf=orientations)
orientation_gdf
```

[34]:
```
            X          Y        Z   dip  azimuth  polarity  \
0 32348727.22 5712946.72 -3170.78  5.86   322.97      1.00
1 32348389.62 5713394.30 -1987.24  5.55   322.97      1.00
2 32348348.98 5713448.18 -1369.52  5.53   322.97      1.00
3 32347429.97 5714666.59  -424.77  0.96   322.97      1.00
4 32360317.31 5691277.51 -2080.68 47.66   332.48      1.00
5 32360264.54 5691378.79 -1789.99 46.97   332.48      1.00
6 32360216.57 5691470.86 -1182.64 49.83   332.48      1.00
7 32360125.42 5691645.79  -902.33 51.87   332.48      1.00

                          geometry    id   formation      name
0  POINT (32348727.216 5712946.725)  None         cnS  Profile1
1  POINT (32348389.620 5713394.303)  None         cwB  Profile1
2  POINT (32348348.983 5713448.178)  None         cwE  Profile1
3  POINT (32347429.970 5714666.586)  None          kr  Profile1
4  POINT (32360317.311 5691277.510)  None  Massenkalk  Profile1
5  POINT (32360264.541 5691378.788)  None         doV  Profile1
6  POINT (32360216.569 5691470.859)  None       ct-vk  Profile1
7  POINT (32360125.421 5691645.794)  None       cv-nH  Profile1
```

[35]:
```
fig, ax = plt.subplots(1,1)

trace.plot(ax=ax, aspect='equal', color='red')

orientation_gdf.plot(ax=ax,aspect='equal')
plt.grid()
```

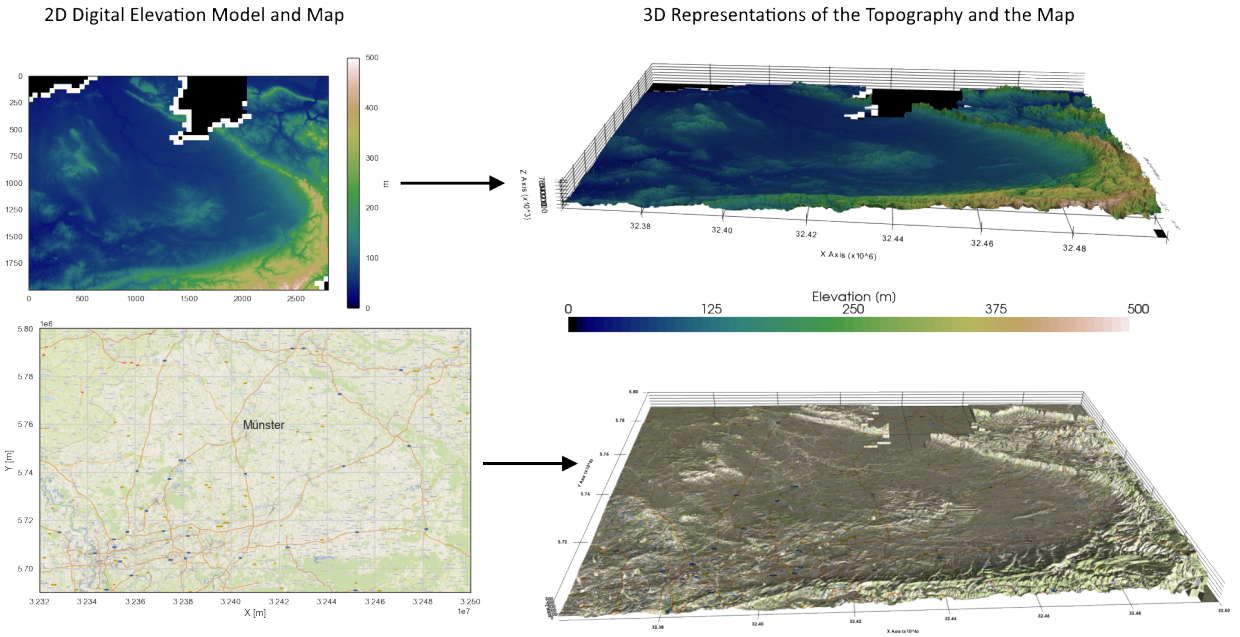


6.14.4 Plotting the data with PyVista

The extracted point and the cross sections could now be plotted in PyVista according to Tutorials 10 and 12.

6.15 14 Visualizing Topography and Maps with PyVista

The topography of an area can be visualized in 3D with PyVista and additional maps can also be draped over the topography.



6.15.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/14_visualizing_topography_and_maps_with_pyvista/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="14_visualizing_topography_and_
↪maps_with_pyvista.zip", dirpath=file_path)
```

```
Downloading file '14_visualizing_topography_and_maps_with_pyvista.zip' from 'https://
↪rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F14_visualizing_topography_and_
↪maps_with_pyvista.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\
↪tutorial\data\14_visualizing_topography_and_maps_with_pyvista'.
```

6.15.2 Loading the data

A 50 m DEM of the Münsterland Basin is loaded to illustrate the visualizing of topography in PyVista. The data will be used under Datenlizenz Deutschland – Zero – Version 2.0. It was obtained from the WCS Service https://www.wcs.nrw.de/geobasis/wcs_nw_dgm. The data will automatically be converted to a StructuredGrid with `Elevation [m]` as data array.

[3]:
```
import pyvista as pv

mesh = gg.visualization.read_raster(path=file_path + 'DEM50.tif',
```

(continues on next page)

(continued from previous page)

```
                                    nodata_val=9999.0,
                                    name='Elevation [m]')

mesh
```

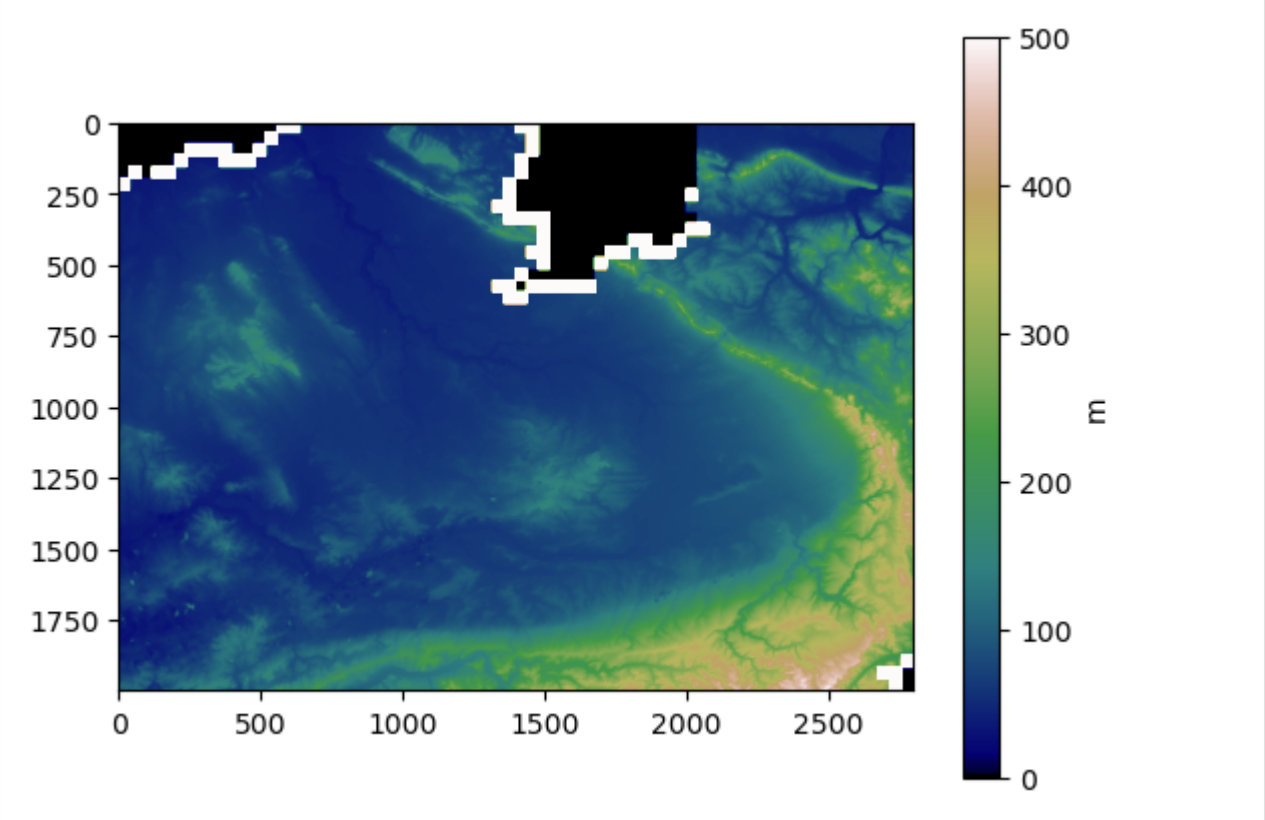
[3]:
```
StructuredGrid (0x23267807b20)
  N Cells:  5595201
  N Points: 5600000
  X Bounds: 3.236e+07, 3.250e+07
  Y Bounds: 5.700e+06, 5.800e+06
  Z Bounds: 0.000e+00, 0.000e+00
  Dimensions:       2000, 2800, 1
  N Arrays: 1
```

6.15.3 Plotting the data in 2D

[4]:
```
import rasterio

dem = rasterio.open(file_path + 'DEM50.tif')
dem.read(1)
```

[4]:
```
array([[  0.  ,   0.  ,   0.  , ...,  40.1 ,  40.09,  44.58],
       [  0.  ,   0.  ,   0.  , ...,  40.08,  40.07,  44.21],
       [  0.  ,   0.  ,   0.  , ...,  40.14,  44.21,  43.98],
       ...,
       [100.56, 102.14, 102.17, ...,   0.  ,   0.  ,   0.  ],
       [ 99.44,  99.85,  99.77, ...,   0.  ,   0.  ,   0.  ],
       [ 88.32,  91.76,  98.68, ...,   0.  ,   0.  ,   0.  ]],
      dtype=float32)
```

[5]:
```
import matplotlib.pyplot as plt

im = plt.imshow(dem.read(1), cmap='gist_earth', vmin=0, vmax=500)
cbar = plt.colorbar(im)
cbar.set_label('m')
```



6.15.4 Wrap Mesh by Scalars

The dataset's points are wrapped by a point data scalars array's values.

[6]:
```
topo = mesh.warp_by_scalar(scalars="Elevation [m]", factor=15.0)

topo
```

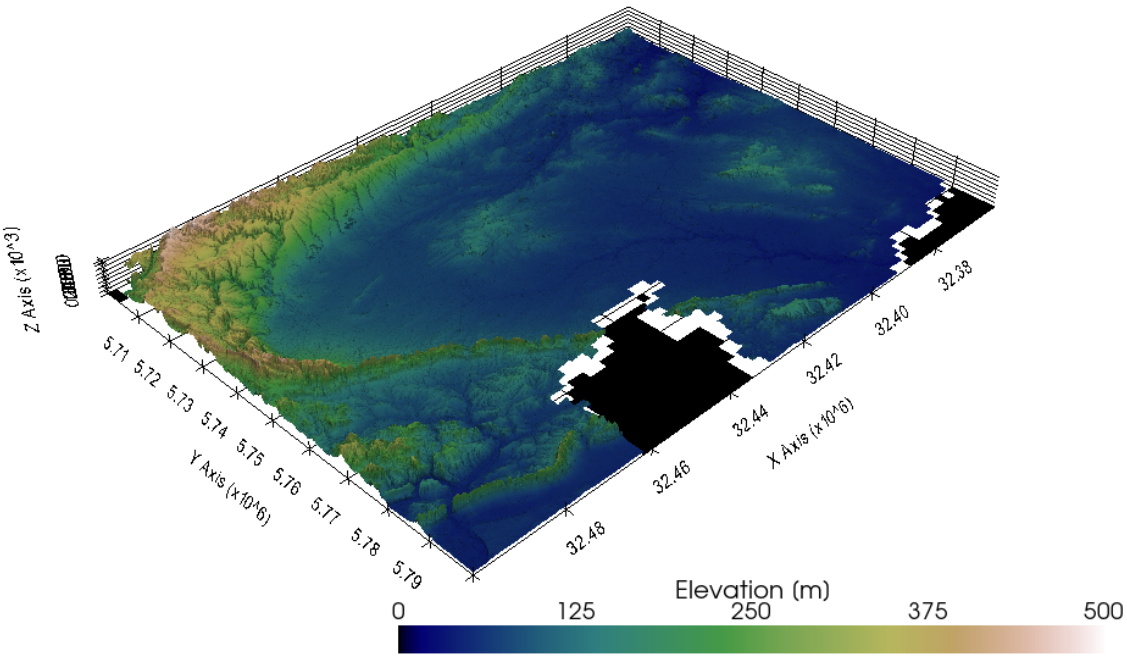
[6]:
```
StructuredGrid (0x2320cd72560)
  N Cells:  5595201
  N Points: 5600000
  X Bounds: 3.236e+07, 3.250e+07
  Y Bounds: 5.700e+06, 5.800e+06
  Z Bounds: 0.000e+00, 7.557e+03
  Dimensions:       2000, 2800, 1
  N Arrays: 1
```

6.15.5 Plotting the Mesh

The mesh can then easily be plotted with PyVista

[7]:
```
sargs = dict(fmt="%.0f", color='black')


p = pv.Plotter(notebook=True)
p.add_mesh(mesh=topo, cmap='gist_earth', scalar_bar_args=sargs, clim=[-0, 500])

p.set_background('white')
p.show_grid(color='black')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis_test\lib\site-packages\pyvista\jupyter\notebook.
↪py:60: UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.15.6 Drape Topographic map over Digital Elevation Model

It is also possible to drape a topographic map over the digital elevation model. The map will be loaded from a WMS service which will be introduced in a later tutorial in more detail. In order to make the feature work, the width and the height of both, the map data and the digital elevation model, must be the same!

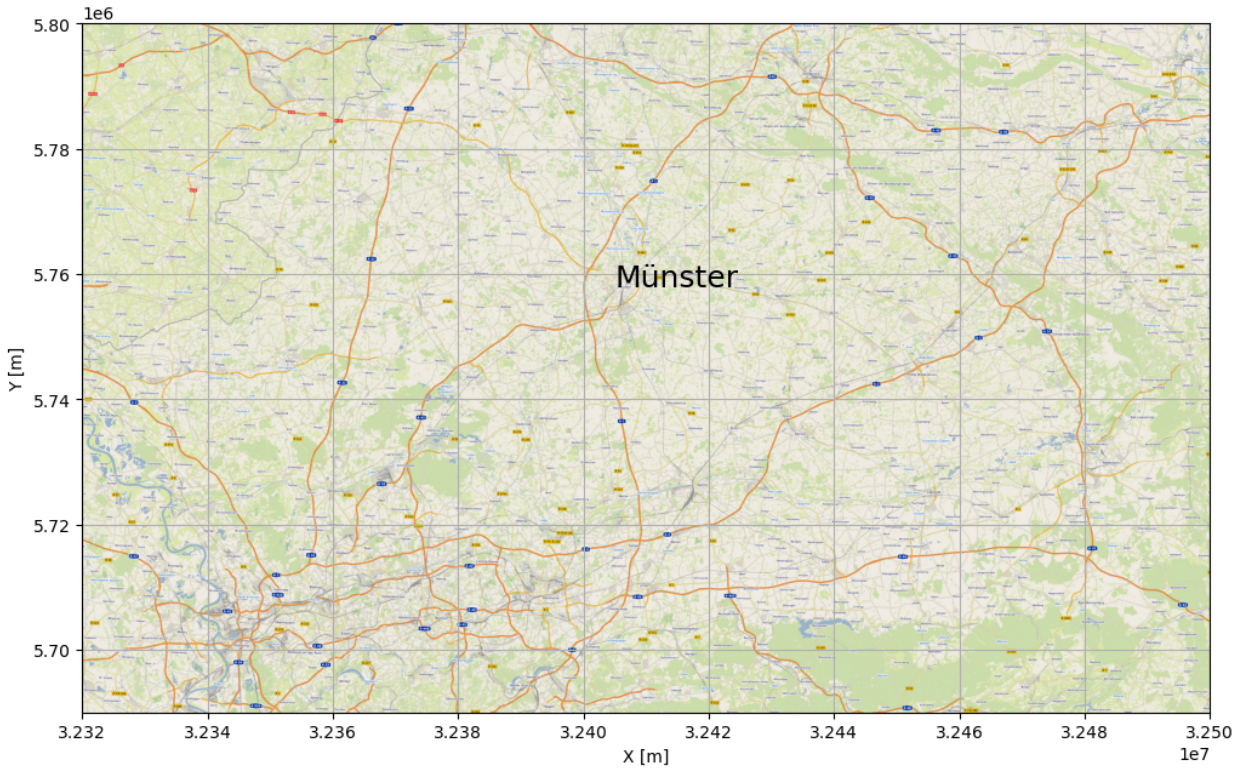
Loading the WMS Service

[8]:
```
wms_map = gg.web.load_as_array('https://ows.terrestris.de/osm/service?',
                               'OSM-WMS', 'default', 'EPSG:4647', [32320000,32500000,␣
↪5690000, 5800000], [2800, 2000], 'image/png')
```

Plotting the map data

[9]:
```
import matplotlib.pyplot as plt

plt.figure(figsize = (12,12))
plt.imshow(wms_map, extent= [32320000,32500000, 5690000, 5800000])
plt.grid()
plt.ylabel('Y [m]')
plt.xlabel('X [m]')
plt.text(32405000,5758000, 'Münster', size = 18)
```

[9]:
```
Text(32405000, 5758000, 'Münster')
```



Converting the array values to RGB values

When downloading the wms_map, a total of three bands are downloaded. These array values need to be converted to RGB values.

[10]:
```
wms_map[0]
```

[10]:
```
array([[0.35686275, 0.35686275, 0.48235294, 1.        ],
       [0.40392157, 0.40784314, 0.52156866, 1.        ],
       [0.92941177, 0.92941177, 0.94509804, 1.        ],
       ...,
       [0.59607846, 0.69803923, 0.8       , 1.        ],
       [0.6156863 , 0.69411767, 0.76862746, 1.        ],
       [0.8784314 , 0.8666667 , 0.8156863 , 1.        ]], dtype=float32)
```

[11]:
```
wms_map.shape
```

[11]:
```
(2000, 2800, 4)
```

[12]:
```
wms_stacked = gg.visualization.convert_to_rgb(array=wms_map)

wms_stacked[:2]
```

[12]:
```
array([[[ 91,  91, 123],
        [103, 104, 133],
        [237, 237, 241],
        ...,
        [152, 178, 204],
        [157, 177, 196],
        [224, 221, 208]],

       [[255, 255, 255],
        [255, 255, 255],
        [247, 248, 243],
        ...,
        [150, 178, 206],
        [171, 175, 170],
        [250, 249, 243]]], dtype=uint8)
```

[13]:
```
wms_stacked.shape
```

[13]:
```
(2000, 2800, 3)
```

Draping the array over the Digital Elevation Model

The map data of the WMS map can be draped over the digital elevation model.

[14]:
```
mesh, texture = gg.visualization.drape_array_over_dem(array=wms_stacked,
                                                      dem=dem)
```

[15]:
```
mesh
```

[15]:
```
StructuredGrid (0x2320e07be80)
  N Cells:  5595201
```

(continues on next page)

(continued from previous page)

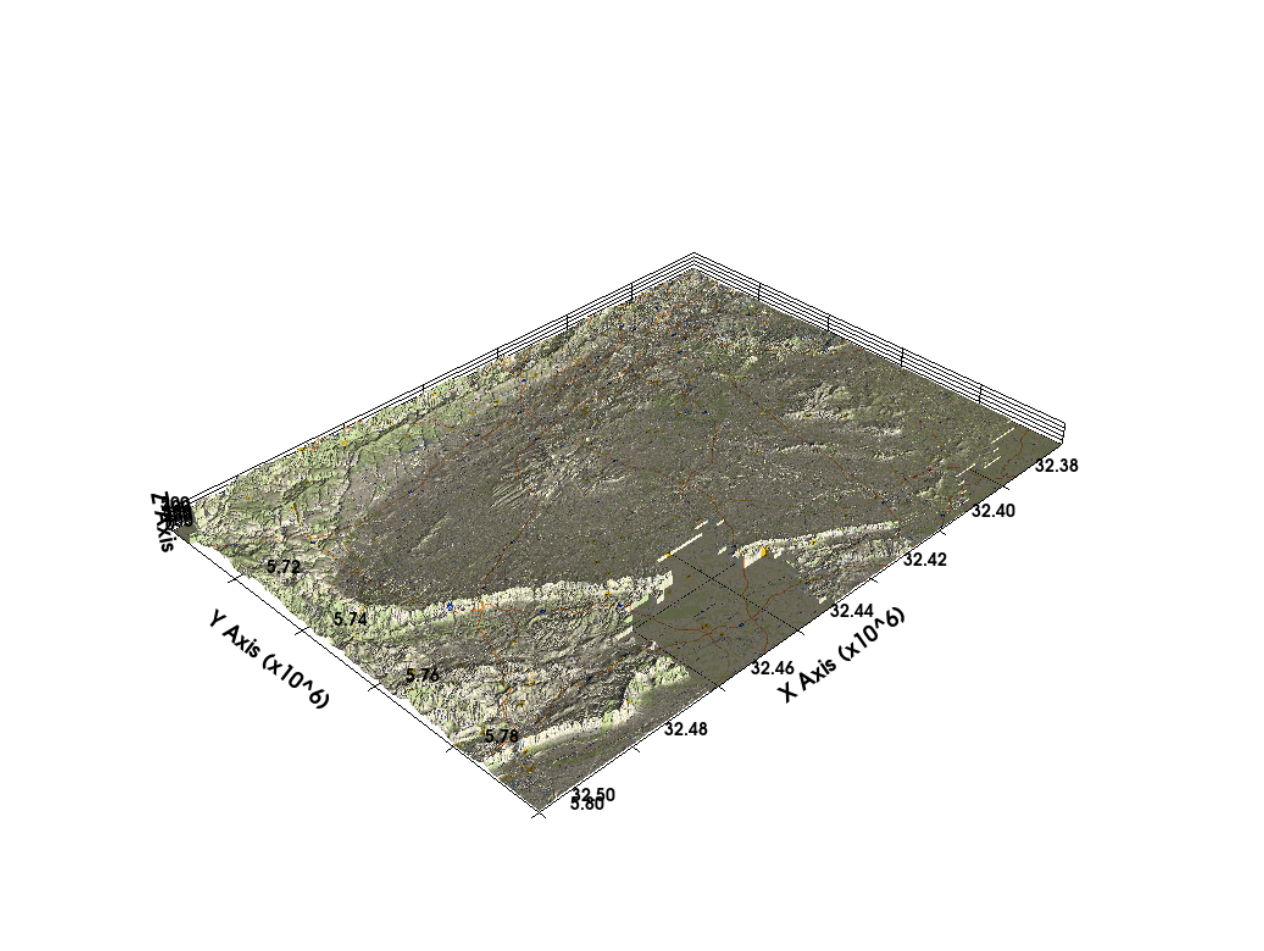
```
  N Points: 5600000
  X Bounds: 3.236e+07, 3.250e+07
  Y Bounds: 5.700e+06, 5.800e+06
  Z Bounds: 0.000e+00, 5.038e+02
  Dimensions:       2000, 2800, 1
  N Arrays: 1
```

[16]:
```
texture
```

[16]:
```
<Texture(0x0000232440EFCF0) at 0x000002320E0786A0>
```

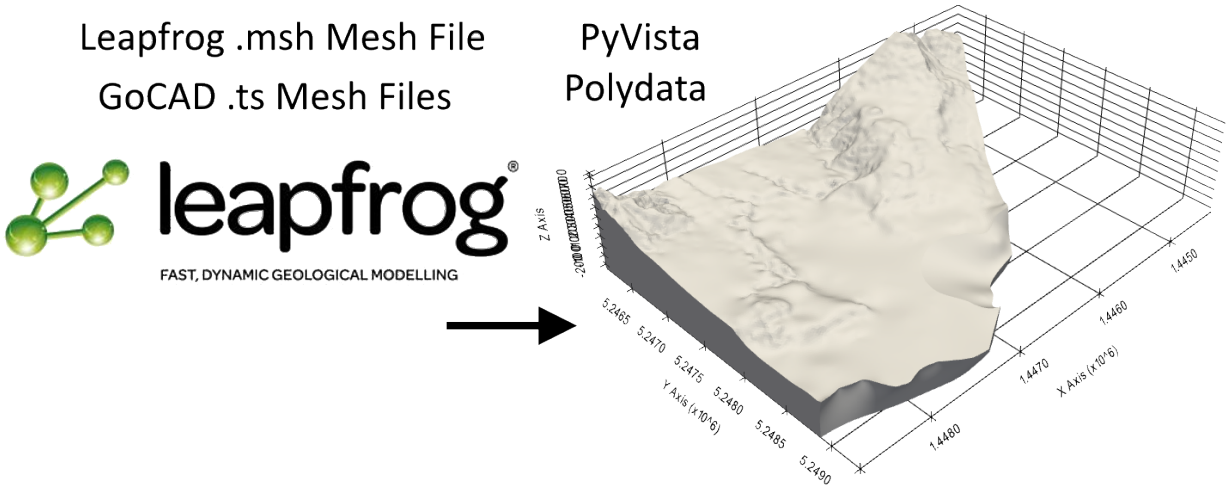
Plotting the data in PyVista

[17]:
```
sargs = dict(fmt="%.0f", color='black')


p = pv.Plotter(notebook=True)
p.add_mesh(mesh=mesh, cmap='gist_earth', scalar_bar_args=sargs, texture=texture)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```



6.16 15 Opening Leapfrog Meshes and GoCAD TSurfaces with GemGIS

Several different modeling packages store their data in different data types. The following illustrates how to load Leapfrog meshes (.msh-files) and GoCAD TSurfaces (.ts-files) with GemGIS and convert them to a plotable PyVista format.



6.16.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/15_opening_leapfrog_meshes_and_gocad_tsurfaces/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="15_opening_leapfrog_meshes_and_
↪gocad_tsurfaces.zip", dirpath=file_path)
```

6.16.2 Reading Leapfrog Meshes

Loading the Mesh Data

The Leapfrog mesh (.msh) is loaded and parsed with `read_msh(..)`. A dictionary containing the face and vertex data will be returned.

[3]:
```
import numpy as np

data = gg.raster.read_msh(file_path + 'GM_Granodiorite.msh')
```

[4]:
```
data
```

[4]:
```
{'Tri': array([[    0,     1,     2],
        [    0,     3,     1],
        [    4,     3,     0],
        ...,
        [53677, 53672, 53680],
        [53679, 53677, 53680],
```

(continues on next page)

(continued from previous page)

```
       [53673, 53672, 53677]]),
 'Location': array([[ 1.44625109e+06,  5.24854344e+06, -1.12743862e+02],
       [ 1.44624766e+06,  5.24854640e+06, -1.15102216e+02],
       [ 1.44624808e+06,  5.24854657e+06, -1.15080548e+02],
       ...,
       [ 1.44831008e+06,  5.24896679e+06, -1.24755449e+02],
       [ 1.44830385e+06,  5.24896985e+06, -1.33694397e+02],
       [ 1.44829874e+06,  5.24897215e+06, -1.42506587e+02]])}
```

[5]:
```
data.keys()
```

[5]:
```
dict_keys(['Tri', 'Location'])
```

Converting Mesh Data to PyVista PolyData

The loaded data will now be converted to PyVista PolyData using `create_polydata_from_msh(..)`.

[6]:
```
surf = gg.visualization.create_polydata_from_msh(data)
surf
```

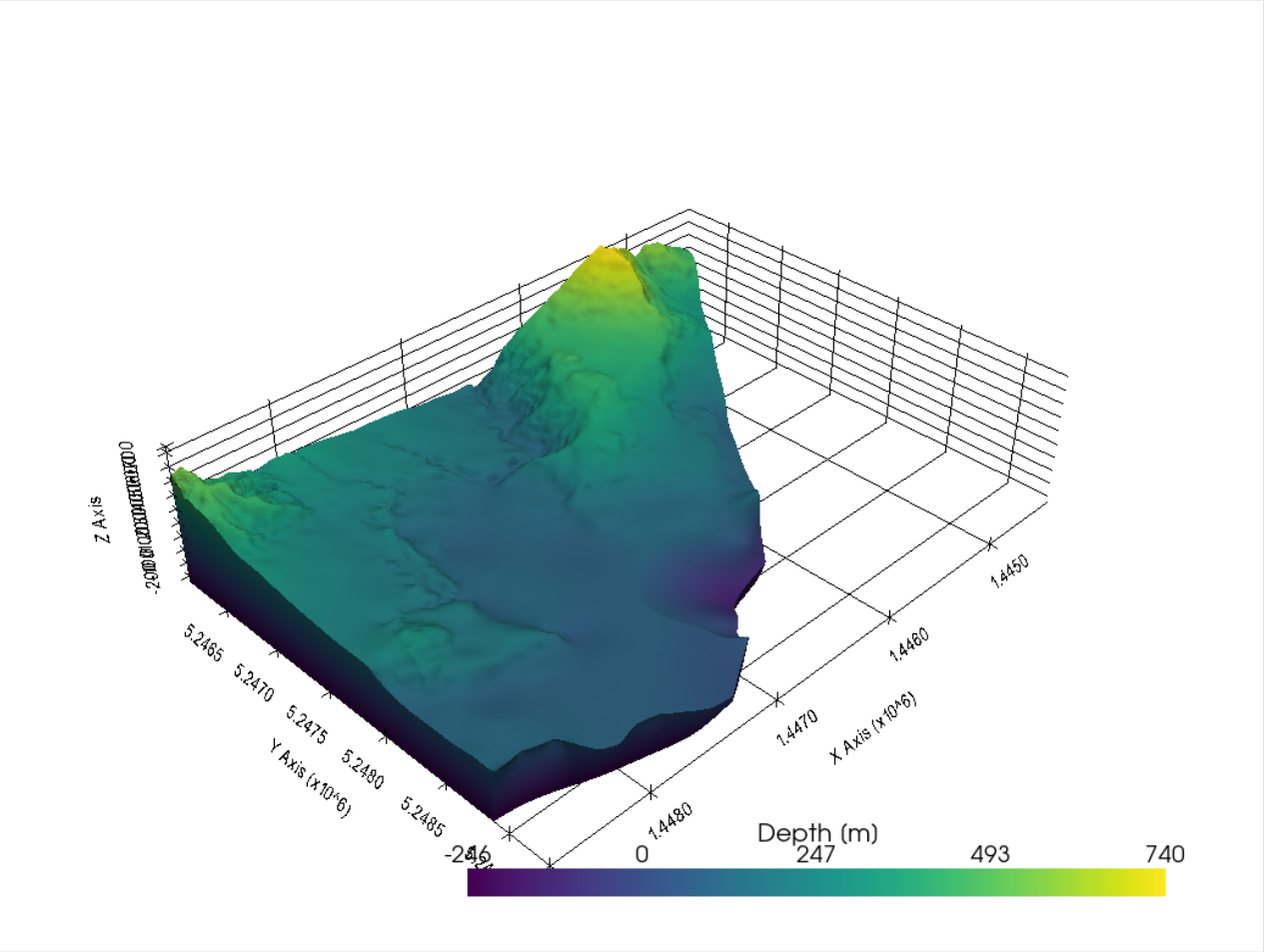
[6]:
```
PolyData (0x276cf76a5c0)
  N Cells:  107358
  N Points: 53681
  N Strips: 0
  X Bounds: 1.444e+06, 1.449e+06
  Y Bounds: 5.246e+06, 5.249e+06
  Z Bounds: -2.464e+02, 7.396e+02
  N Arrays: 1
```

Plotting the data

Once converted, the data can easily be plotted using PyVista.

[7]:
```
import pyvista as pv
sargs = dict(fmt="%.0f", color='black')
p = pv.Plotter(notebook=True)

p.add_mesh(surf, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis_test\lib\site-packages\pyvista\jupyter\notebook.
↪py:60: UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.16.3 Reading GoCAD TSurfaces

Loading the Mesh Data

The GoCAD mesh (.ts) is loaded and parsed with `read_ts(..)`. An array containing the face data and a DataFrame containing the vertex data will be returned.

Source: KVB Model of the Geological Survey NRW

[8]:
```
import gemgis as gg
import numpy as np

data = gg.raster.read_ts(file_path + 'KVB_12_Hermann_Katharina.ts')
```

[9]:
```
data[0][0]
```

[9]:
```
    id         X          Y       Z  aproz     blk   fl  flko  flz1  flz2  \
0    0 297077.41 5677487.26 -838.50      0  672132 5.01    63  1777   277
1    1 297437.54 5676992.09 -816.61      0  672132 5.01    63  1777   277
2    2 298816.17 5677906.68 -590.82      0  672132 5.01    63  1777   277
3    3 298031.07 5678779.55 -648.69      0  672132 5.01    63  1777   277
```

(continues on next page)

(continued from previous page)

```
4    4 298852.68 5678065.33 -578.15      0  672132 5.01    63  1777   277
5    5 298937.09 5677681.53 -586.45      0  672132 5.01    63  1777   277
6    6 298879.99 5677753.93 -590.39      0  672132 5.01    63  1777   277
7    7 296511.14 5678571.41 -857.09      0  672132 5.01    63  1777   277
8    8 296737.27 5677981.61 -857.57      0  672132 5.01    63  1777   277
9    9 295827.68 5680951.57 -825.33      0  672132 5.01    63  1777   277
10  10 295622.92 5680839.83 -857.65      0  672132 5.01    63  1777   277
11  11 298966.25 5677660.32 -583.73      0  672132 5.01    63  1777   277
12  12 299004.59 5677618.71 -580.85      0  672132 5.01    63  1777   277
13  13 297014.31 5679862.08 -726.26      0  672132 5.01    63  1777   277

    grs         id_gocad  kenn     vol    vola
0   672 6721321830000.00     1 3156.85 3156.85
1   672 6721321830000.00     1 3156.85 3156.85
2   672 6721321830000.00     1 3156.85 3156.85
3   672 6721321830000.00     1 3156.85 3156.85
4   672 6721321830000.00     1 3156.85 3156.85
5   672 6721321830000.00     1 3156.85 3156.85
6   672 6721321830000.00     1 3156.85 3156.85
7   672 6721321830000.00     1 3156.85 3156.85
8   672 6721321830000.00     1 3156.85 3156.85
9   672 6721321830000.00     1 3156.85 3156.85
10  672 6721321830000.00     1 3156.85 3156.85
11  672 6721321830000.00     1 3156.85 3156.85
12  672 6721321830000.00     1 3156.85 3156.85
13  672 6721321830000.00     1 3156.85 3156.85
```

[10]:
```
data[1][0]
```

[10]:
```
array([[ 0,  1,  2],
       [ 3,  2,  4],
       [ 1,  5,  6],
       [ 3,  7,  8],
       [ 9, 10,  7],
       [ 3,  0,  2],
       [ 1, 11,  5],
       [ 3,  8,  0],
       [ 1, 12, 11],
       [ 3, 13,  7],
       [ 1,  6,  2],
       [13,  9,  7]])
```

Converting Mesh Data to PyVista PolyData

The loaded data will now be converted to PyVista PolyData using `create_polydata_from_ts(..)`.

[11]:
```
surf = gg.visualization.create_polydata_from_ts(data=data, concat=True)
surf.save(file_path + 'mesh.vtk')
surf
```

[11]:
```
PolyData (0x276da6feda0)
  N Cells:  29271
```

(continues on next page)

(continued from previous page)

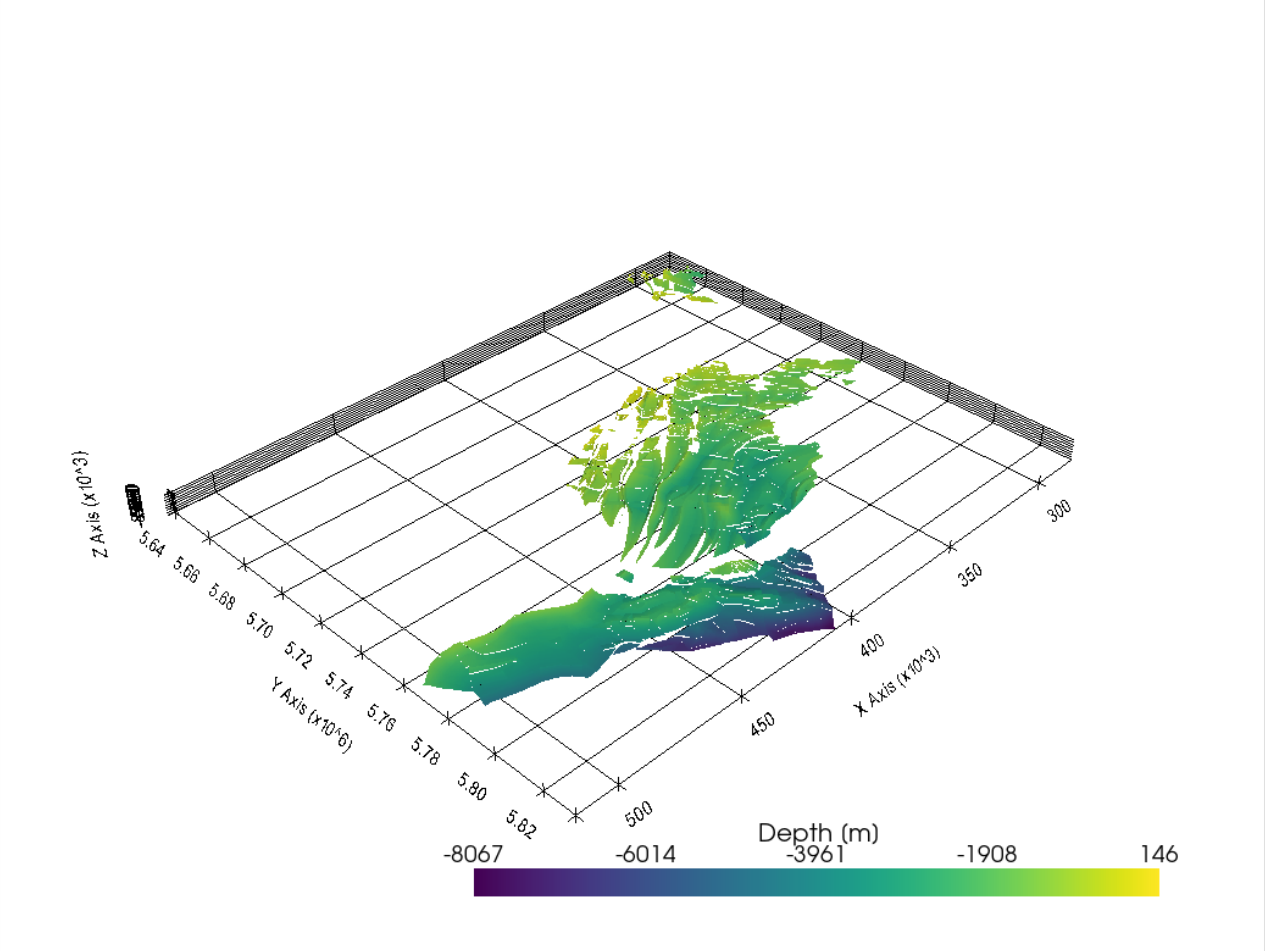
```
  N Points: 40339
  N Strips: 0
  X Bounds: 2.804e+05, 5.161e+05
  Y Bounds: 5.640e+06, 5.833e+06
  Z Bounds: -8.067e+03, 1.457e+02
  N Arrays: 1
```

Plotting the data

Once converted, the data can easily be plotted using PyVista.

[12]:
```
import pyvista as pv
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(surf, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis_test\lib\site-packages\pyvista\jupyter\notebook.
↪py:60: UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



Extracting Polygons from Faces

If you would like to display your mesh data or in particular the faces in a GIS, the faces can be converted to Shapely Polygons using `create_polygons_from_faces(..)`. However, each connected surface is now divided in these triangles. The next step is to unify/merge the triangles that belonged to one surface before.

[13]:
```
polygons = gg.vector.create_polygons_from_faces(mesh=surf, crs='EPSG:25832')
polygons
```

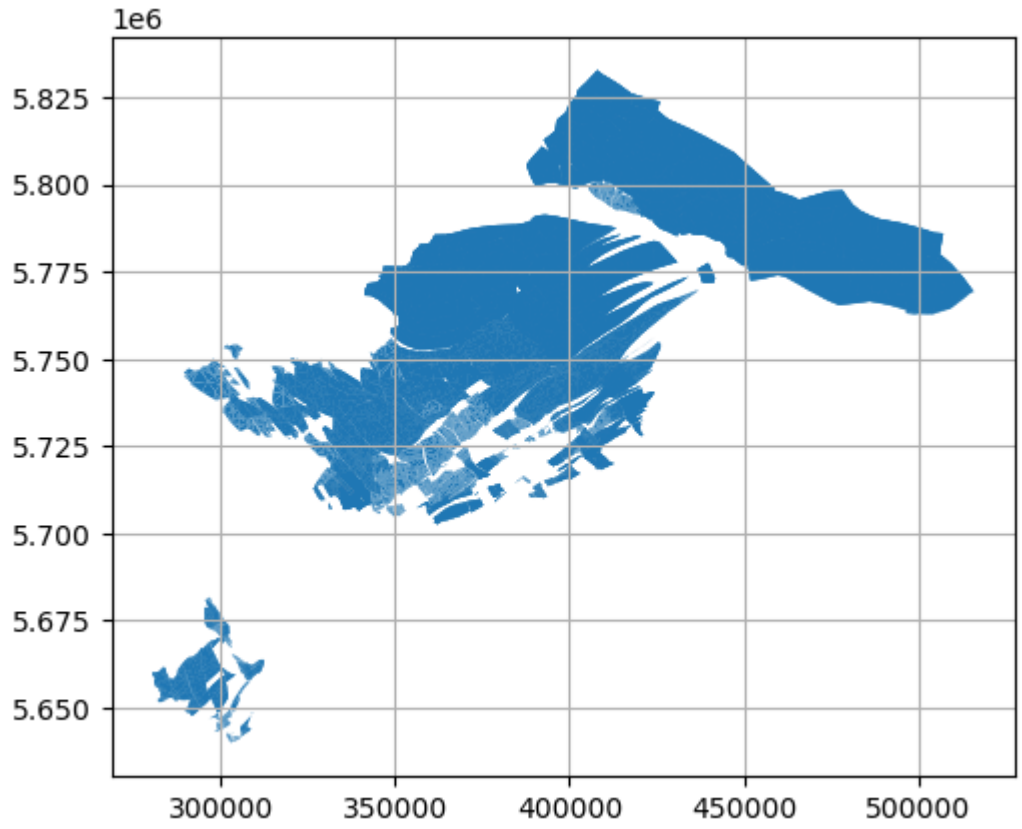
[13]:
```
                                                geometry
0      POLYGON Z ((297077.414 5677487.262 -838.496, 2...
1      POLYGON Z ((298031.070 5678779.547 -648.688, 2...
2      POLYGON Z ((297437.539 5676992.094 -816.608, 2...
3      POLYGON Z ((298031.070 5678779.547 -648.688, 2...
4      POLYGON Z ((295827.680 5680951.574 -825.328, 2...
...                                                  ...
29266  POLYGON Z ((344329.793 5706418.469 -393.606, 3...
29267  POLYGON Z ((344329.793 5706418.469 -393.606, 3...
29268  POLYGON Z ((344329.793 5706418.469 -393.606, 3...
29269  POLYGON Z ((345667.453 5707314.279 -470.917, 3...
29270  POLYGON Z ((345667.453 5707314.279 -470.917, 3...
```

(continues on next page)

(continued from previous page)

```
[29271 rows x 1 columns]
```

[14]:
```
import matplotlib.pyplot as plt

polygons.plot()
plt.grid()
```



6.16.4 Merging Triangles to Polygons

Adjacent triangle can now be merged to form single larger polygons using `unify_polygons(..)`.

NB: Currently, Polygons overlapping in their Z dimension are being connected as the underlying Shapely ``unary_union(..)`` function does not account for these overlaps. This was mentioned here: https://github.com/Toblerity/Shapely/issues/1062.

[15]:
```
polygons = polygons[polygons.is_valid]
polygons_merged = gg.vector.unify_polygons(polygons=polygons)
polygons_merged
```

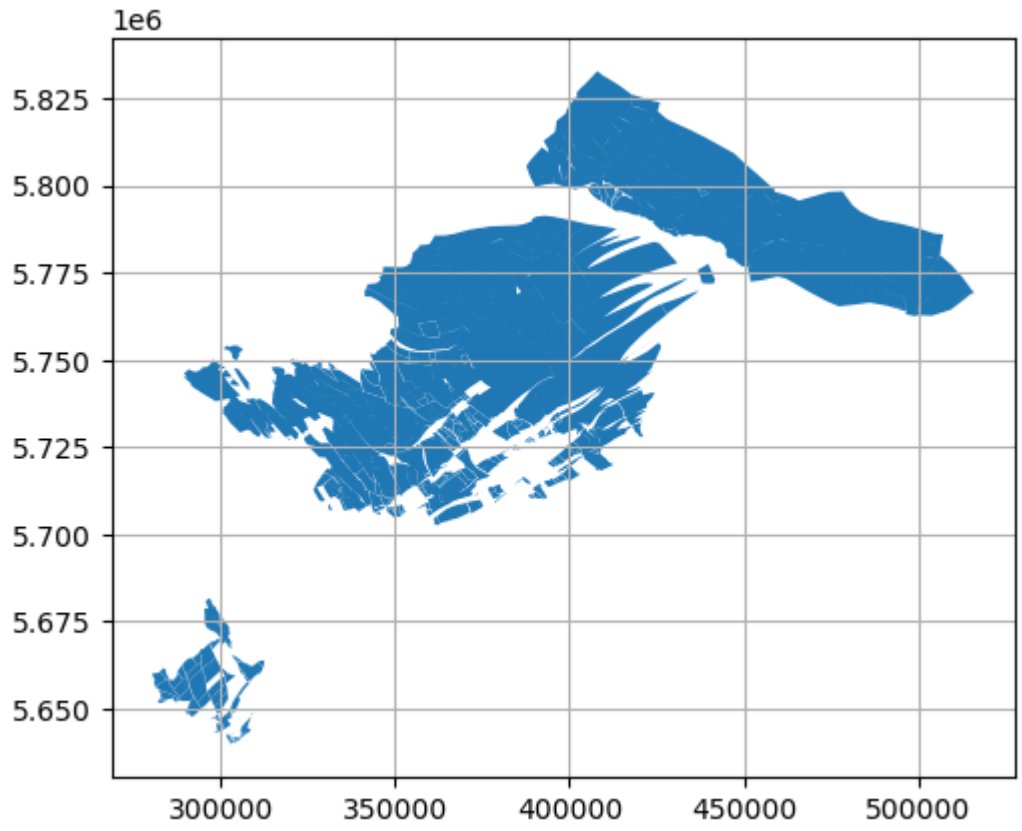
[15]:
```
                                             geometry
0    POLYGON Z ((291580.844 5649724.719 -1109.020, ...
1    POLYGON Z ((290608.406 5648212.812 -429.361, 2...
2    POLYGON Z ((290663.789 5649081.398 -611.540, 2...
3    POLYGON Z ((282206.945 5651577.906 -1157.900, ...
```

(continues on next page)

(continued from previous page)

```
4    POLYGON Z ((288040.180 5652251.234 -1502.510, ...
..                                                ...
252  POLYGON Z ((427861.164 5763280.822 -1735.720, ...
253  POLYGON Z ((426182.065 5765870.283 -1396.278, ...
254  POLYGON Z ((429349.305 5785956.133 -1630.430, ...
255  POLYGON Z ((437544.578 5773703.578 -2109.440, ...
256  POLYGON Z ((438054.836 5796648.406 -2511.880, ...

[257 rows x 1 columns]
```

[16]:
```
polygons_merged.plot()
plt.grid()
```



6.16.5 Opening a second dataset with different file structure

It was encountered that the file structure of TS surfaces of the TUNB model for northern Germany have a different file structure. The code was adapted accordingly and now, both file structures can be opened.

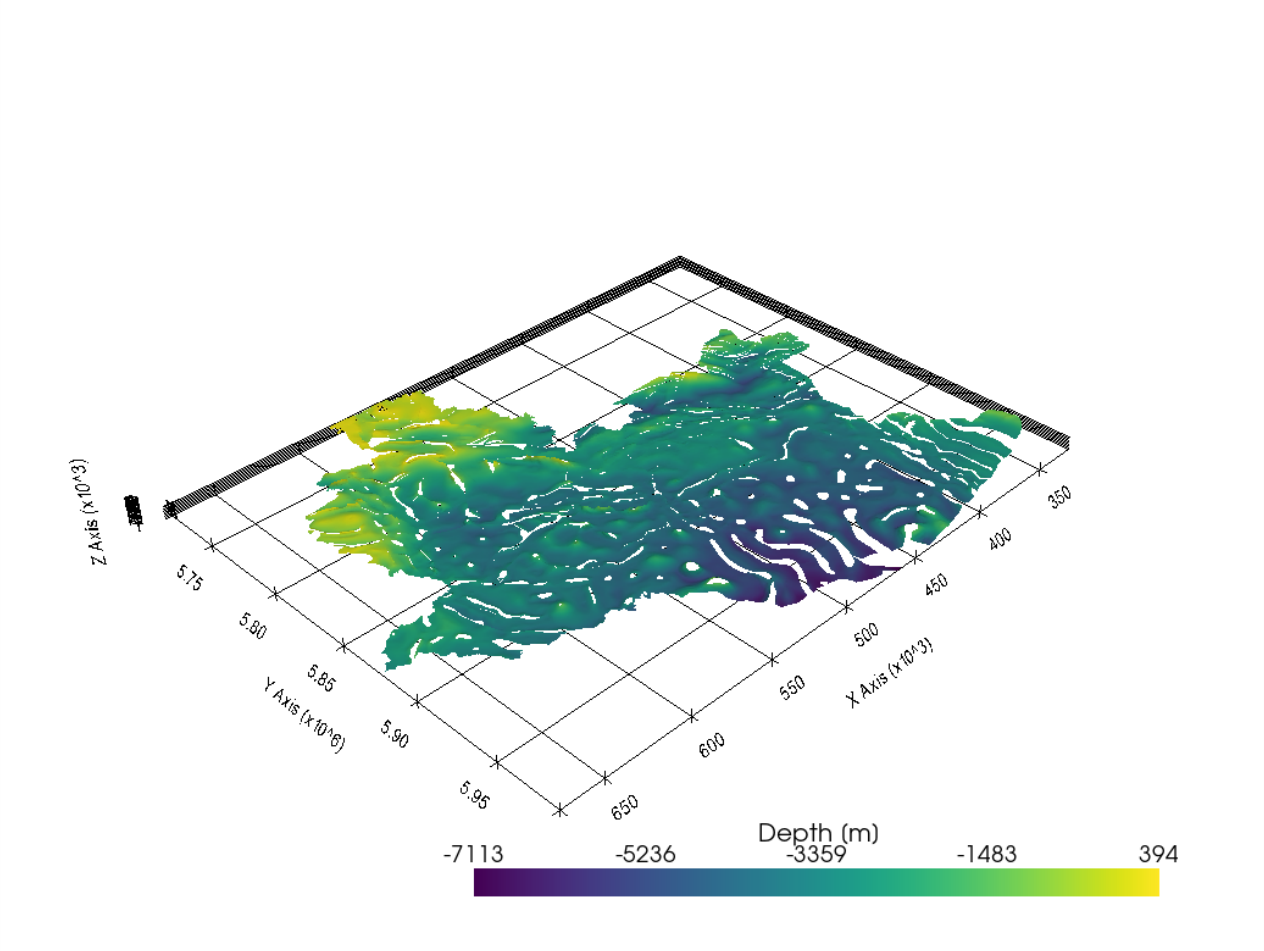
[17]:
```
data2 = gg.raster.read_ts(file_path + '11_NI_sm.ts')
```

[18]:
```
surf2 = gg.visualization.create_polydata_from_ts(data=data2, concat=False)
surf2.save(file_path + 'mesh2.vtk')
surf2
```

[18]:
```
PolyData (0x276e0c84d60)
  N Cells:  1494602
  N Points: 807418
  N Strips: 0
  X Bounds: 3.249e+05, 6.742e+05
  Y Bounds: 5.717e+06, 5.987e+06
  Z Bounds: -7.113e+03, 3.940e+02
  N Arrays: 1
```

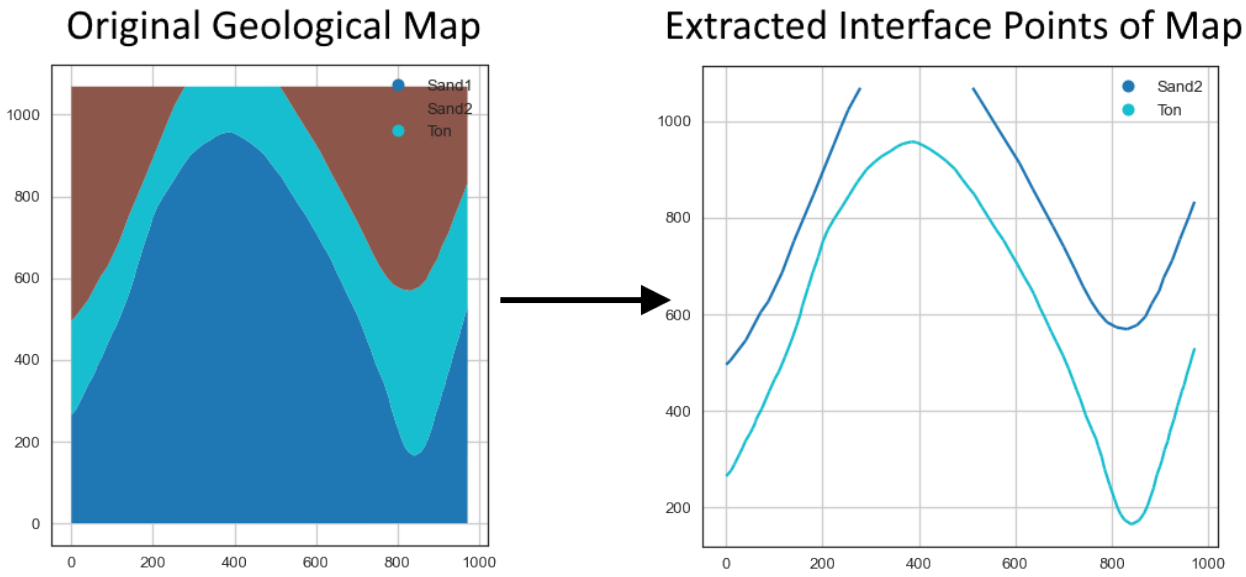
[19]:
```
import pyvista as pv
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(surf2, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis_test\lib\site-packages\pyvista\jupyter\notebook.
↪py:60: UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.17 16 Extracting Interfaces from Geological Maps

Geological Maps nowadays are usually available in a vector format consisting of connected polygons. These polygons consist of single vertices and hence, boundaries between younger and older stratigraphic units can be used for modeling. However, these specific vertices need to be kept while removing all unnecessary ones. The following will introduce how that works in GemGIS.



6.17.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/16_extracting_interfaces_from_geological_maps/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="16_extracting_interfaces_from_
↪geological_maps.zip", dirpath=file_path)
```

```
Downloading file '16_extracting_interfaces_from_geological_maps.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F16_extracting_interfaces_from_
↪geological_maps.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\
↪tutorial\data\16_extracting_interfaces_from_geological_maps'.
```

6.17.2 Loading Data

A simple geological map will be used to demonstrate the feature.

[3]:
```
import geopandas as gpd

gmap = gpd.read_file(file_path + 'interfaces_polygons.shp')
gmap
```

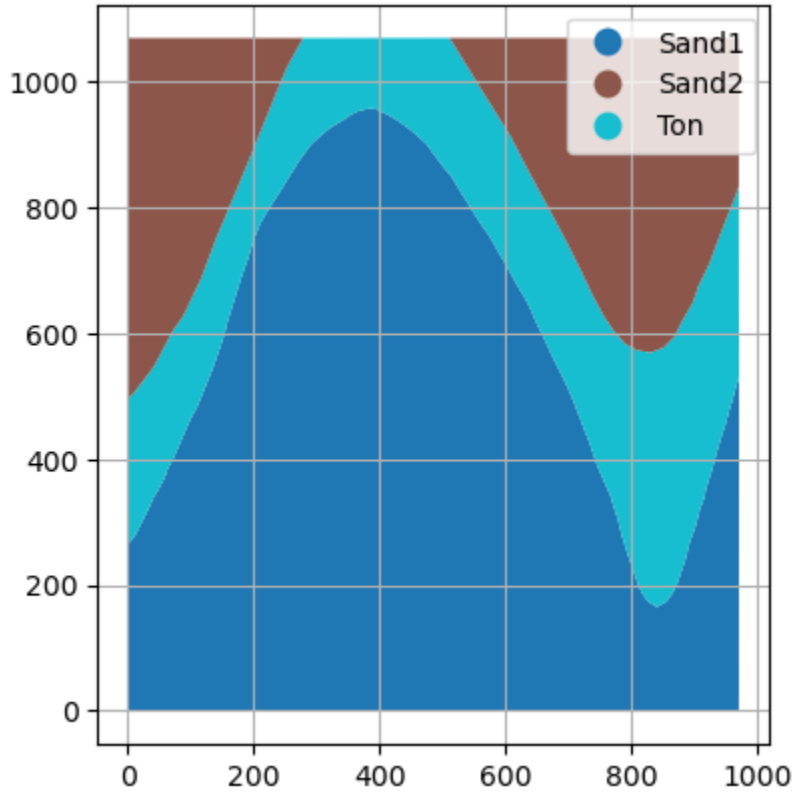
[3]:
```
     id formation                                           geometry
0  None     Sand1  POLYGON ((0.25633 264.86215, 10.59347 276.7337...
1  None       Ton  POLYGON ((0.25633 264.86215, 0.18819 495.78721...
```

(continues on next page)

(continued from previous page)

```
2  None     Sand2  POLYGON ((0.18819 495.78721, 0.24897 1068.7595...
3  None     Sand2  POLYGON ((511.67477 1068.85246, 971.69794 1068...
```

[4]:
```
import matplotlib.pyplot as plt

gmap.plot(column='formation', aspect='equal', legend=True)
plt.grid()
```



6.17.3 Sorting the GeoDataFrame by stratigraphic age

The probably most important step of this workflow is to sort the stratigraphic units according to their age. This can be done using the function `sort_by_stratigraphy(..)` and by providing a list with the formations in stratigraphic order from young to old.

[5]:
```
stratigraphy = ['Sand2', 'Ton', 'Sand1']

gmap_sorted = gg.vector.sort_by_stratigraphy(gdf=gmap,
                                             stratigraphy=stratigraphy)

gmap_sorted
```

[5]:
```
     id formation                                           geometry
0  None     Sand2  POLYGON ((0.18819 495.78721, 0.24897 1068.7595...
1  None     Sand2  POLYGON ((511.67477 1068.85246, 971.69794 1068...
```

(continues on next page)

(continued from previous page)

```
2  None       Ton  POLYGON ((0.25633 264.86215, 0.18819 495.78721...
3  None     Sand1  POLYGON ((0.25633 264.86215, 10.59347 276.7337...
```

6.17.4 Extracting Intersections from Polygons

The functionality is mainly based on the intersections between Shapely Polygons which will be demonstrated below.

Intersections between two polygons

[6]:
```
gmap_sorted.loc[3].geometry
```

[6]:

[7]:
```
gmap_sorted.loc[2].geometry
```

[7]:

[8]:
```
intersection = gg.vector.intersection_polygon_polygon(polygon1=gmap_sorted.loc[3].
↪geometry,
                                                      polygon2=gmap_sorted.loc[2].
↪geometry)
intersection
```

[8]:

Intersections between a polygon and polygons

[9]:
```
gmap_sorted.loc[2].geometry
```

[9]:

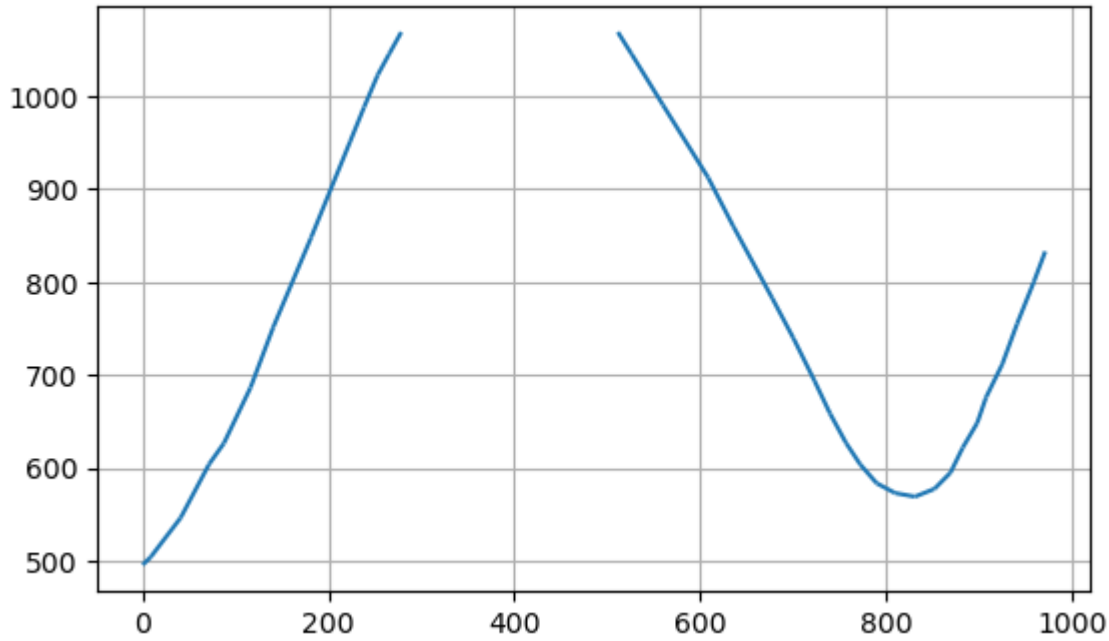
[10]:
```
poly_list=[gmap_sorted.loc[1].geometry, gmap_sorted.loc[0].geometry]
poly_list
```

[10]:
```
[<POLYGON ((511.675 1068.852, 971.698 1068.8, 971.738 832.976, 959.372 800.02...>,
 <POLYGON ((0.188 495.787, 0.249 1068.76, 278.524 1068.772, 265.602 1045.513,...>]
```

[11]:
```
intersection = gg.vector.intersections_polygon_polygons(polygon1=gmap_sorted.loc[2].
↪geometry,
                                                        polygons2=poly_list)
intersection_gdf = gpd.GeoDataFrame(geometry=intersection)
intersection_gdf
```

[11]:
```
                                          geometry
0  MULTILINESTRING ((511.675 1068.852, 526.375 10...
1  MULTILINESTRING ((0.188 495.787, 8.841 504.142...
```

[12]:
```
intersection_gdf.plot()
plt.grid()
```



Intersections between polygons and polygons

[13]:
```
gdf1 = gpd.GeoDataFrame(geometry=[gmap_sorted.loc[2].geometry])
gdf1
```

[13]:
```
                                            geometry
0  POLYGON ((0.256 264.862, 0.188 495.787, 8.841 ...
```

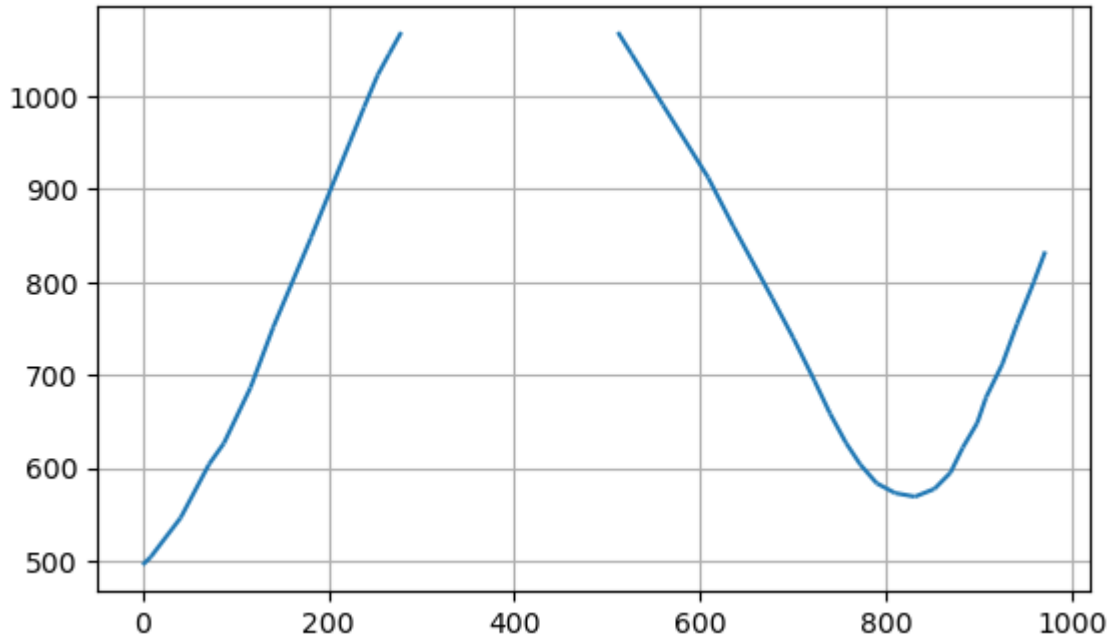
[14]:
```
gdf2 = gpd.GeoDataFrame(geometry=[gmap_sorted.loc[1].geometry, gmap_sorted.loc[0].
↪geometry])
gdf2
```

[14]:
```
                                            geometry
0  POLYGON ((511.675 1068.852, 971.698 1068.800, ...
1  POLYGON ((0.188 495.787, 0.249 1068.760, 278.5...
```

[15]:
```
intersection = gg.vector.intersections_polygons_polygons(polygons1=gdf1,
                                                         polygons2=gdf2)

intersection_gdf = gpd.GeoDataFrame(geometry=intersection)
intersection_gdf
```

[15]:
```
                                            geometry
0  MULTILINESTRING ((511.675 1068.852, 526.375 10...
1  MULTILINESTRING ((0.188 495.787, 8.841 504.142...
```

[16]:
```
intersection_gdf.plot()
plt.grid()
```



6.17.5 Extracting Interfaces from Geological Map

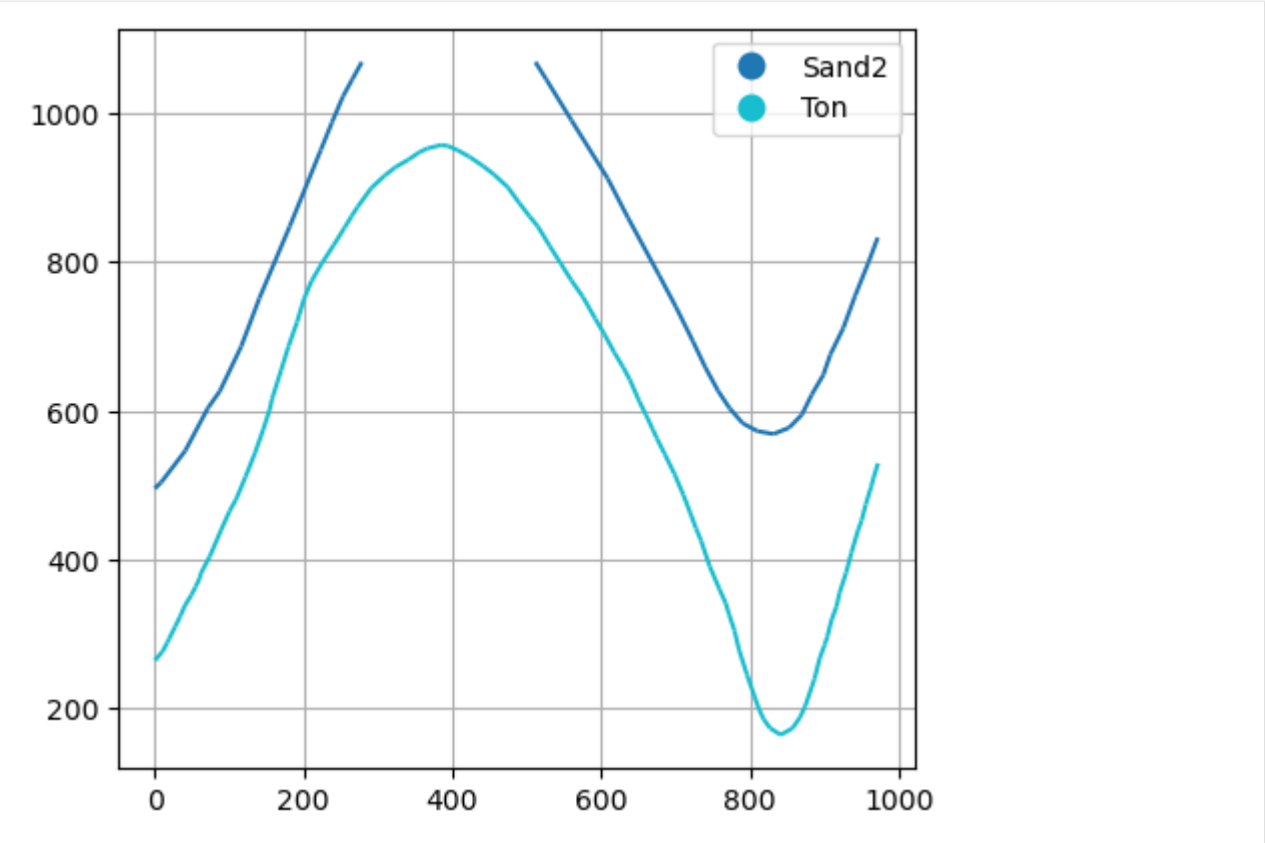
The final step is to use the entire GeoDataFrame for the extraction of the intersections. The resulting GeoDataFrame only contains interfaces of formations that also have a base. Therefore, interfaces of `Sand1` are missing as this formation represents the basement of the stratigraphy.

[17]:
```
intersection = gg.vector.extract_xy_from_polygon_intersections(gdf=gmap_sorted)

intersection
```

[17]:
```
     id formation                                           geometry
0  None     Sand2  MULTILINESTRING ((278.52378 1068.77171, 265.60...
1  None     Sand2  MULTILINESTRING ((971.73811 832.97587, 959.372...
2  None       Ton  MULTILINESTRING ((972.08890 529.79176, 966.073...
```

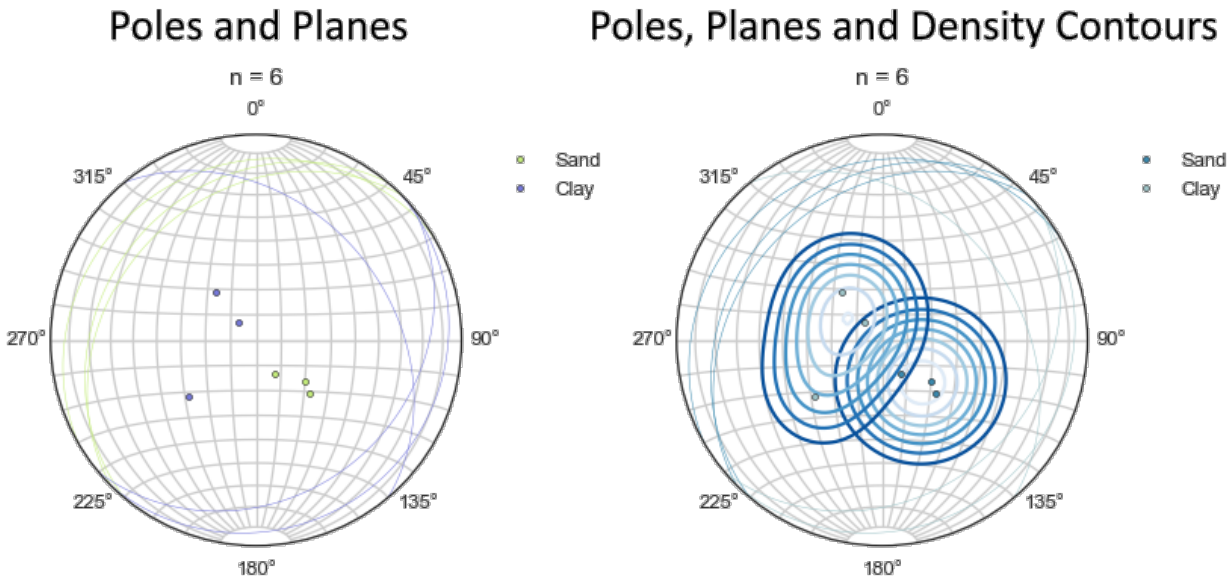
When plotting the data, it can be seen that there are three different boundaries. First, there is the base of `Ton` and then twice the base of `Sand2`.

[18]:
```
intersection.plot(column='formation', aspect='equal', legend=True)
plt.grid()
```



6.18 17 Plotting Orientations with mplstereonet

Orientation measurements stored as GeoDataFrame can easily be plotted using `GemGIS` and `mplstereonet`.



6.18.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/17_plotting_orientations_with_mplstereonet/'
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\gemgis\gemgis.py:27:␣
↪UserWarning: Shapely 2.0 is installed, but because PyGEOS is also installed, GeoPandas␣
↪will still use PyGEOS by default for now. To force to use and test Shapely 2.0, you␣
↪have to set the environment variable USE_PYGEOS=0. You can do this before starting the␣
↪Python process, or in your code before importing geopandas:

import os
os.environ['USE_PYGEOS'] = '0'
import geopandas

In a future release, GeoPandas will switch to using Shapely by default. If you are using␣
↪PyGEOS directly (calling PyGEOS functions on geometries from GeoPandas), this will␣
↪then stop working and you are encouraged to migrate from PyGEOS to Shapely 2.0 (https:/
↪/shapely.readthedocs.io/en/latest/migration_pygeos.html).
  import geopandas as gpd
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="17_plotting_orientations_with_
↪mplstereonet.zip", dirpath=file_path)
```

6.18.2 Loading the Data

[3]:
```
import geopandas as gpd

orientations = gpd.read_file(file_path + 'orientations.shp')
orientations
```

[3]:

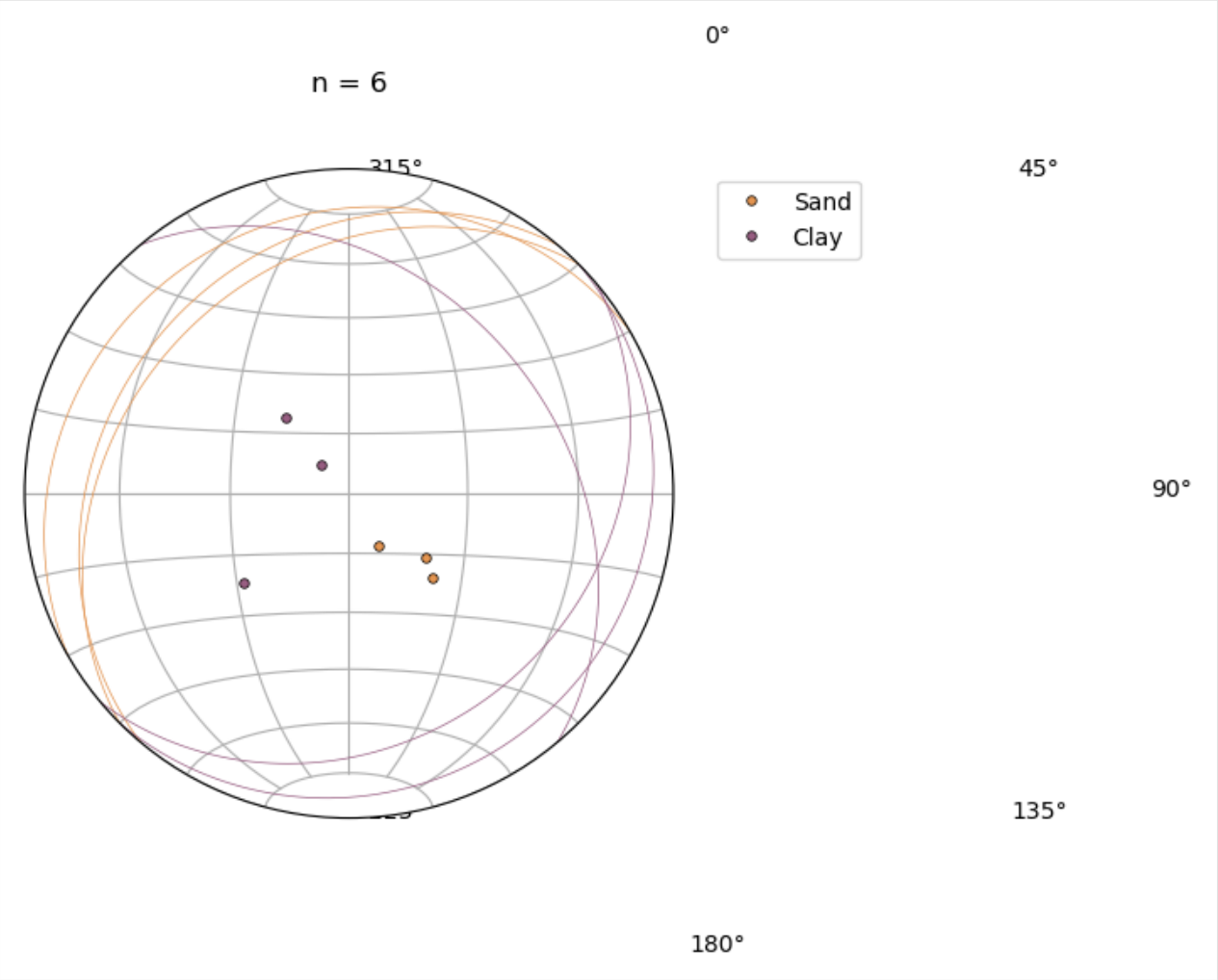
	id	formation	dip	azimuth	geometry
0	None	Sand	25.00	310	POINT (49.24855 1033.89335)
1	None	Sand	30.00	315	POINT (355.21200 947.55713)
2	None	Sand	15.00	330	POINT (720.24760 880.91163)
3	None	Clay	10.00	135	POINT (526.36977 611.30027)
4	None	Clay	25.00	140	POINT (497.59103 876.36762)
5	None	Clay	35.00	50	POINT (394.59343 481.03860)

6.18.3 Plotting the data

The data can be plotted using the `GemGIS` function `plot_orientations` and the underlying `mplstereonet` functions.

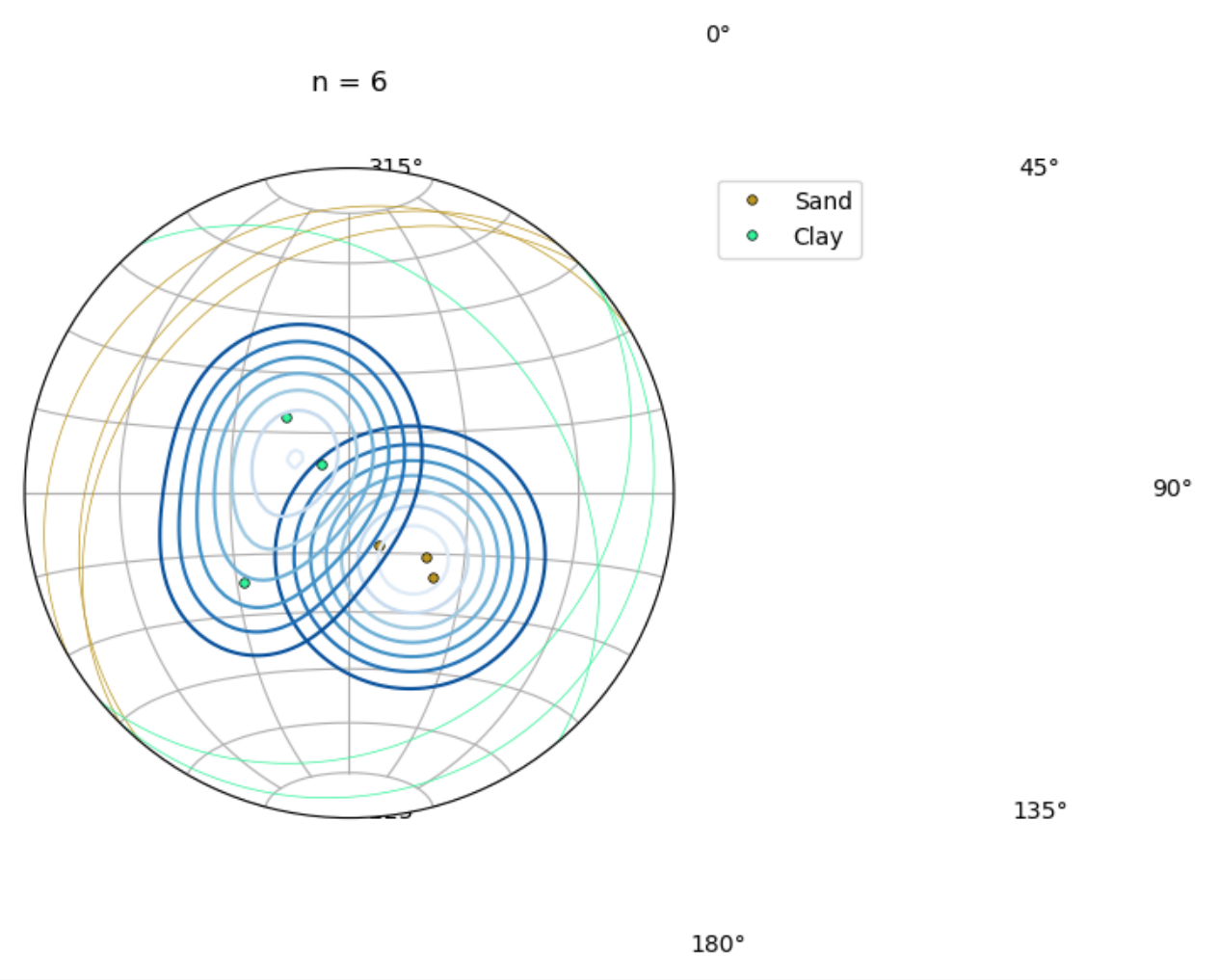
In the plot below, the poles and great circles are plotted.

[4]:
```
gg.visualization.plot_orientations(gdf=orientations,
                                   show_planes=True,
                                   show_density_contours=False,
                                   show_density_contourf=False,
                                   )
```



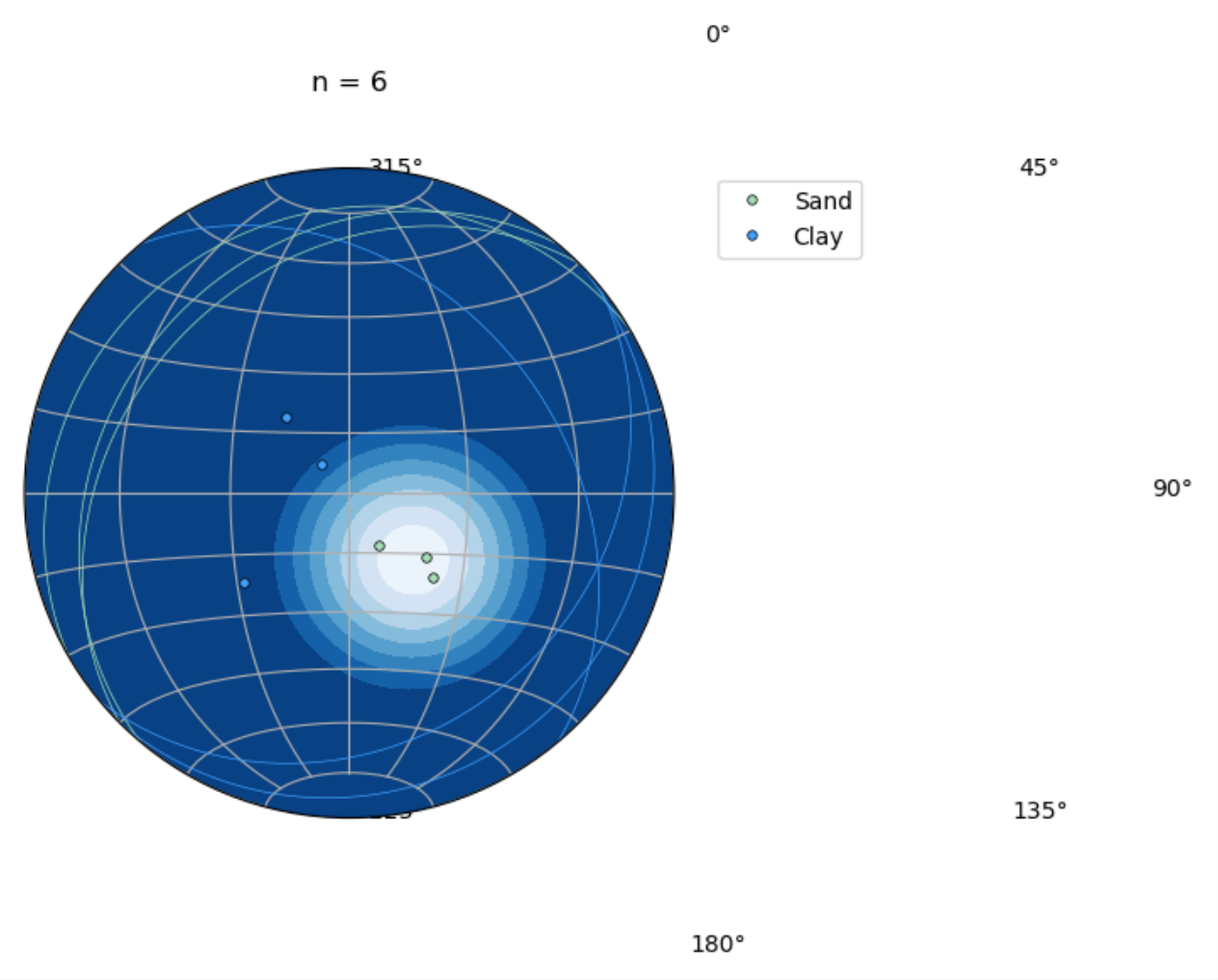
In addition, the density contours can be plotted. The `method` for calculating the density contours can be changed.

[5]:
```
gg.visualization.plot_orientations(gdf=orientations,
                                   show_planes=True,
                                   show_density_contours=True,
                                   show_density_contourf=False,
                                   )
```

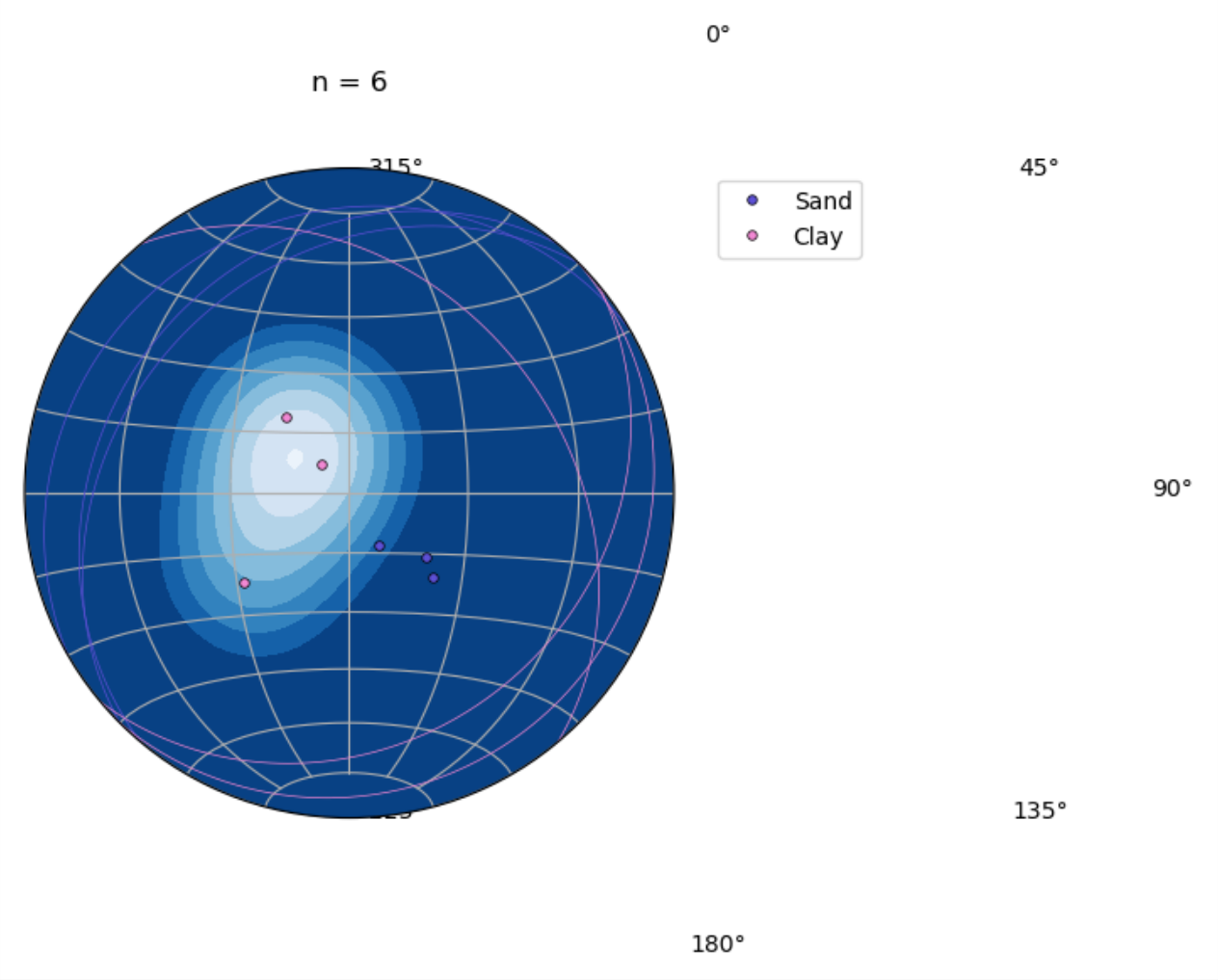


The filled density contours can be plotted for one formation by providing a `formation` name.

[6]:
```
gg.visualization.plot_orientations(gdf=orientations,
                                   show_planes=True,
                                   show_density_contours=False,
                                   show_density_contourf=True,
                                   formation='Sand'
                                   )
```

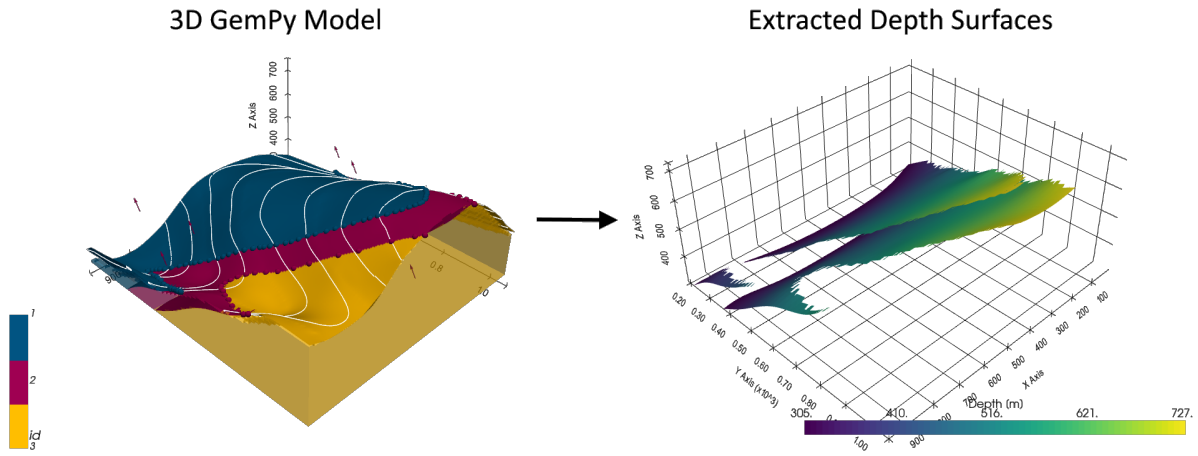


[7]:
```
gg.visualization.plot_orientations(gdf=orientations,
                                   show_planes=True,
                                   show_density_contours=False,
                                   show_density_contourf=True,
                                   formation='Clay'
                                   )
```



6.19 18 Creating Depth Maps from GemPy Models

Depth maps are an important tool for Geologists to visualize the spatial distribution of layers at depth. A simple model (Example 1) is created of which depth maps for the two existing layers are created.



6.19.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/18_creating_depth_maps_from_gempy_models/'
```

```
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\numpy\_distributor_init.py:30:␣
↪UserWarning: loaded more than 1 DLL from .libs:
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\numpy\.libs\libopenblas.
↪4SP5SUA7CBGXUEOC35YP2ASOICYYEQZZ.gfortran-win_amd64.dll
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\numpy\.libs\libopenblas.
↪GK7GX5KEQ4F6UYO3P26ULGBQYHGQO7J4.gfortran-win_amd64.dll
  warnings.warn("loaded more than 1 DLL from .libs:"
```

[12]:
```
gg.download_gemgis_data.download_tutorial_data(filename="18_creating_depth_maps_from_
↪gempy_models.zip", dirpath=file_path)
```

6.19.2 Loading the data

[3]:
```
import geopandas as gpd
import rasterio

interfaces = gpd.read_file(file_path + 'interfaces.shp')
orientations = gpd.read_file(file_path + 'orientations.shp')
extent = [0,972,0,1069, 300, 800]
resolution = [50, 50, 50]
```

[4]:
```
interfaces.head()
```

[4]:
```
   level_0  level_1 formation     X      Y      Z                    geometry
0        0        0     Sand1  0.26 264.86 353.97   POINT (0.25633 264.86215)
1        0        0     Sand1 10.59 276.73 359.04  POINT (10.59347 276.73371)
2        0        0     Sand1 17.13 289.09 364.28  POINT (17.13494 289.08982)
3        0        0     Sand1 19.15 293.31 364.99  POINT (19.15013 293.31349)
4        0        0     Sand1 27.80 310.57 372.81  POINT (27.79512 310.57169)
```

[5]:
```
orientations['polarity'] = 1
orientations.head()
```

[5]:
```
  formation   dip  azimuth      X      Y      Z                    geometry  \
0       Ton 30.50   180.00  96.47 451.56 477.73   POINT (96.47104 451.56362)
1       Ton 30.50   180.00 172.76 661.88 481.73  POINT (172.76101 661.87650)
2       Ton 30.50   180.00 383.07 957.76 444.45  POINT (383.07389 957.75787)
3       Ton 30.50   180.00 592.36 722.70 480.57  POINT (592.35583 722.70229)
4       Ton 30.50   180.00 766.59 348.47 498.96  POINT (766.58562 348.46907)

   polarity
0         1
1         1
2         1
3         1
4         1
```

6.19.3 Creating the GemPy Model

[6]:
```
import sys
sys.path.append('../../../../gempy-master')
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\theano\configdefaults.py:560:␣
↪UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and with␣
↪Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
```

```
Not subsurface compatibility available
```

```
WARNING (theano.configdefaults): install mkl with `conda install mkl-service`: No module␣
↪named 'mkl'
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[7]:
```
geo_model = gp.create_model('Model1')
geo_model
```

[7]:
```
Model1  2023-03-01 12:34
```

Initiating the Model

[8]:
```
import pandas as pd

gp.init_data(geo_model, extent, resolution,
             surface_points_df = interfaces,
             orientations_df = orientations,
             default_values=True)
geo_model.surfaces
```

```
Active grids: ['regular']
```

[8]:
```
  surface          series  order_surfaces    color  id
0   Sand1  Default series               1  #015482   1
1     Ton  Default series               2  #9f0052   2
```

The vertices and edges are currently NaN values, so no model has been computed so far.

[9]:
```
geo_model.surfaces.df
```

[9]:
```
  surface          series  order_surfaces  isBasement isFault  isActive  \
0   Sand1  Default series               1       False   False      True
1     Ton  Default series               2        True   False      True

   hasData    color vertices edges sfai  id
0     True  #015482      NaN   NaN  NaN   1
1     True  #9f0052      NaN   NaN  NaN   2
```

Mapping Stack to Surfaces

[10]:
```
gp.map_stack_to_surfaces(geo_model,
                         {"Strat_Series": ('Sand1', 'Ton')},
                         remove_unused_series=True)
geo_model.add_surfaces('basement')
```

[10]:
```
    surface        series  order_surfaces    color  id
0     Sand1  Strat_Series               1  #015482   1
1       Ton  Strat_Series               2  #9f0052   2
2  basement  Strat_Series               3  #ffbe00   3
```

Loading Topography

[13]:
```
geo_model.set_topography(
    source='gdal', filepath='data/18_creating_depth_maps_from_gempy_models/raster1.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[13]:
```
Grid Object. Values:
array([[   9.72      ,   10.69      ,  305.        ],
```

(continues on next page)

(continued from previous page)

```
       [   9.72      ,   10.69      ,  315.        ],
       [   9.72      ,   10.69      ,  325.        ],
       ...,
       [ 970.056     , 1059.28181818,  622.0892334 ],
       [ 970.056     , 1063.16909091,  622.06713867],
       [ 970.056     , 1067.05636364,  622.05786133]])
```

Setting Interpolator

[14]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                   values
range            1528.90
$C_o$           55655.83
drift equations      [3]
```

[14]:
```
<gempy.core.interpolator.InterpolatorModel at 0x24a0e6b8df0>
```

Computing Model

[15]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

The surfaces DataFrame now contains values for vertices and edges.

[16]:
```
geo_model.surfaces.df
```

[16]:
```
    surface        series  order_surfaces  isBasement isFault  isActive  \
0     Sand1  Strat_Series               1       False   False      True
1       Ton  Strat_Series               2       False   False      True
2  basement  Strat_Series               3        True   False      True

   hasData    color                                            vertices  \
0     True  #015482  [[29.160000000000004, 194.27877317428587, 305...
1     True  #9f0052  [[29.160000000000004, 365.78652999877926, 305...
2     True  #ffbe00                                                NaN

                                          edges sfai  id
0  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.26   1
```

(continues on next page)

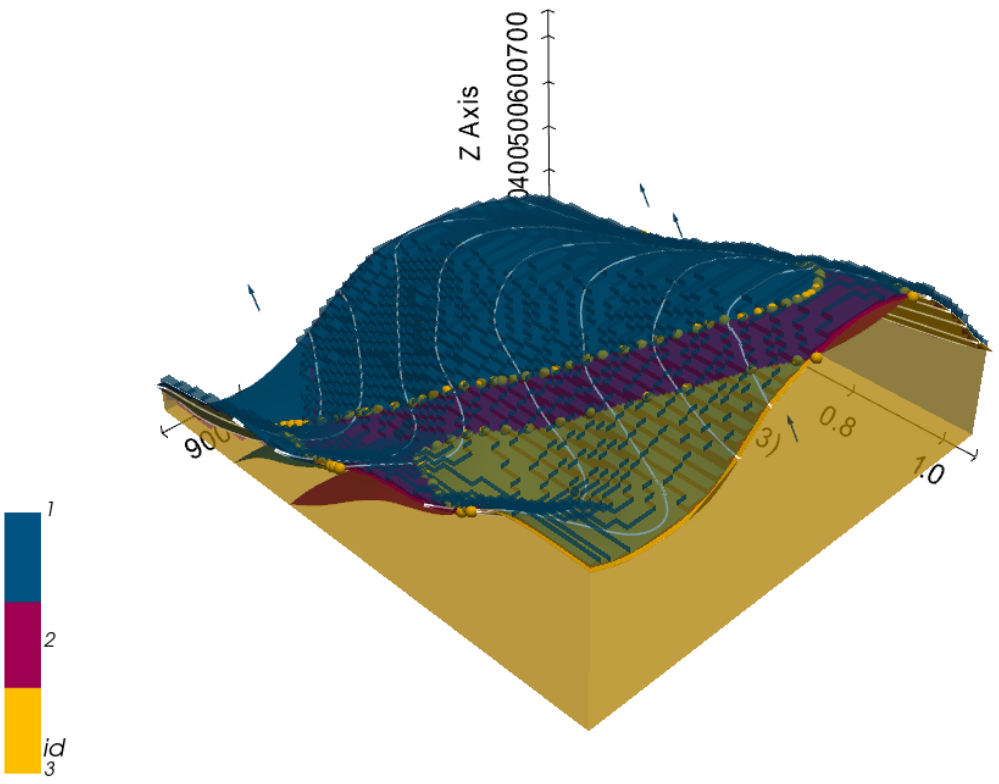
(continued from previous page)

```
1  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.21   2
2                                                  NaN  NaN   3
```

Plotting the 3D Model

[17]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```

```
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\pyvista\plotting\tools.py:571:␣
↪PyvistaDeprecationWarning: The usage of `parse_color` is deprecated in favor of the␣
↪new `Color` class.
  warnings.warn(
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:␣
↪UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.19.4 Creating Depth Maps

When creating the depth maps, a dict containing the mesh, the depth values and the color of the surface within the GemPy Model are returned.

[18]:
```
dict_sand1 = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model,
                                                           surfaces='Sand1')

dict_sand1
```

[18]:
```
{'Sand1': [PolyData (0x24a0e712520)
    N Cells:    4174
    N Points:   2303
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   1.881e+02, 9.491e+02
    Z Bounds:   3.050e+02, 7.250e+02
    N Arrays:   1,
  '#015482']}
```

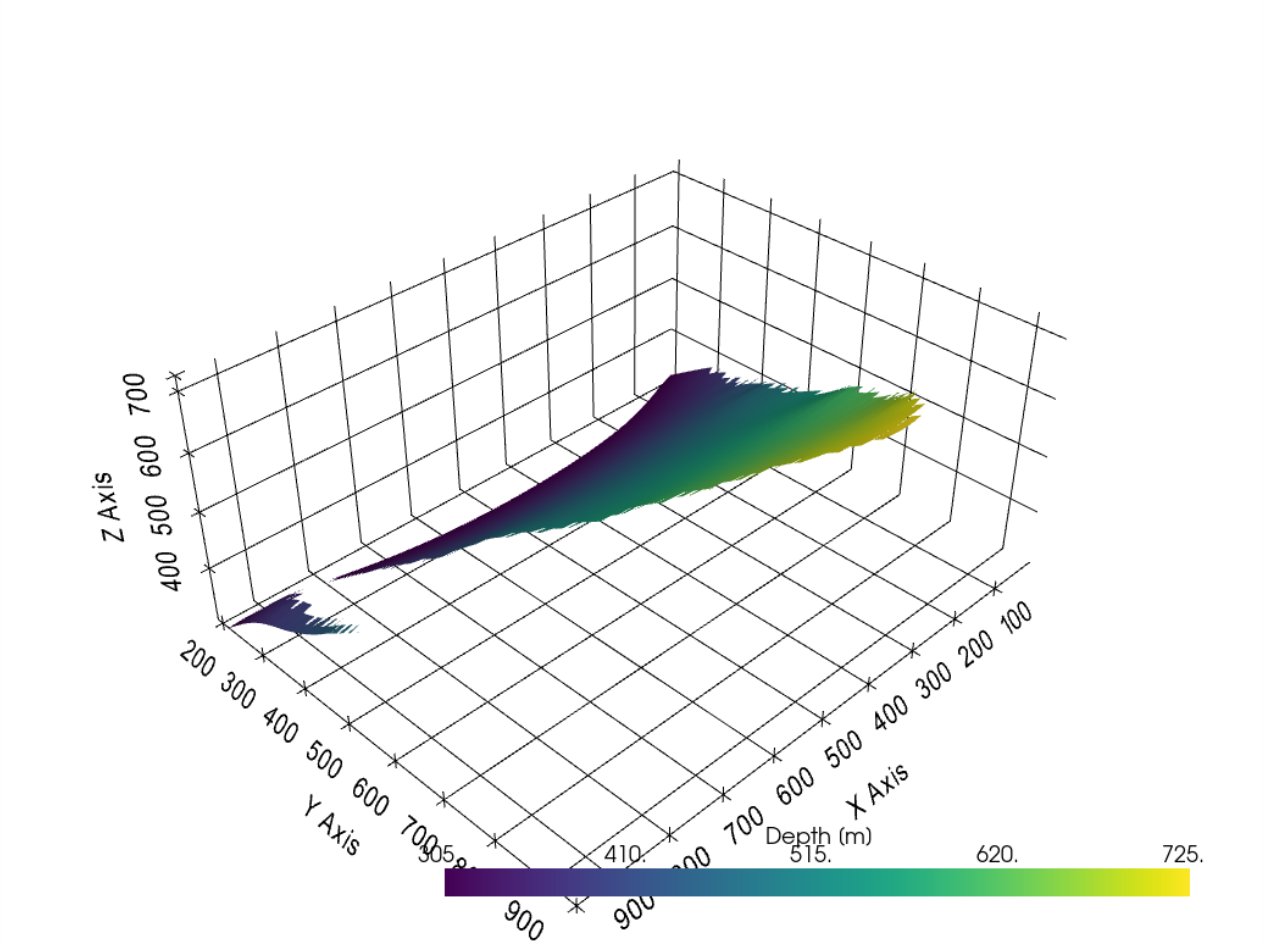
[19]:
```
dict_sand1['Sand1'][0]
```

[19]:
```
PolyData (0x24a0e712520)
  N Cells:  4174
  N Points: 2303
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 1
```

Plotting Depth Maps

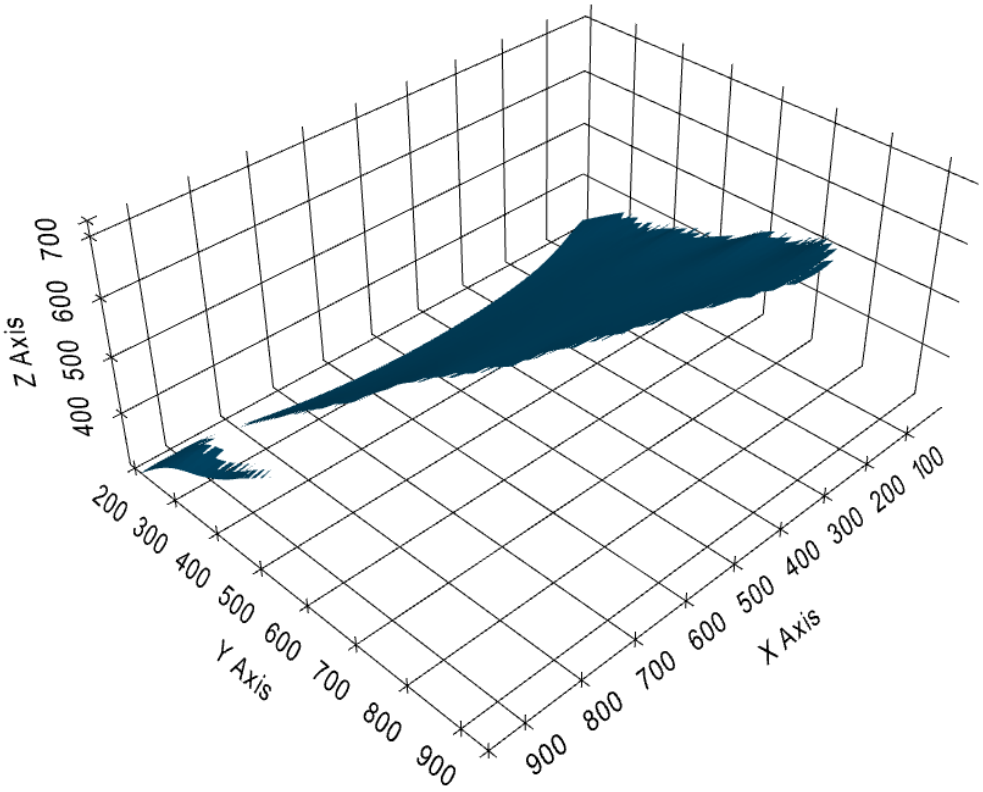
The depth maps can easily be plotted with PyVista.

[20]:
```
import pyvista as pv

p = pv.Plotter(notebook=True)

p.add_mesh(dict_sand1['Sand1'][0], scalars='Depth [m]')

p.set_background('white')
p.show_grid(color='black')
p.show()
```



If the depth is not needed, the original GemPy Model color can also be plotted.

[21]:
```
import pyvista as pv

p = pv.Plotter(notebook=True)

p.add_mesh(dict_sand1['Sand1'][0], color=dict_sand1['Sand1'][1])

p.set_background('white')
p.show_grid(color='black')
p.show()
```



By providing a list of surfaces, a dict containing the data for different surfaces is created.

[22]:
```
dict_all = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model,
                                                         surfaces=['Sand1', 'Ton'])

dict_all
```

[22]:
```
{'Sand1': [PolyData (0x24a134c9fa0)
    N Cells:    4174
    N Points:   2303
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   1.881e+02, 9.491e+02
    Z Bounds:   3.050e+02, 7.250e+02
    N Arrays:   1,
  '#015482'],
 'Ton': [PolyData (0x24a137dfa00)
    N Cells:    5111
    N Points:   2739
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   3.578e+02, 1.058e+03
    Z Bounds:   3.050e+02, 7.265e+02
    N Arrays:   1,
  '#9f0052']}
```

[23]:
```
dict_all['Sand1'][0]
```

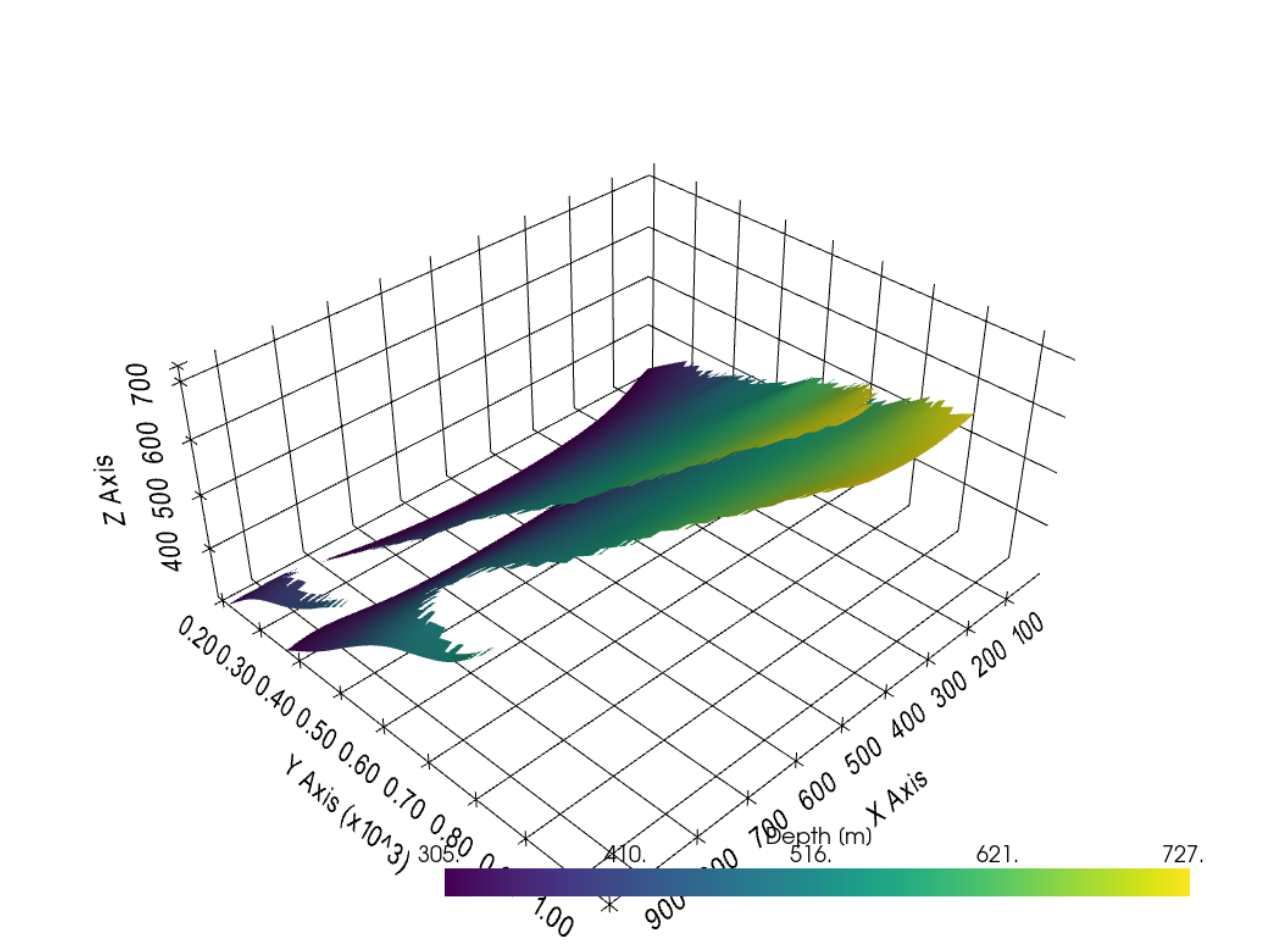
[23]:
```
PolyData (0x24a134c9fa0)
  N Cells:  4174
  N Points: 2303
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 1
```

[24]:
```
dict_all['Ton'][0]
```

[24]:
```
PolyData (0x24a137dfa00)
  N Cells:  5111
  N Points: 2739
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 3.578e+02, 1.058e+03
  Z Bounds: 3.050e+02, 7.265e+02
  N Arrays: 1
```

This data can be accessed as before to display both surfaces.

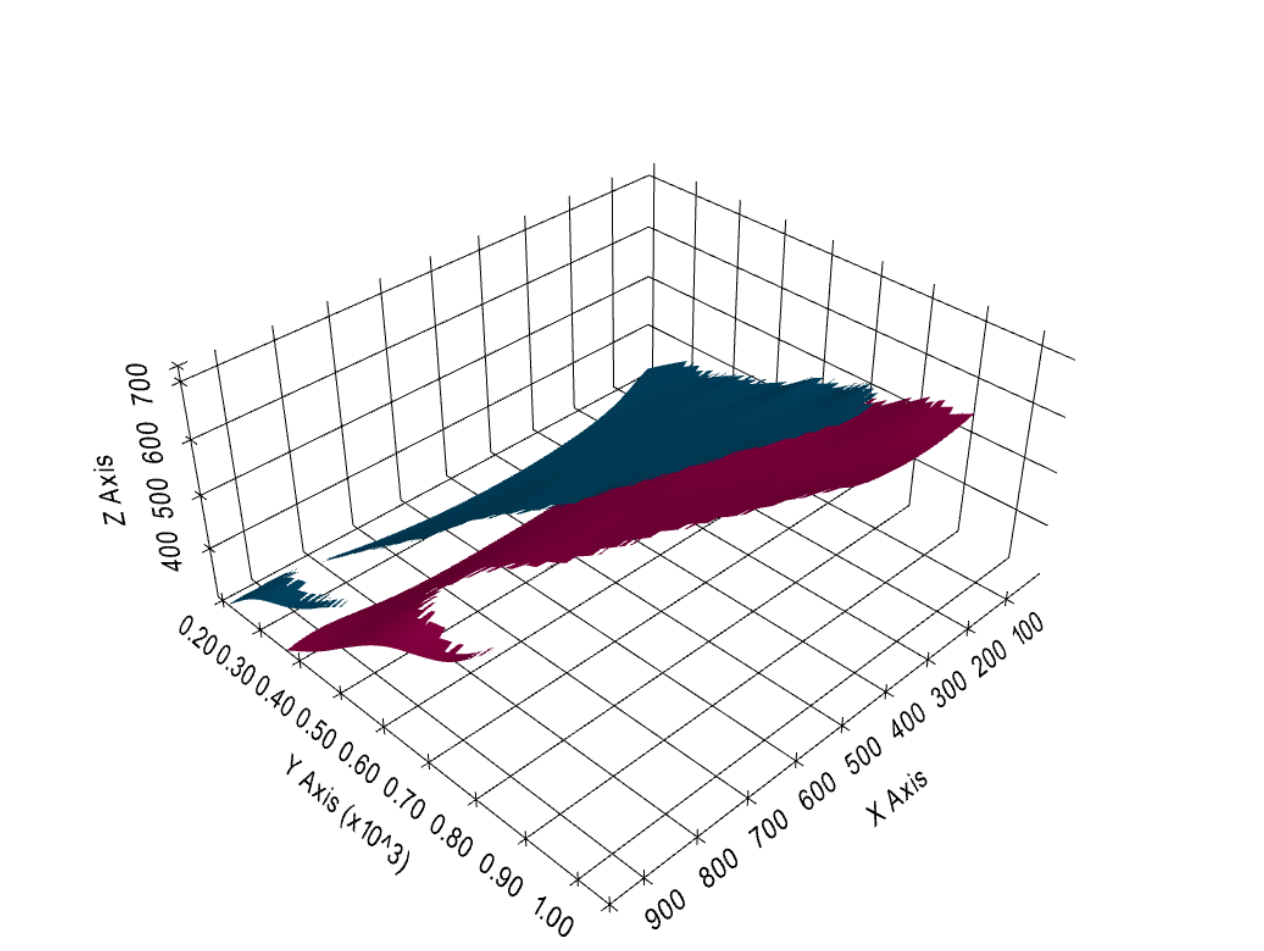
[25]:
```
import pyvista as pv

p = pv.Plotter(notebook=True)

p.add_mesh(dict_all['Sand1'][0], scalars='Depth [m]')
p.add_mesh(dict_all['Ton'][0], scalars='Depth [m]')

p.set_background('white')
p.show_grid(color='black')
p.show()
```



In a same way, the original colors can be used for plotting.

[26]:
```
dict_all['Sand1'][0].save(file_path+'Layer1.vtk')
dict_all['Ton'][0].save(file_path+'Layer2.vtk')
```

[27]:
```
import pyvista as pv

p = pv.Plotter(notebook=True)

p.add_mesh(dict_all['Sand1'][0], color=dict_all['Sand1'][1])
p.add_mesh(dict_all['Ton'][0], color=dict_all['Ton'][1])

p.set_background('white')
p.show_grid(color='black')
p.show()
```



Creating depth maps from PyVista Meshes

[28]:
```
dict_all['Sand1'][0]
```

[28]:
```
PolyData (0x24a134c9fa0)
  N Cells:  4174
  N Points: 2303
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 1
```

Clearing the data arrays.

[29]:
```
dict_all['Sand1'][0].clear_arrays()
dict_all['Sand1'][0]
```

```
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\pyvista\core\dataset.py:1560:␣
↪PyvistaDeprecationWarning: Use of `clear_arrays` is deprecated. Use `clear_data`␣
↪instead.
  warnings.warn(
```

[29]:
```
PolyData (0x24a134c9fa0)
  N Cells:  4174
  N Points: 2303
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 0
```

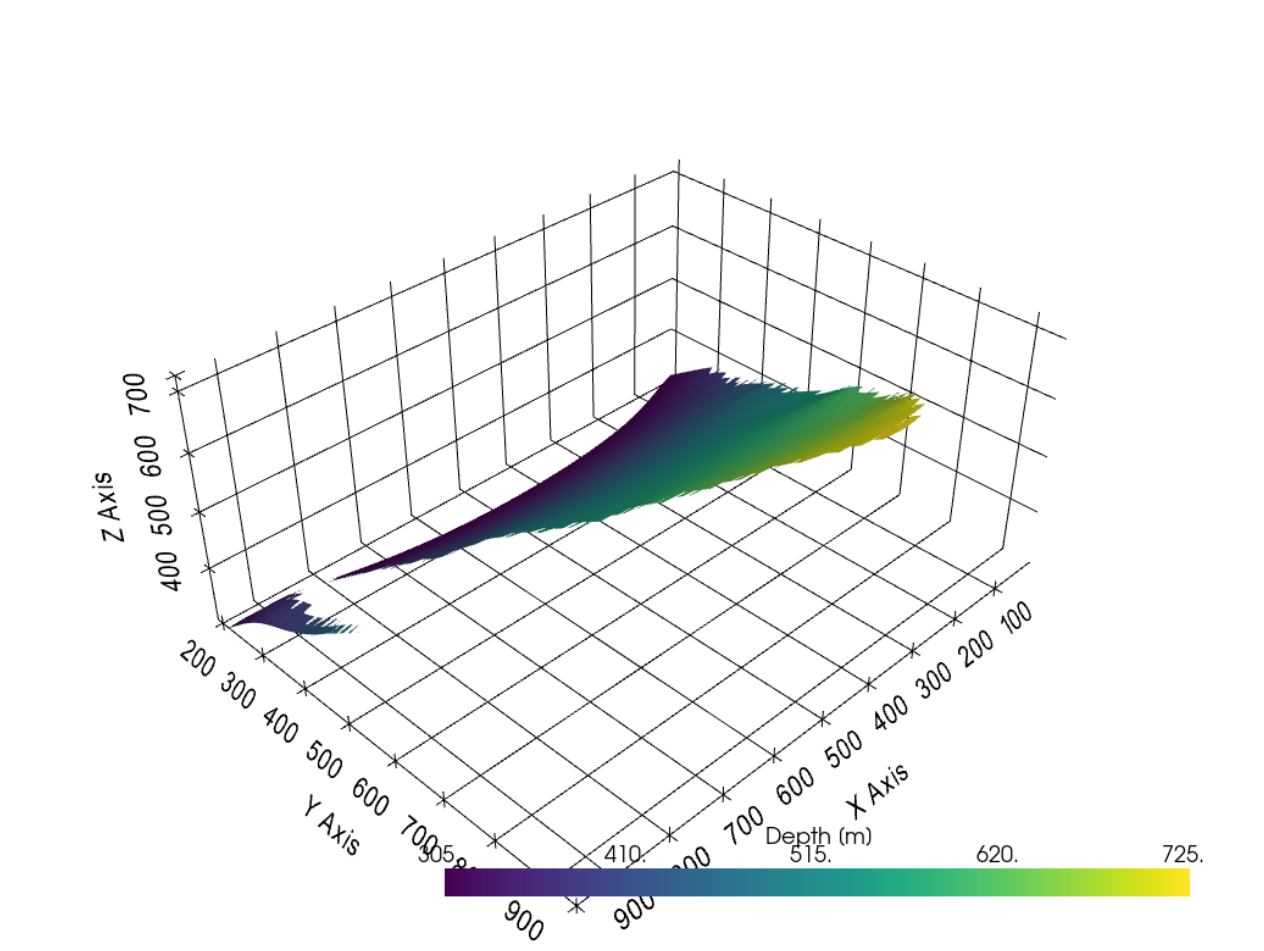
Extracting the depth information.

[30]:
```
dict_all['Sand1'][0] = gg.visualization.create_depth_map(mesh=dict_all['Sand1'][0], name=
↪'Depth [m]')
dict_all['Sand1'][0]
```

[30]:
```
PolyData (0x24a134c9fa0)
  N Cells:  4174
  N Points: 2303
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 1
```

Plotting the mesh again.

[31]:
```
import pyvista as pv

p = pv.Plotter(notebook=True)

p.add_mesh(dict_all['Sand1'][0], scalars='Depth [m]')

p.set_background('white')
p.show_grid(color='black')
p.show()
```



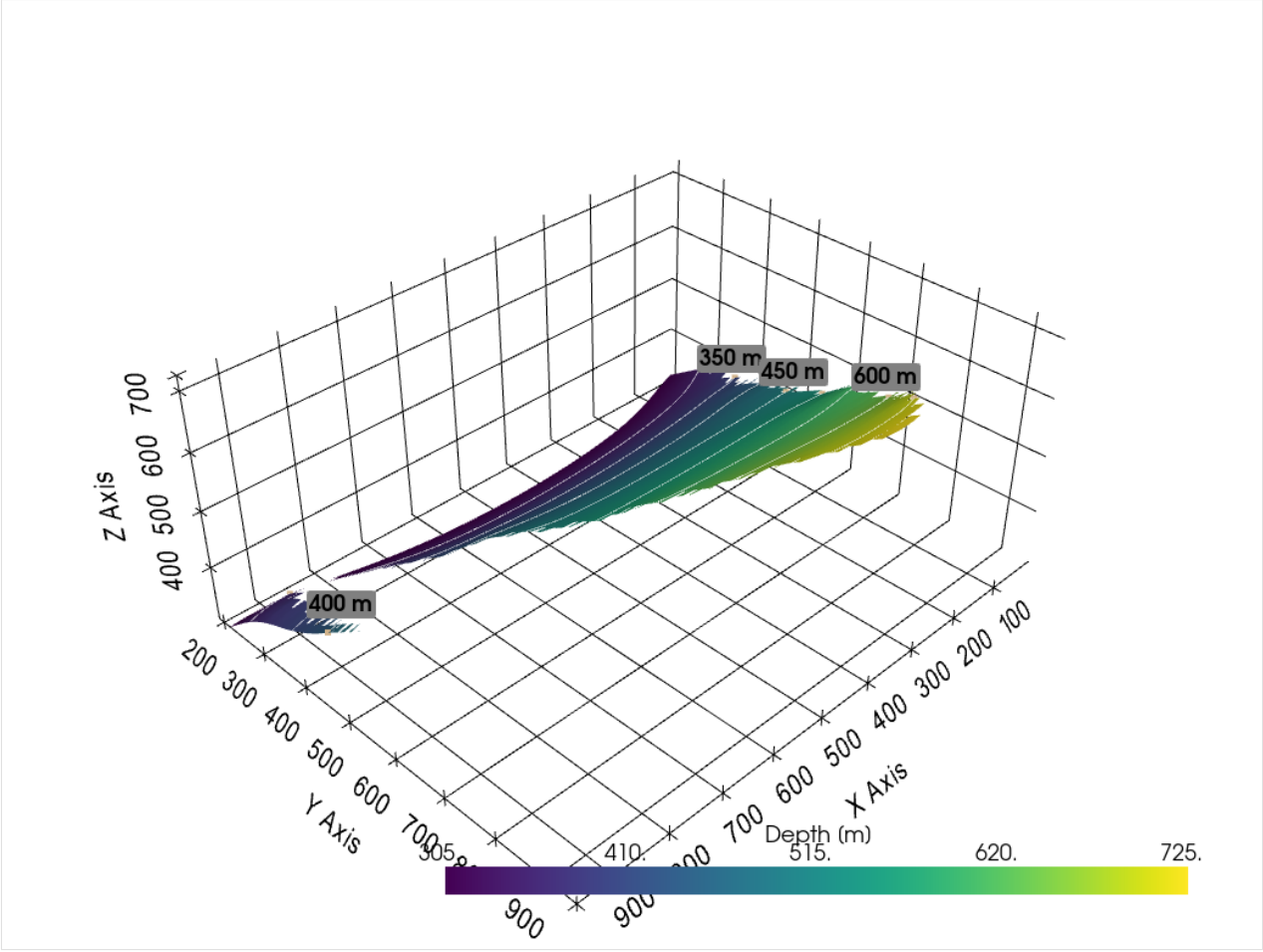
Plotting labels to contour lines

[47]:
```
import numpy as np
contours = dict_all['Sand1'][0].contour(np.arange(300,750, 50))

levels = contours.split_bodies()
pts = []
values = []

for level in levels:
    pt = level.points[0]
    v = level['Depth [m]'][0]
    pts.append(pt)
    values.append(v)

p = pv.Plotter(notebook=True)

p.add_mesh(dict_all['Sand1'][0], scalars='Depth [m]')
p.add_point_labels(np.array(pts), [f'{v:.0f} m' for v in values])
p.add_mesh(contours, color='white')
p.set_background('white')
```

(continues on next page)

(continued from previous page)

```
p.show_grid(color='black')
p.show()
```



6.20 19 Working with Web Map Services - WMS

A Web Map Service (WMS) is a standard protocol developed by the Open Geospatial Consortium in 1999 for serving georeferenced map images over the Internet. These images are typically produced by a map server from data provided by a GIS database.

The map information can be downloaded as map data and stored as array for further usage.

Source: https://en.wikipedia.org/wiki/Web_Map_Service

6.20.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[2]:
```
import gemgis as gg

file_path ='data/19_working_with_web_map_services/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.20.2 Loading the Web Map Service

Loading the Web Map Service from https://ows.terrestris.de/.

[2]:
```
wms = gg.web.load_wms(url='https://ows.terrestris.de/osm/service?')
wms
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
```

(continues on next page)

(continued from previous page)

```
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
<owslib.map.wms111.WebMapService_1_1_1 at 0x261a7a250d0>
```

Checking the type of the WMS

[3]:
```
wms.identification.type
```

[3]:
```
'OGC:WMS'
```

Checking the version of the WMS

[4]:
```
wms.identification.version
```

[4]:
```
'1.1.1'
```

Checking the title of the WMS

[5]:
```
wms.identification.title
```

[5]:
```
'OpenStreetMap WMS'
```

Checking the abstract of the WMS

[6]:
```
wms.identification.abstract
```

[6]:
```
'OpenStreetMap WMS, bereitgestellt durch terrestris GmbH und Co. KG. Beschleunigt mit␣
↪MapProxy (http://mapproxy.org/)'
```

Checking the operations of the WMS

[7]:
```
wms.getOperationByName('GetMap').methods
```

[7]:
```
[{'type': 'Get', 'url': 'https://ows.terrestris.de/osm/service?'}]
```

Checking the format options of the WMS

[8]:
```
# The different formats a layer can be saved as
wms.getOperationByName('GetMap').formatOptions
```

[8]:
```
['image/jpeg', 'image/png']
```

Checking the title of a WMS layer

[9]:
```
# Title of a layer
wms['OSM-WMS'].title
```

[9]:
```
'OpenStreetMap WMS - by terrestris'
```

Checking the CRS options of a WMS layer

[10]:
```
# Available CRS systems for a layer
wms['OSM-WMS'].crsOptions
```

[10]:
```
['EPSG:25833',
 'EPSG:4326',
 'EPSG:29192',
 'EPSG:32648',
 'EPSG:5243',
 'EPSG:4674',
 'EPSG:4686',
 'EPSG:900913',
 'EPSG:2056',
 'EPSG:31466',
 'EPSG:4258',
 'EPSG:2180',
 'EPSG:2100',
 'EPSG:21781',
 'EPSG:29193',
 'EPSG:31463',
 'EPSG:3068',
 'EPSG:31467',
 'EPSG:4647',
 'EPSG:4839',
 'EPSG:31468',
 'EPSG:3857',
 'EPSG:25832',
 'EPSG:3034',
 'EPSG:3035']
```

Checking the styles of the WMS Layer

[11]:
```
# Available styles
wms['OSM-WMS'].styles
```

[11]:
```
{'default': {'title': 'default',
  'legend': 'https://ows.terrestris.de/osm/service?styles=&layer=OSM-WMS&service=WMS&
↪format=image%2Fpng&sld_version=1.1.0&request=GetLegendGraphic&version=1.1.1'}}
```

Checking the bounding box of the WMS layer

[12]:
```
wms['OSM-WMS'].boundingBox
```

[12]:
```
(-20037508.3428, -25819498.5135, 20037508.3428, 25819498.5135, 'EPSG:900913')
```

Checking the bounding box of the WMS layer

[13]:
```
wms['OSM-WMS'].boundingBoxWGS84
```

[13]:
```
(-180.0, -88.0, 180.0, 88.0)
```

[14]:
```
wms['OSM-WMS'].opaque
```

[14]:
```
0
```

[15]:
```
wms['OSM-WMS'].queryable
```

[15]:
```
1
```

Getting map data for the WMS layer

[16]:
```
wms_map = gg.web.load_as_map(url=wms.url,
                             layer='OSM-WMS',
                             style='default',
                             crs='EPSG:4647',
                             bbox=[32286000,32328000, 5620000,5648000],
                             size=[4200, 2800],
                             filetype='image/png')
```

[17]:
```
wms_map
```

[17]:
```
<owslib.util.ResponseWrapper at 0x261d348cc10>
```

Converting the map data to an array

[18]:
```
# Converting WMS map object to array
import io
import matplotlib.pyplot as plt

maps = io.BytesIO(wms_map.read())
wms_array = plt.imread(maps)
wms_array[:1]
```

[18]:
```
array([[[0.8039216 , 0.7647059 , 0.65882355],
        [0.85882354, 0.8784314 , 0.6627451 ],
        [0.87058824, 0.91764706, 0.6666667 ],
        ...,
        [0.78431374, 0.7647059 , 0.65882355],
        [0.8862745 , 0.9019608 , 0.81960785],
        [0.9529412 , 0.93333334, 0.9019608 ]]], dtype=float32)
```

Getting the array data for the WMS layer

This layer shows the OpenStreet Map data for the Aachen area.

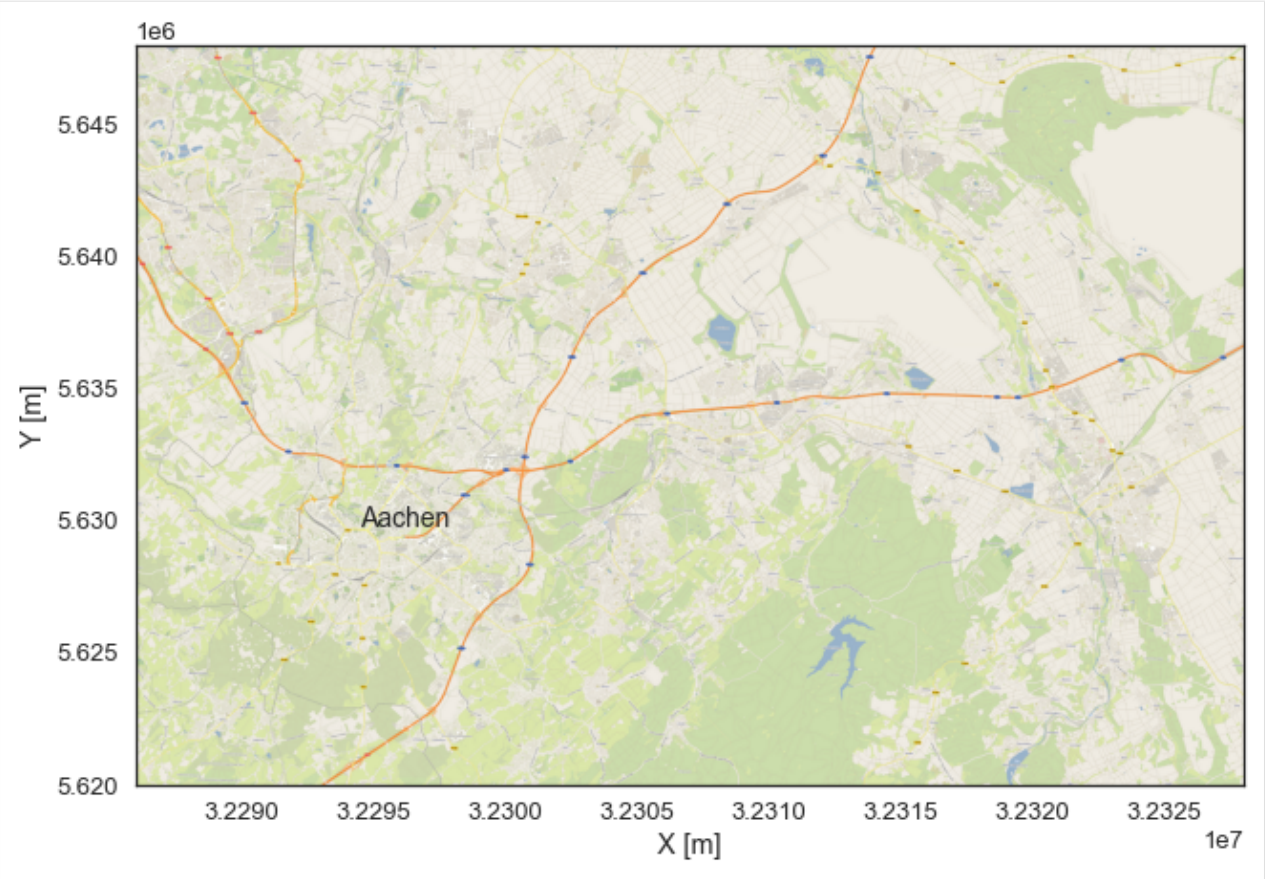
[19]:
```
wms_array = gg.web.load_as_array(url=wms.url,
                                 layer='OSM-WMS',
                                 style='default',
                                 crs='EPSG:4647',
                                 bbox=[32286000,32328000, 5620000,5648000],
                                 size=[4200, 2800],
                                 filetype='image/png')

wms_array[:1]
```

[19]:
```
array([[[0.8039216 , 0.7647059 , 0.65882355],
        [0.85882354, 0.8784314 , 0.6627451 ],
        [0.87058824, 0.91764706, 0.6666667 ],
        ...,
        [0.78431374, 0.7647059 , 0.65882355],
        [0.8862745 , 0.9019608 , 0.81960785],
        [0.9529412 , 0.93333334, 0.9019608 ]]], dtype=float32)
```

Plotting the map data

[20]:
```
plt.imshow(wms_array, extent=[32286000, 32328000,5620000,5648000])
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.text(32294500,5629750, 'Aachen', size = 14)
```

[20]:
```
Text(32294500, 5629750, 'Aachen')
```



Getting map data for the WMS layer

This layer shows the hillshades of the Aachen area together with a colored digital elevation model.

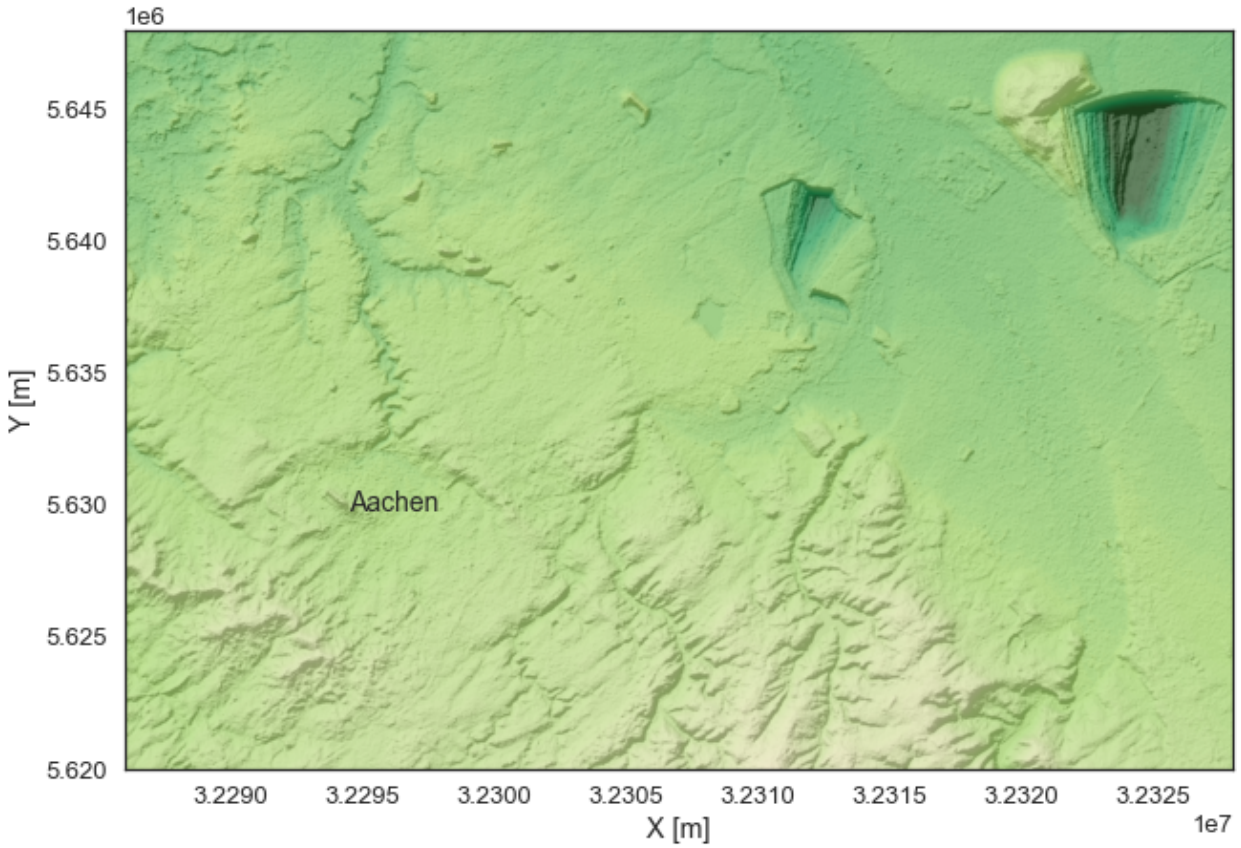
[21]:
```
wms_array = gg.web.load_as_array(url=wms.url,
                                 layer='SRTM30-Colored-Hillshade',
                                 style='default',
                                 crs='EPSG:4647',
                                 bbox=[32286000,32328000, 5620000,5648000],
                                 size=[4200, 2800],
                                 filetype='image/png')

wms_array[:1]
```

[21]:
```
array([[[0.56078434, 0.77254903, 0.5137255 , 1.        ],
        [0.54901963, 0.7607843 , 0.5019608 , 1.        ],
        [0.5568628 , 0.76862746, 0.5058824 , 1.        ],
        ...,
        [0.49411765, 0.7490196 , 0.5058824 , 1.        ],
        [0.49411765, 0.7490196 , 0.5058824 , 1.        ],
        [0.49803922, 0.7529412 , 0.50980395, 1.        ]]], dtype=float32)
```

Plotting the map data

[22]:
```
plt.imshow(wms_array, extent=[32286000, 32328000,5620000,5648000])
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.text(32294500,5629750, 'Aachen', size = 14)
```

[22]:
```
Text(32294500, 5629750, 'Aachen')
```



Checking the data

[23]:
```
wms_array[:1]
```

[23]:
```
array([[[0.56078434, 0.77254903, 0.5137255 , 1.        ],
        [0.54901963, 0.7607843 , 0.5019608 , 1.        ],
        [0.5568628 , 0.76862746, 0.5058824 , 1.        ],
        ...,
        [0.49411765, 0.7490196 , 0.5058824 , 1.        ],
        [0.49411765, 0.7490196 , 0.5058824 , 1.        ],
        [0.49803922, 0.7529412 , 0.50980395, 1.        ]]], dtype=float32)
```

Loading the WMS

The next layer represents orthophotos of the Aachen area.

[24]:
```
wms = gg.web.load_wms('https://www.wms.nrw.de/geobasis/wms_nw_dop')
wms
```

[24]:
```
<owslib.map.wms111.WebMapService_1_1_1 at 0x261d53bed60>
```

Getting map data for the WMS layer

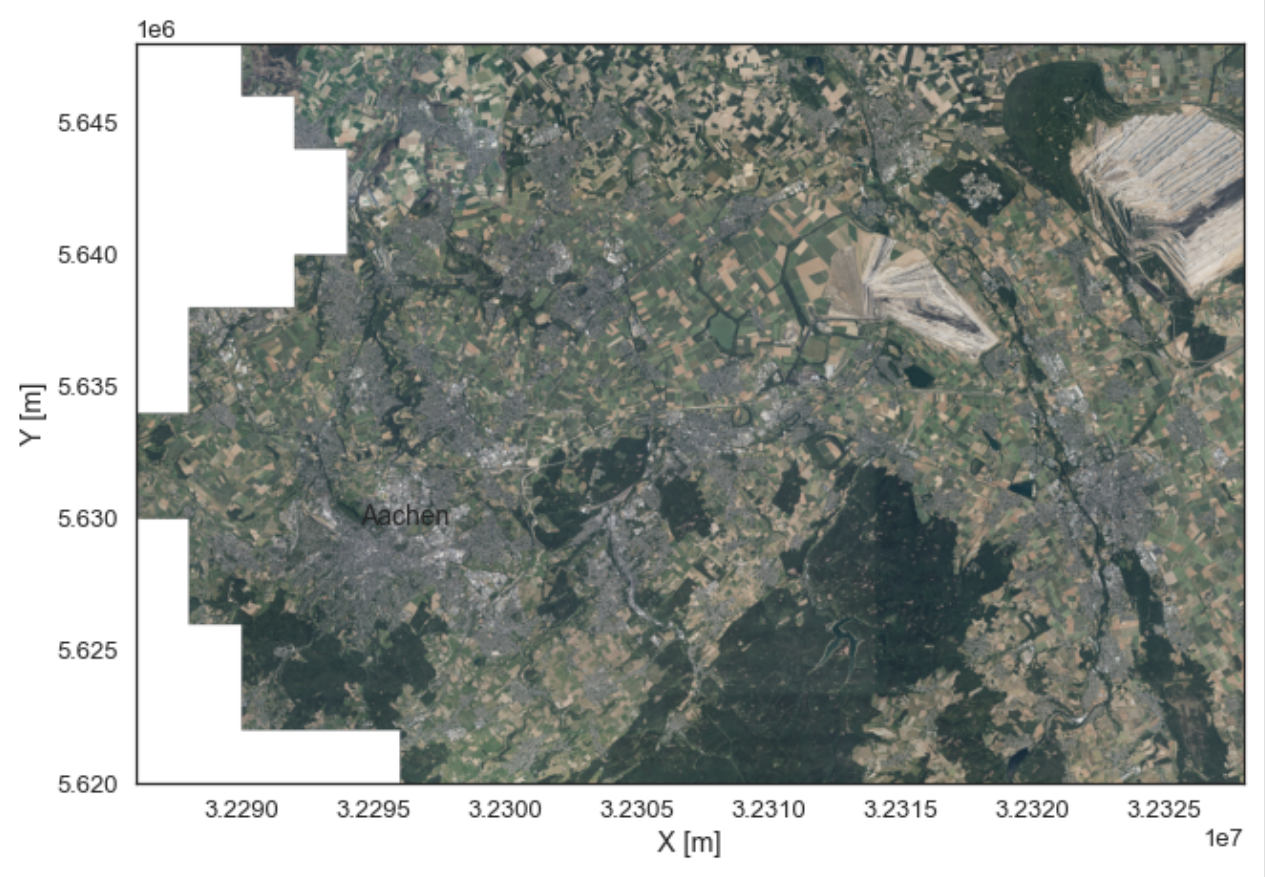
[25]:
```
wms_array = gg.web.load_as_array(url=wms.url,
                                 layer='nw_dop_rgb',
                                 style='default',
                                 crs='EPSG:4647',
                                 bbox=[32286000,32328000, 5620000,5648000],
                                 size=[4200, 2800],
                                 filetype='image/png')

wms_array[:1]
```

[25]:
```
array([[[0.        , 0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        ],
        [0.        , 0.        , 0.        , 0.        ],
        ...,
        [0.29803923, 0.33333334, 0.3137255 , 1.        ],
        [0.3019608 , 0.33333334, 0.32941177, 1.        ],
        [0.2901961 , 0.3019608 , 0.2901961 , 1.        ]]], dtype=float32)
```

Plotting the map data

[26]:
```
plt.imshow(wms_array, extent=[32286000, 32328000,5620000,5648000])
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.text(32294500,5629750, 'Aachen', size = 14)
```

[26]:
```
Text(32294500, 5629750, 'Aachen')
```



Loading the WMS

The next layer is the geological map of the Aachen area.

[27]:
```
wms = gg.web.load_wms('http://www.wms.nrw.de/gd/GK100')
wms
```

[27]:
```
<owslib.map.wms111.WebMapService_1_1_1 at 0x261d53d8280>
```

Getting map data for the WMS layer

[28]:
```
wms_array = gg.web.load_as_array(url=wms.url,
                                 layer='0',
                                 style='default',
                                 crs='EPSG:4647',
                                 bbox=[32286000,32328000, 5620000,5648000],
                                 size=[1400,933],
                                 filetype='image/png')

wms_array[:1]
```

[28]:
```
array([[[0.9882353 , 0.87058824, 0.87058824, 1.        ],
        [0.9882353 , 0.87058824, 0.87058824, 1.        ],
```

(continues on next page)

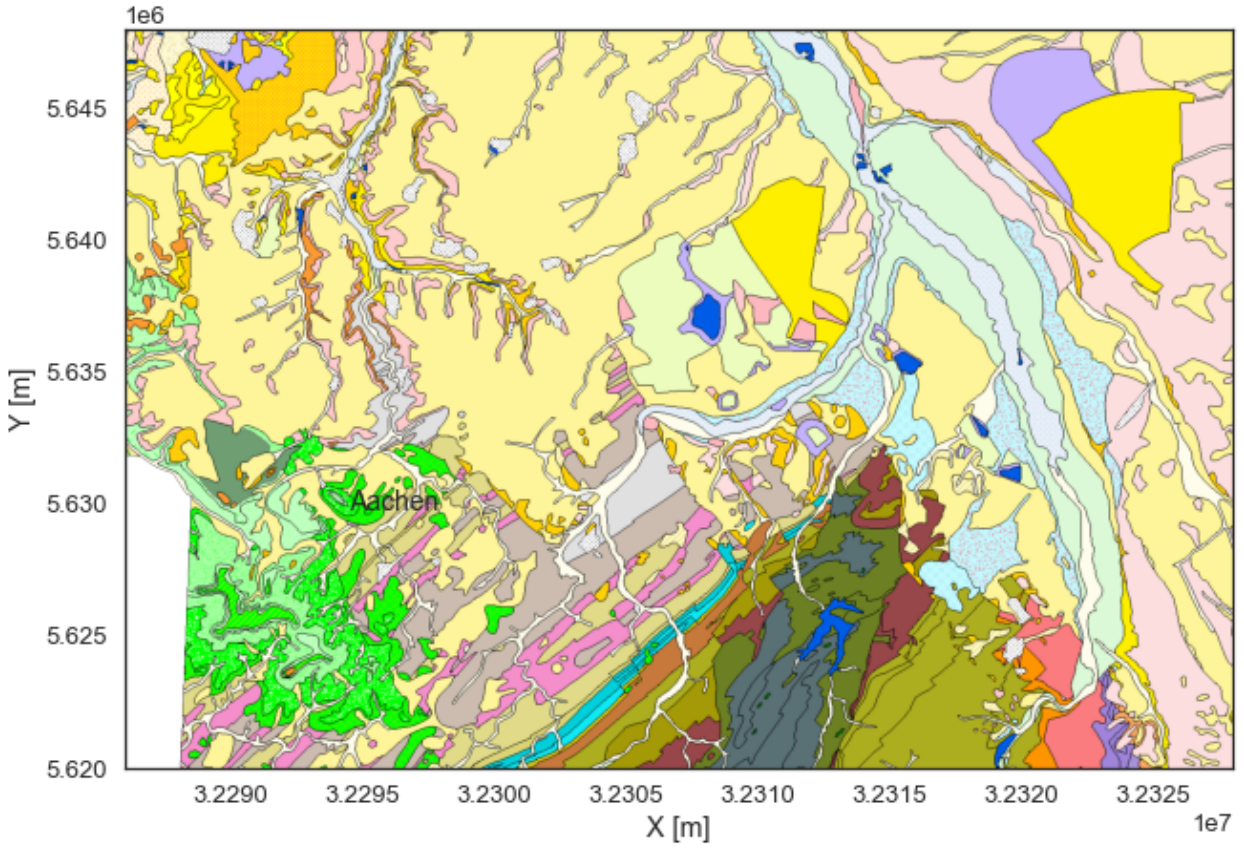
(continued from previous page)

```
        [0.9882353 , 0.87058824, 0.87058824, 1.        ],
        ...,
        [0.99607843, 0.9607843 , 0.6039216 , 1.        ],
        [0.99607843, 0.9607843 , 0.6039216 , 1.        ],
        [0.99607843, 0.9607843 , 0.6039216 , 1.        ]]], dtype=float32)
```

Plotting the map data

[29]:
```
plt.imshow(wms_array, extent=[32286000, 32328000,5620000,5648000])
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.text(32294500,5629750, 'Aachen', size = 14)
```

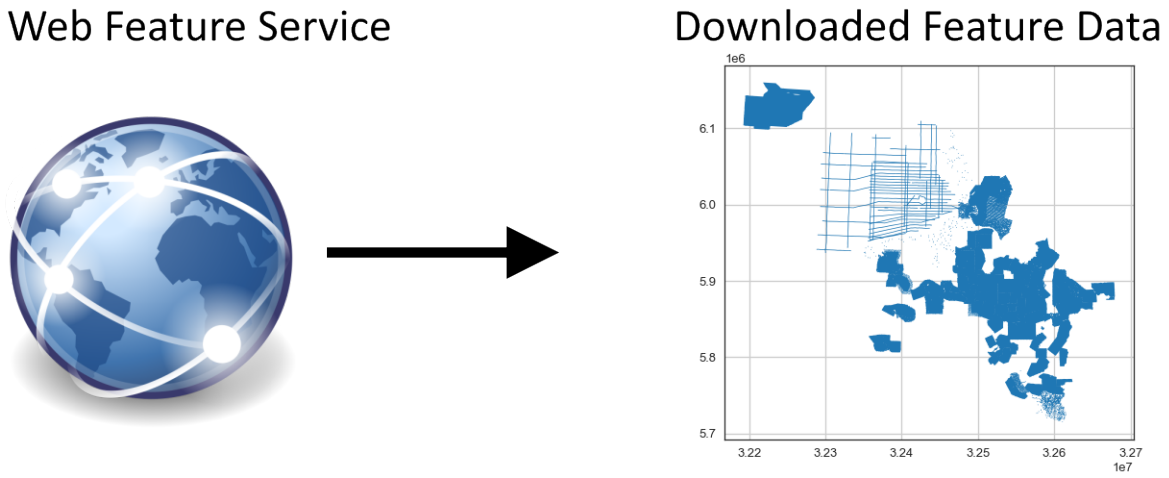
[29]:
```
Text(32294500, 5629750, 'Aachen')
```



6.21 20 Working with Web Feature Services

In computing, the Open Geospatial Consortium Web Feature Service (WFS) Interface Standard provides an interface allowing requests for geographical features across the web using platform-independent calls. One can think of geographical features as the "source code" behind a map, whereas the WMS interface or online tiled mapping portals like Google Maps return only an image, which end-users cannot edit or spatially analyze. The XML-based GML furnishes the default payload-encoding for transporting geographic features, but other formats like shapefiles can also serve for transport.

Source: https://en.wikipedia.org/wiki/Web_Feature_Service



6.21.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/20_working_with_web_feature_services/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.21.2 Loading the WFS Service

For this example, we are downloading a WFS Service provided by the *Landesamt für Bergbau, Energie und Geologie - Niedersachsen*. It contains Polygon, Line and Point data about seismic surveys and wells in Northern Germany.

[3]:
```
wfs = gg.web.load_wfs("https://nibis.lbeg.de/net3/public/ogc.ashx?NodeId=476&Service=WFS&
↪")
wfs
```

[3]:
```
<owslib.feature.wfs100.WebFeatureService_1_0_0 at 0x19260e21340>
```

[3]:
```
type(wfs)
```

[3]:
```
owslib.feature.wfs100.WebFeatureService_1_0_0
```

Checking the version of the WFS

[4]:
```
wfs.version
```

[4]:
```
'1.0.0'
```

Checking the version of the WFS

[5]:
```
wfs.identification.version
```

[5]:
```
'1.0.0'
```

Checking the identification type of the WFS

[6]:
```
wfs.identification.type
```

[6]:
```
'Geophysik und Tiefohrungen'
```

Checking the title of the WFS

[7]:
```
wfs.identification.title
```

[7]:
```
'Geophysik und Tiefohrungen'
```

Checking the abstract of the WFS

[8]:
```
wfs.identification.abstract
```

[8]:
```
'Geophysik und Tiefohrungen'
```

Checking the contents of the WFS

[9]:
```
list(wfs.contents)
```
[9]:
```
['iwan:L383']
```

Checking the title of the WFS layer

[10]:
```
wfs['iwan:L383'].title
```
[10]:
```
'Gravimetrie'
```

Checking the bounding box of the WFS layer

[11]:
```
wfs['iwan:L383'].boundingBoxWGS84
```
[11]:
```
(5.395175801132899, 47.16510247399335, 17.002272548448747, 54.853980760069O2)
```

Checking the operations for the WFS

[12]:
```
[op.name for op in wfs.operations]
```
[12]:
```
['GetCapabilities', 'DescribeFeatureType', 'GetFeature']
```

Checking the format for getting the feature

[13]:
```
wfs.getOperationByName('GetFeature').formatOptions
```
[13]:
```
['{http://www.opengis.net/wfs}GML2']
```

Checking the format for describing the feature

[14]:
```
wfs.getOperationByName('DescribeFeatureType').formatOptions
```
[14]:
```
[]
```

Checking the format for getting the capabilities

[15]:
```
wfs.getOperationByName('GetCapabilities').formatOptions
```
[15]:
```
[]
```

Requesting data from the WFS

The features can be loaded with the function `load_as_gpd(..)`.

[16]:
```
feature = gg.web.load_as_gpd("https://nibis.lbeg.de/net3/public/ogc.ashx?NodeId=476&
↪Service=WFS&")
```

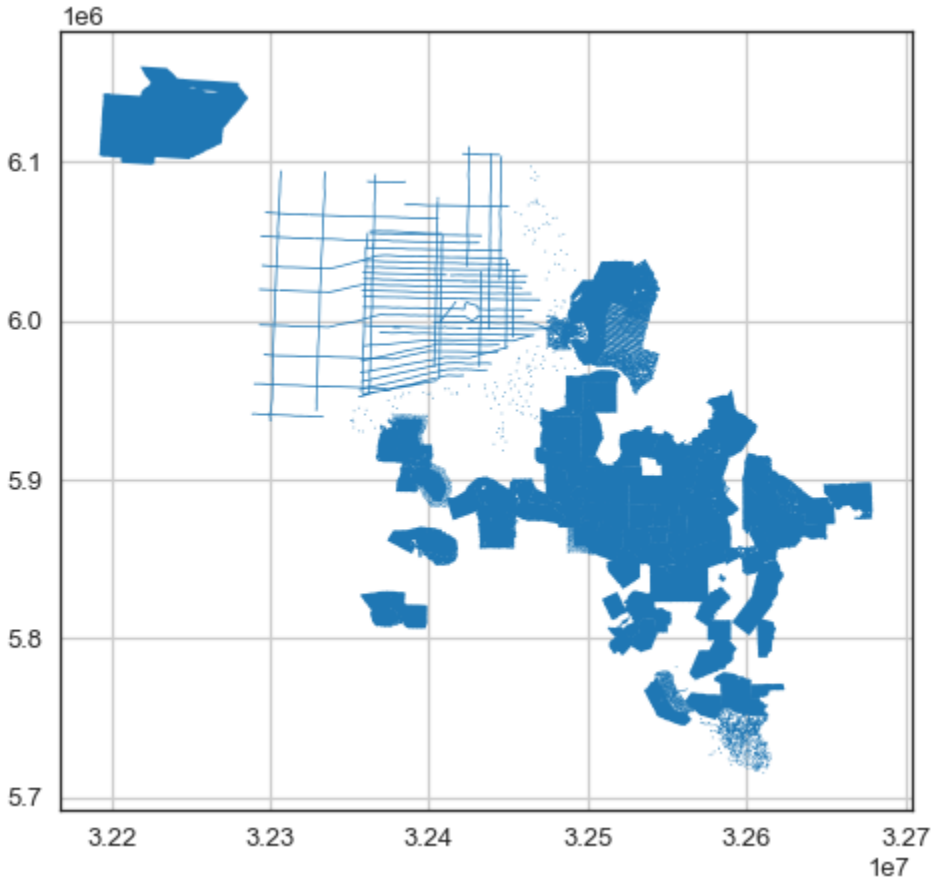
Checking the head of the data

[17]:
```
feature.head()
```

[17]:
```
  gml_id  OBJECTID   ID              SURVEYNAME   ARCHIV  MESSJAHR  \
0   1541      1541  112             Jemgum 2007  0127494      2007
1   1542      1542  111          Sagermeer 2005  0125831      2005
2   1543      1543  120           Hümmling 2013  0131520      2013
3   1544      1544   49  Rotenburg/Wümme (1984)  0112162      1984
4   1545      1545  102        Odisheim 1989/90  0112092      1989

                                    OPERATOR  \
0  GdF Produktion Exploration Deutschland GmbH
1       ExxonMobil Production Deutschland GmbH
2                GDF SUEZ E&P DEUTSCHLAND GMBH
3       Deutsche Texaco AG Aufschluss und Gew.
4                    BEB Erdgas und Erdöl GmbH

                              OP_NACHFOL  \
0         Neptune Energy Deutschland GmbH
1  ExxonMobil Production Deutschland GmbH
2         Neptune Energy Deutschland GmbH
3          Wintershall Dea Deutschland AG
4      BEB Erdgas und Erdöl GmbH & Co. KG

                              MESSFIRMA  MESSPUNKTE  \
0  Geophysik und Geotechnik Leipzig GmbH        1340
1             Comp. Generale Geophysique        2803
2       Deutsche Montan Technologie GmbH         342
3                         Prakla Seismos        2184
4                         Prakla Seismos        2824

                     UP_DATE  \
0  2020-01-20T00:00:00+01:00
1  2020-01-20T00:00:00+01:00
2  2020-01-20T00:00:00+01:00
3  2020-01-20T00:00:00+01:00
4  2020-01-20T00:00:00+01:00

                                          geometry
0  MULTIPOLYGON (((32395246.839 5907777.660, 3239...
1  MULTIPOLYGON (((32446717.522 5856710.088, 3244...
2  MULTIPOLYGON (((32416592.825 5852885.544, 3241...
3  POLYGON ((32545955.921 5896473.525, 32544536.2...
4  POLYGON ((32504255.640 5967591.298, 32503387.8...
```

Plotting the data

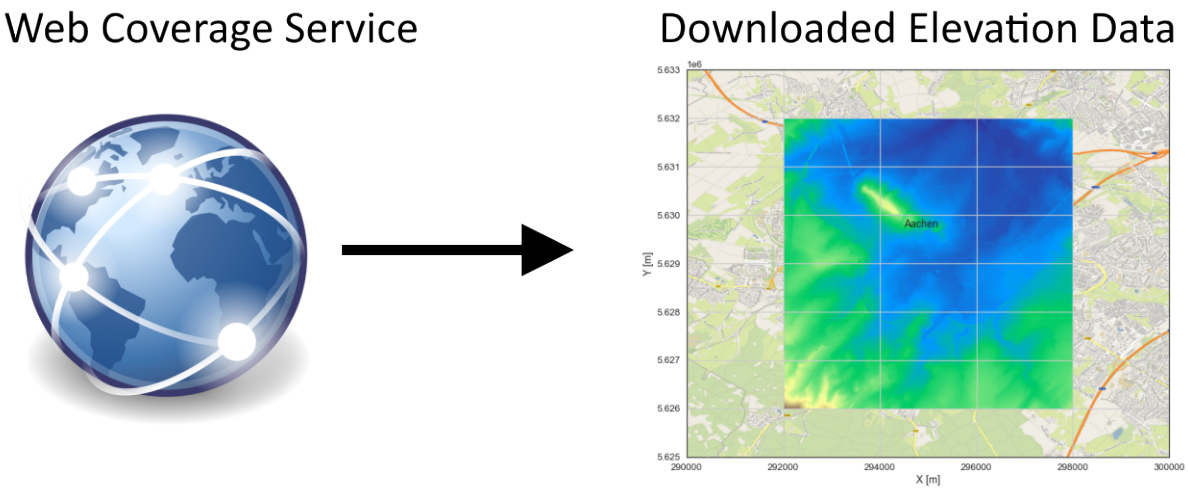
[18]:
```
import matplotlib.pyplot as plt

feature.plot()
plt.grid()
```



6.22 21 Working with Web Coverage Services

The Web Coverage Service (WCS) is a standard issued by the Open Geospatial Consortium (OGC). It is designed to simplify remote access to coverages, commonly known as raster maps in GIS. WCS functions over the HTTP protocol, setting out how to obtain data and meta-data using the requests available in that protocol. In practice it allows raster maps to be obtained from a web browser or from any other programme that uses the protocol.

Source: https://www.isric.org/web-coverage-services-wcs



6.22.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/21_working_with_web_coverage_services/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.22.2 Loading the WCS Service and loading WMS Data

For this example, the digital elevation model of the State of Northrhine Westfalia will be downloaded. More information and the license for using the data can be found here.

Loading the WCS Service

The WCS Server is being accessed via OWSLib. The attributes `url` and `version` are needed for the following request.

[2]:
```
wcs_url = 'https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'
wcs = gg.web.load_wcs(url=wcs_url)
print(type(wcs))
wcs
```

```
<class 'owslib.coverage.wcs201.WebCoverageService_2_0_1'>
```

[2]:
```
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27d3d027a00>
```

[3]:
```
wcs.url
```

[3]:
```
'https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'
```

[4]:
```
wcs.version
```

[4]:
```
'2.0.1'
```

Load WMS Layer and Map

A map of the Aachen area is loaded to later on plot the downloaded Digital Elevation Model data.

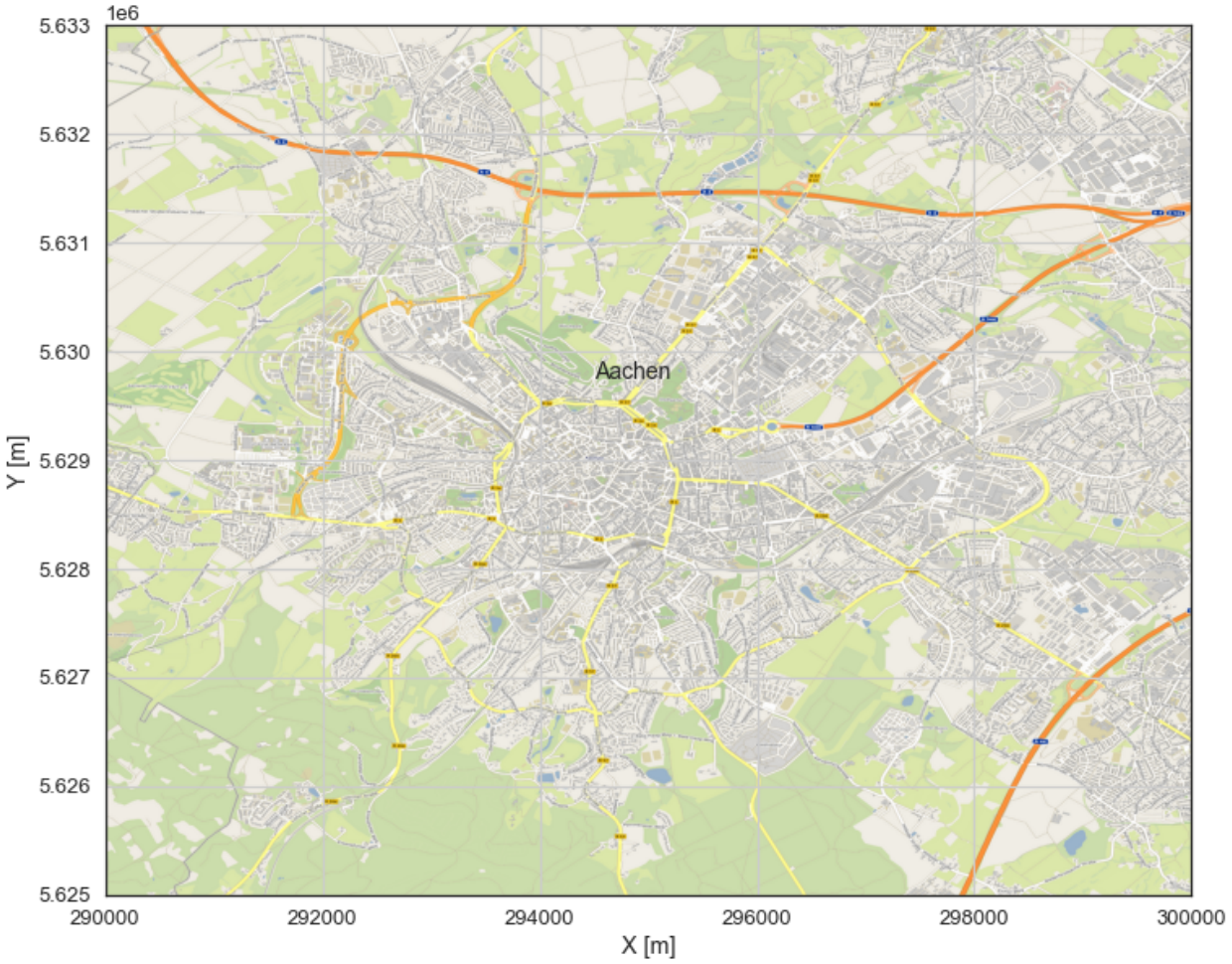
[5]:
```
wms = gg.web.load_wms('https://ows.terrestris.de/osm/service?')
```

[6]:
```
wms_map = gg.web.load_as_array('https://ows.terrestris.de/osm/service?',
                               'OSM-WMS', 'default', 'EPSG:25832', [290000, 300000,5624000,
↪5634000], [2000, 2000], 'image/png')
```

Plot WMS Data

[7]:
```
import matplotlib.pyplot as plt

plt.figure(figsize = (12,12))
plt.imshow(wms_map, extent= [290000, 300000,5625000,5633000])
plt.grid()
plt.ylabel('Y [m]')
plt.xlabel('X [m]')
plt.text(294500,5629750, 'Aachen', size = 14)
```

[7]:
```
Text(294500, 5629750, 'Aachen')
```



6.22.3 Creating and Executing WCS Request

The WCS needs to be created by providing min and max values for X and Y locations. Here, a for loop is created to automatically download four tiles with an extent of 2 by 2 km each. Due to their size, they will be saved outside the repository.

[8]:
```
gg.web.load_as_files(wcs_url=wcs.url,
                     version=wcs.version,
                     identifier='nw_dgm',
                     form='image/tiff',
                     extent=[292000, 298000, 5626000, 5632000],
                     size=2000,
                     path=file_path,
                     create_directory=True)
```

```
  0%|
↪         | 0/3 [00:00<?, ?it/s]
```

```
Extent X:  6000  m
Extent Y:  6000  m
Number of tiles in X directions:  3
```

(continues on next page)

(continued from previous page)

```
Number of tiles in Y directions:  3
Total Number of Tiles:  9
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(292000,294000)&SUBSET=y(5626000,5628000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(292000,294000)&SUBSET=y(5628000,5630000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(292000,294000)&SUBSET=y(5630000,5632000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
 33%|                                                    | 1/3 [00:04<00:09,  4.59s/
↪it]
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(294000,296000)&SUBSET=y(5626000,5628000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(294000,296000)&SUBSET=y(5628000,5630000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(294000,296000)&SUBSET=y(5630000,5632000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
 67%|                              | 2/3 [00:09<00:04,  4.64s/it]
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(296000,298000)&SUBSET=y(5626000,5628000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(296000,298000)&SUBSET=y(5628000,5630000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
https://www.wcs.nrw.de/geobasis/wcs_nw_dgm?REQUEST=GetCoverage&SERVICE=WCS&VERSION=2.0.1&
↪COVERAGEID=nw_dgm&FORMAT=image/tiff&SUBSET=x(296000,298000)&SUBSET=y(5630000,5632000)&
↪OUTFILE=data/21_working_with_web_coverage_services/
100%|| 3/3 [00:13<00:00,  4.67s/it]
```

6.22.4 Create List of File paths

A list of file paths is created for the subsequent merging of the data.

[9]:
```
dem_fps = gg.raster.create_filepaths(dirpath=file_path,
                                     search_criteria='tile*.tif')
dem_fps[:4]
```

[9]:
```
['C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\21_
↪working_with_web_coverage_services\\tile_292000_294000_5626000_5628000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\21_
↪working_with_web_coverage_services\\tile_292000_294000_5628000_5630000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\21_
↪working_with_web_coverage_services\\tile_292000_294000_5630000_5632000.tif',
 'C:\\Users\\ale93371\\Documents\\gemgis\\docs\\getting_started\\tutorial\\data\\21_
↪working_with_web_coverage_services\\tile_294000_296000_5626000_5628000.tif']
```

(continues on next page)

(continued from previous page)

6.22.5 Create List of Tiles

The above created list of file paths is automatically being created when executing the function below. In addition, a list of the loaded tiles is created.

[10]:
```
src_files_to_mosaic = gg.raster.create_src_list(filepaths=dem_fps)
src_files_to_mosaic[:15]
```

[10]:
```
[<open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_292000_294000_5626000_5628000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_292000_294000_5628000_5630000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_292000_294000_5630000_5632000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_294000_296000_5626000_5628000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_294000_296000_5628000_5630000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_294000_296000_5630000_5632000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_296000_298000_5626000_5628000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_296000_298000_5628000_5630000.
↪tif' mode='r'>,
 <open DatasetReader name='C:/Users/ale93371/Documents/gemgis/docs/getting_started/
↪tutorial/data/21_working_with_web_coverage_services/tile_296000_298000_5630000_5632000.
↪tif' mode='r'>]
```

6.22.6 Merge tiles to mosaic

The single files can now automatically be merged to form a mosaic. In addition, the transform of the raster is being returned to save the raster later on.

[11]:
```
mosaic, transform = gg.raster.merge_tiles(src_files=src_files_to_mosaic)
```

[12]:
```
mosaic
```

[12]:
```
array([[200.72, 200.73, 200.72, ..., 204.42, 204.45, 204.45],
       [200.74, 200.74, 200.75, ..., 204.43, 204.44, 204.48],
       [200.76, 200.76, 200.76, ..., 204.42, 204.48, 204.5 ],
       ...,
```

(continues on next page)

(continued from previous page)

```
       [329.15, 328.86, 328.74, ..., 242.45, 242.38, 242.28],
       [329.29, 329.06, 328.87, ..., 242.45, 242.39, 242.31],
       [329.47, 329.3 , 329.09, ..., 242.42, 242.37, 242.32]],
      dtype=float32)
```

[13]:
```
transform
```

[13]:
```
Affine(1.0, 0.0, 292000.0,
       0.0, -1.0, 5632000.0)
```

6.22.7 Safe Raster to disc

[14]:
```
import numpy as np
gg.raster.save_as_tiff(raster=np.flipud(mosaic),
                       path=file_path +'raster.tif',
                       extent=[292000,298000,5626000,5632000],
                       crs='EPSG:25832',
                       transform=transform,
                       overwrite_file=True)
```

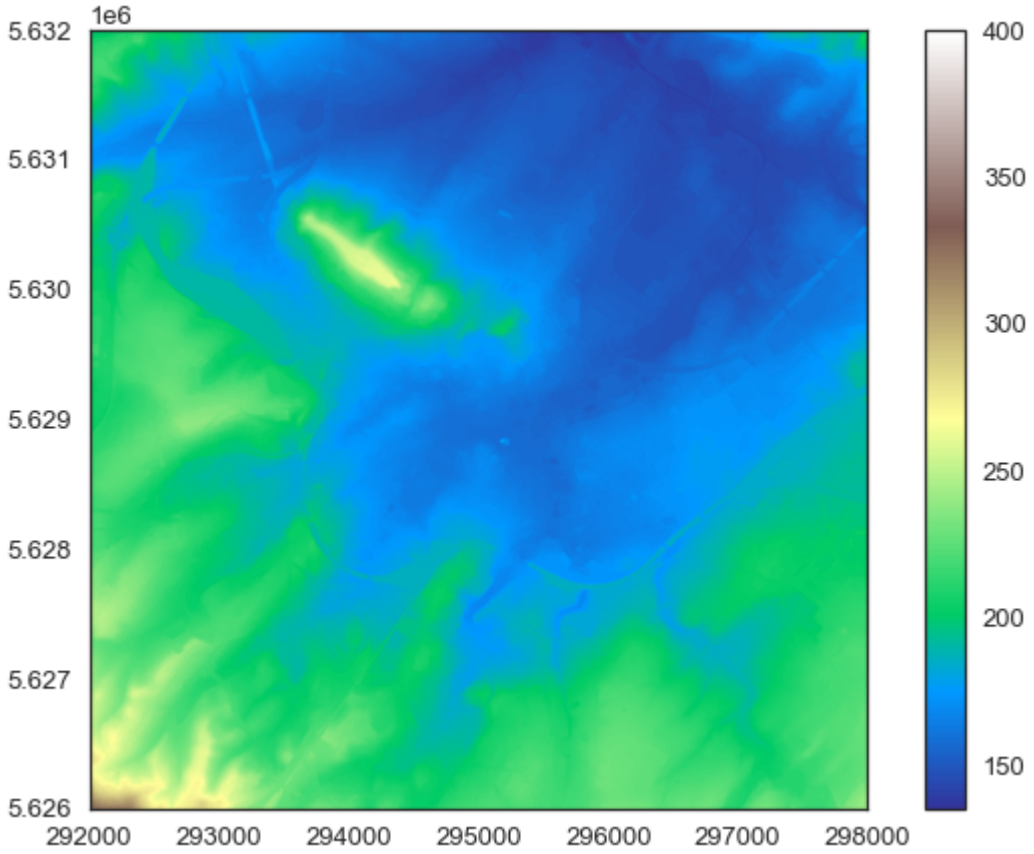
```
Raster successfully saved
```

6.22.8 Plot DEM

The mosaic/DEM can now be plotted using the built-in rasterio functionality or using matplotlib.

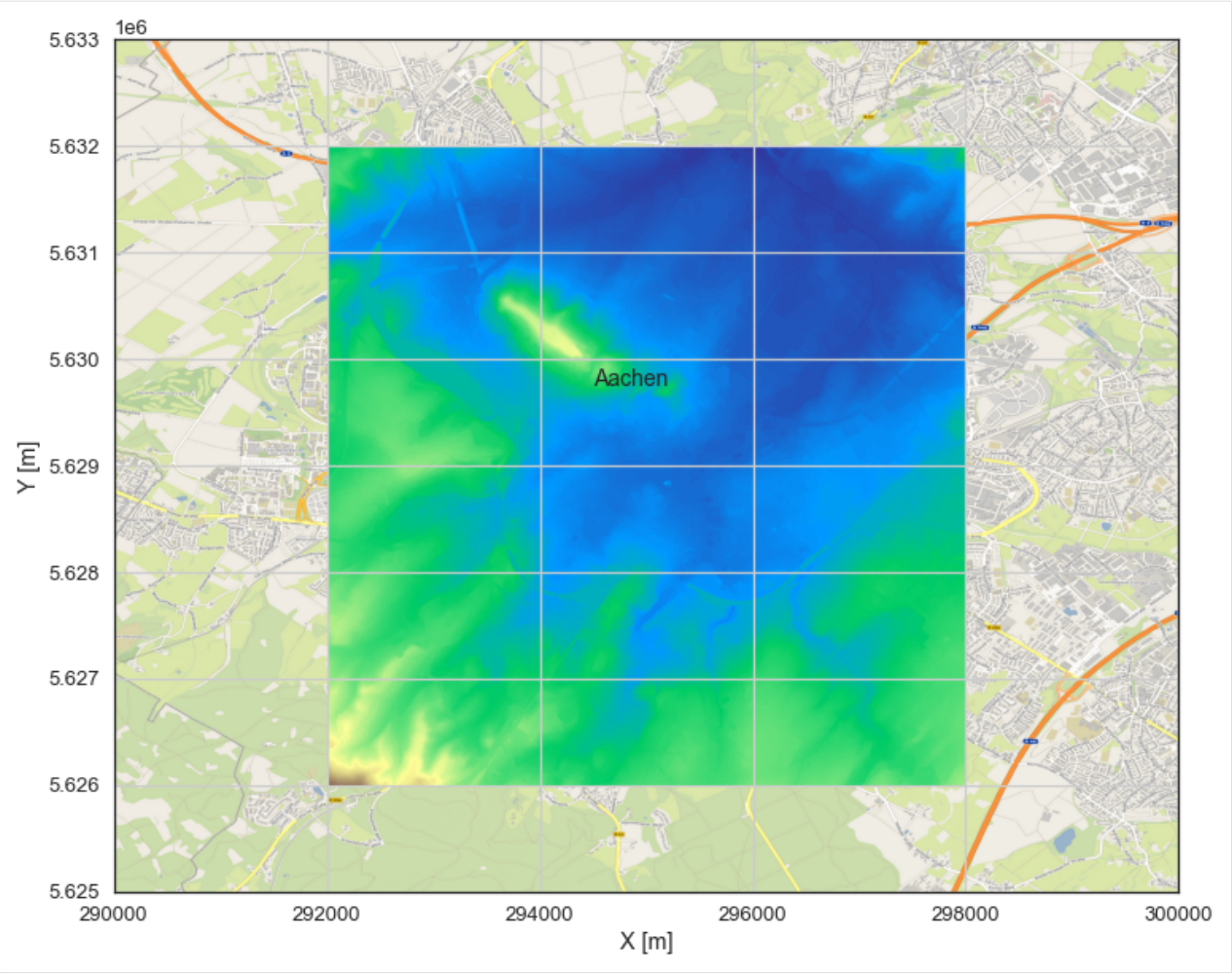
[15]:
```
im = plt.imshow(mosaic, cmap='terrain', vmax=400, extent = [292000,298000,5626000,
↪5632000])
plt.colorbar(im)
```

[15]:
```
<matplotlib.colorbar.Colorbar at 0x27d3d1a3760>
```



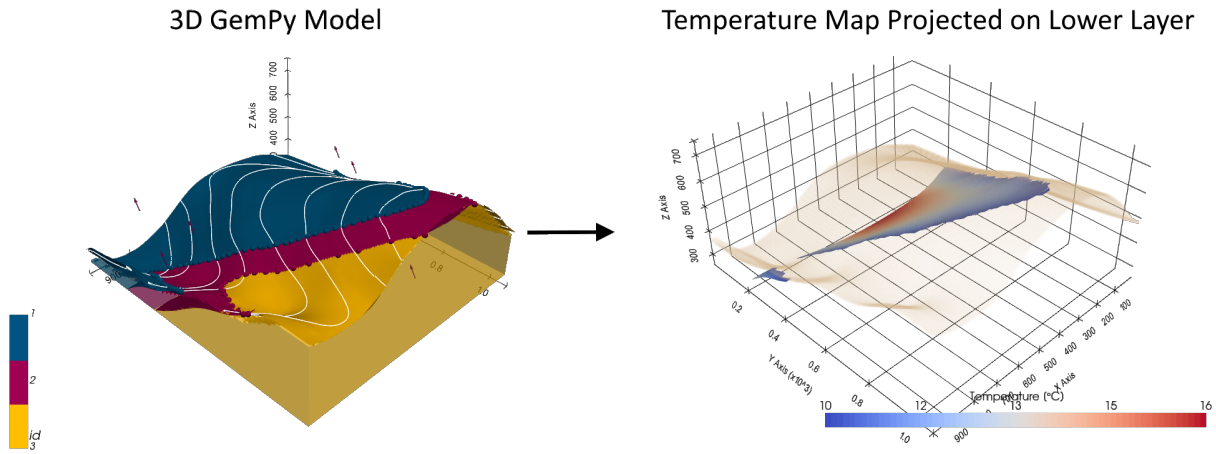
[16]:
```
plt.figure(figsize = (12,12))
plt.imshow(wms_map, extent= [290000, 300000,5625000,5633000])
plt.grid()
plt.ylabel('Y [m]')
plt.xlabel('X [m]')
plt.text(294500,5629750, 'Aachen', size = 14)
im = plt.imshow(mosaic, cmap='terrain', vmax=400, extent = [292000,298000,5626000,
↪5632000])
plt.xlim(290000, 300000)
plt.ylim(5625000,5633000)
```

[16]:
```
(5625000.0, 5633000.0)
```



6.23 22 Creating Temperature Maps from GemPy Models

Temperature maps are an important tool for Geologists to visualize the temperature distribution of layers at depth. A simple model (Example 1) is created of which temperature maps for the two existing layers are created. The temperature will be calculated as function of the thickness between a given topography and the distance to the layer at depth.



6.23.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/22_creating_temperature_maps_from_gempy_models/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="22_creating_temperature_maps_
↪from_gempy_models.zip", dirpath=file_path)
```

6.23.2 Loading the data

[2]:
```
import geopandas as gpd
import rasterio

interfaces = gpd.read_file(file_path + 'interfaces.shp')
orientations = gpd.read_file(file_path + 'orientations.shp')
extent = [0,972,0,1069, 300, 800]
resolution = [50, 50, 50]
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[3]:
```
interfaces.head()
```

[3]:

	level_0	level_1	formation	X	Y	Z	geometry
0	0	0	Sand1	0.26	264.86	353.97	POINT (0.256 264.862)
1	0	0	Sand1	10.59	276.73	359.04	POINT (10.593 276.734)
2	0	0	Sand1	17.13	289.09	364.28	POINT (17.135 289.090)
3	0	0	Sand1	19.15	293.31	364.99	POINT (19.150 293.313)
4	0	0	Sand1	27.80	310.57	372.81	POINT (27.795 310.572)

[4]:
```
orientations['polarity'] = 1
orientations.head()
```

[4]:

	formation	dip	azimuth	X	Y	Z	geometry	polarity
0	Ton	30.50	180.00	96.47	451.56	477.73	POINT (96.471 451.564)	1
1	Ton	30.50	180.00	172.76	661.88	481.73	POINT (172.761 661.877)	1
2	Ton	30.50	180.00	383.07	957.76	444.45	POINT (383.074 957.758)	1
3	Ton	30.50	180.00	592.36	722.70	480.57	POINT (592.356 722.702)	1
4	Ton	30.50	180.00	766.59	348.47	498.96	POINT (766.586 348.469)	1

6.23.3 Creating the GemPy Model

[5]:
```
import sys
sys.path.append('../../../../gempy-master')
import gempy as gp
```

[6]:
```
geo_model = gp.create_model('Model1')
geo_model
```

[6]:
```
Model1  2021-01-02 13:32
```

Initiating the Model

[7]:
```
import pandas as pd

gp.init_data(geo_model, extent, resolution,
             surface_points_df = interfaces,
             orientations_df = orientations,
             default_values=True)
geo_model.surfaces
```

```
Active grids: ['regular']
```

[7]:
```
  surface          series  order_surfaces    color  id
0   Sand1  Default series               1  #015482   1
1     Ton  Default series               2  #9f0052   2
```

The vertices and edges are currently `NaN` values, so no model has been computed so far.

[8]:
```
geo_model.surfaces.df
```

[8]:
```
  surface          series  order_surfaces  isBasement isFault  isActive  \
0   Sand1  Default series               1       False   False      True
1     Ton  Default series               2        True   False      True

   hasData    color vertices edges sfai  id
0     True  #015482      NaN   NaN  NaN   1
1     True  #9f0052      NaN   NaN  NaN   2
```

Mapping Stack to Surfaces

[9]:
```
gp.map_stack_to_surfaces(geo_model,
                         {"Strat_Series": ('Sand1', 'Ton')},
                         remove_unused_series=True)
geo_model.add_surfaces('basement')
```

[9]:
```
    surface        series  order_surfaces    color  id
0     Sand1  Strat_Series               1  #015482   1
1       Ton  Strat_Series               2  #9f0052   2
2  basement  Strat_Series               3  #ffbe00   3
```

Loading Topography

[10]:
```
geo_model.set_topography(
    source='gdal', filepath='../../../../gemgis_data/data/22_creating_temperature_maps_
↪from_gempy_models/raster.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[10]:
```
Grid Object. Values:
array([[   9.72      ,   10.69      ,  305.        ],
       [   9.72      ,   10.69      ,  315.        ],
       [   9.72      ,   10.69      ,  325.        ],
       ...,
       [ 970.056     , 1059.28181818,  622.0892334 ],
       [ 970.056     , 1063.16909091,  622.06713867],
       [ 970.056     , 1067.05636364,  622.05786133]])
```

Setting Interpolator

[11]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range            1528.90
$C_o$           55655.83
drift equations      [3]
```

[11]:
```
<gempy.core.interpolator.InterpolatorModel at 0x201ed3abfa0>
```

Computing Model

[12]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

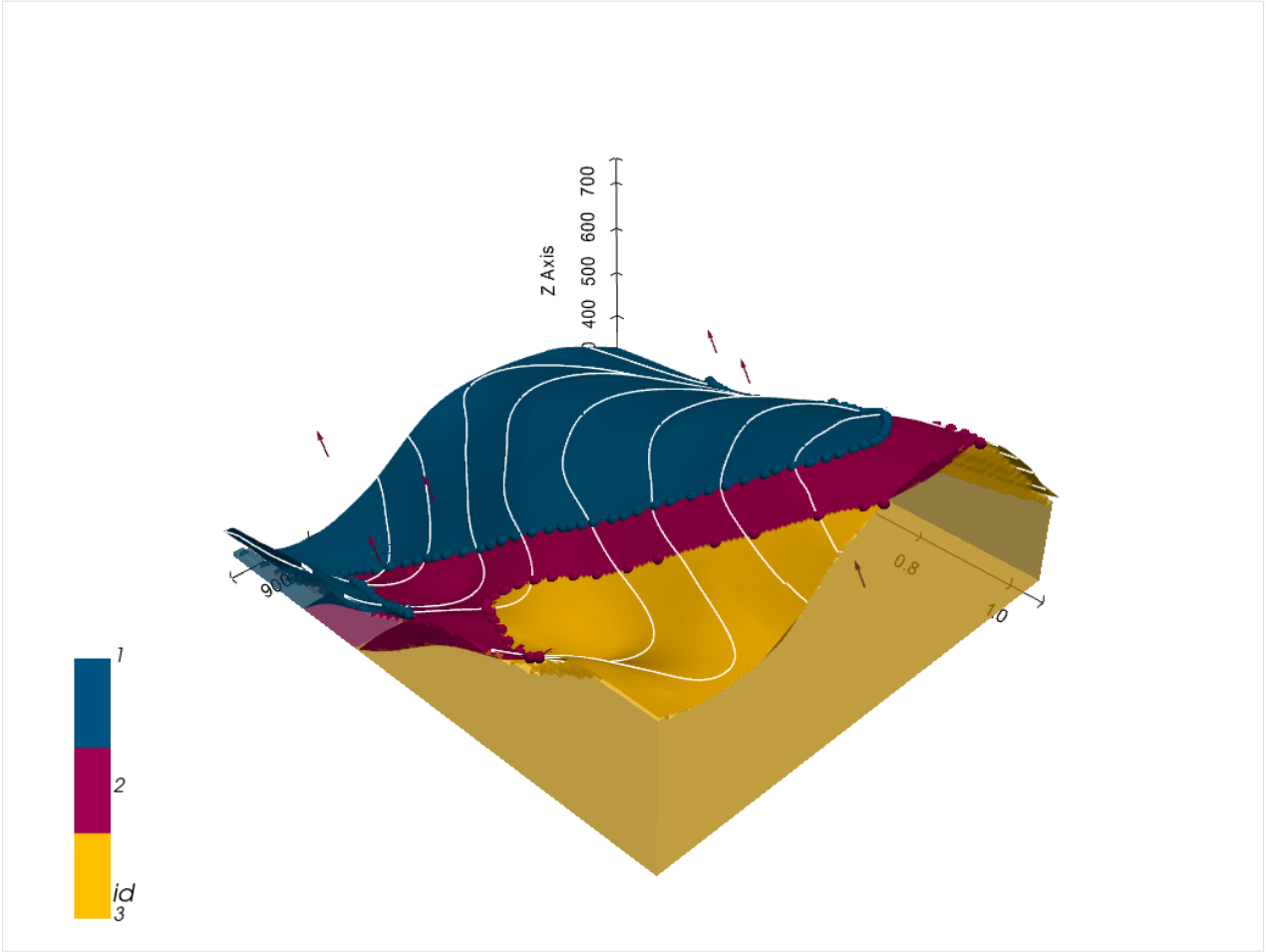
The surfaces DataFrame now contains values for vertices and edges.

[13]:
```
geo_model.surfaces.df
```

[13]:
```
    surface        series  order_surfaces  isBasement isFault  isActive  \
0     Sand1  Strat_Series               1       False   False      True
1       Ton  Strat_Series               2       False   False      True
2  basement  Strat_Series               3        True   False      True

   hasData    color                                              vertices  \
0     True  #015482  [[29.160000000000004, 194.27877317428587, 305...
1     True  #9f0052  [[29.160000000000004, 365.78652999877926, 305...
2     True  #ffbe00                                                NaN

                                              edges sfai  id
0  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.26   1
1  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.21   2
2                                                NaN  NaN   3
```

Plotting the 3D Model

[14]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



6.23.4 Creating Temperature Maps

Temperature Maps in GemGIS can be created by passing the Digital Elevation Model as a Rasterio Object and the surface in the subsurface for which the temperature is supposed to be calculated.

Creating Depth Maps

By providing a list of surfaces, a dict containing the data for different surfaces is created.

[17]:
```
dict_all = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model,
                                                         surfaces=['Sand1', 'Ton'])

dict_all
```

[17]:
```
{'Sand1': [PolyData (0x201ee61f640)
    N Cells:    4174
    N Points:   2303
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   1.881e+02, 9.491e+02
    Z Bounds:   3.050e+02, 7.250e+02
    N Arrays:   1,
```

(continues on next page)

(continued from previous page)

```
  '#015482'],
 'Ton': [PolyData (0x201fb4bf9a0)
   N Cells:    5111
   N Points:   2739
   X Bounds:   9.720e+00, 9.623e+02
   Y Bounds:   3.578e+02, 1.058e+03
   Z Bounds:   3.050e+02, 7.265e+02
   N Arrays:   1,
  '#9f0052']}
```

The temperature for the Sand Surface is now being calculated.

[22]:
```
mesh = dict_all['Sand1'][0]
mesh.clear_arrays()
mesh
```

[22]:
```
PolyData (0x201ee61f640)
  N Cells:  4174
  N Points: 2303
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 0
```

Coordinates of the vertices.

[23]:
```
mesh.points
```

[23]:
```
pyvista_ndarray([[ 29.16      , 194.27877317, 305.        ],
                 [  9.72      , 188.12027063, 305.        ],
                 [  9.72      , 203.11      , 314.39133763],
                 ...,
                 [958.49513855, 545.19      , 495.        ],
                 [962.28      , 546.6037711 , 495.        ],
                 [962.28      , 545.19      , 494.12916183]])
```

Opening the Digital Elevation Model

[24]:
```
dem = rasterio.open(file_path + 'raster.tif')
dem.read(1)
```

[24]:
```
array([[482.82904, 485.51953, 488.159  , ..., 618.8612 , 620.4424 ,
        622.05786],
       [481.6521 , 484.32193, 486.93958, ..., 618.8579 , 620.44556,
        622.06714],
       [480.52563, 483.18893, 485.80444, ..., 618.8688 , 620.4622 ,
        622.08923],
       ...,
       [325.49225, 327.21985, 328.94498, ..., 353.6889 , 360.03125,
        366.3984 ],
       [325.0538 , 326.78473, 328.51276, ..., 351.80603, 357.84106,
        363.96167],
       [324.61444, 326.34845, 328.0794 , ..., 350.09247, 355.87598,
        361.78635]], dtype=float32)
```

Creating the mesh containing the temperature data

[25]:
```
mesh = gg.visualization.create_temperature_map(dem=dem,
                                               mesh=mesh,
                                               name= 'Thickness [m]',
                                               apply_threshold=True,
                                               tsurface=10,
                                               gradient=0.03)
mesh
```

[25]:
```
UnstructuredGrid (0x20201e84fa0)
  N Cells:  3946
  N Points: 2130
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 2
```

[26]:
```
mesh['Thickness [m]']
```

[26]:
```
array([34.28413844, 41.96035767, 50.86764526, ...,  0.58549309,
        0.42822266,  0.42822266])
```

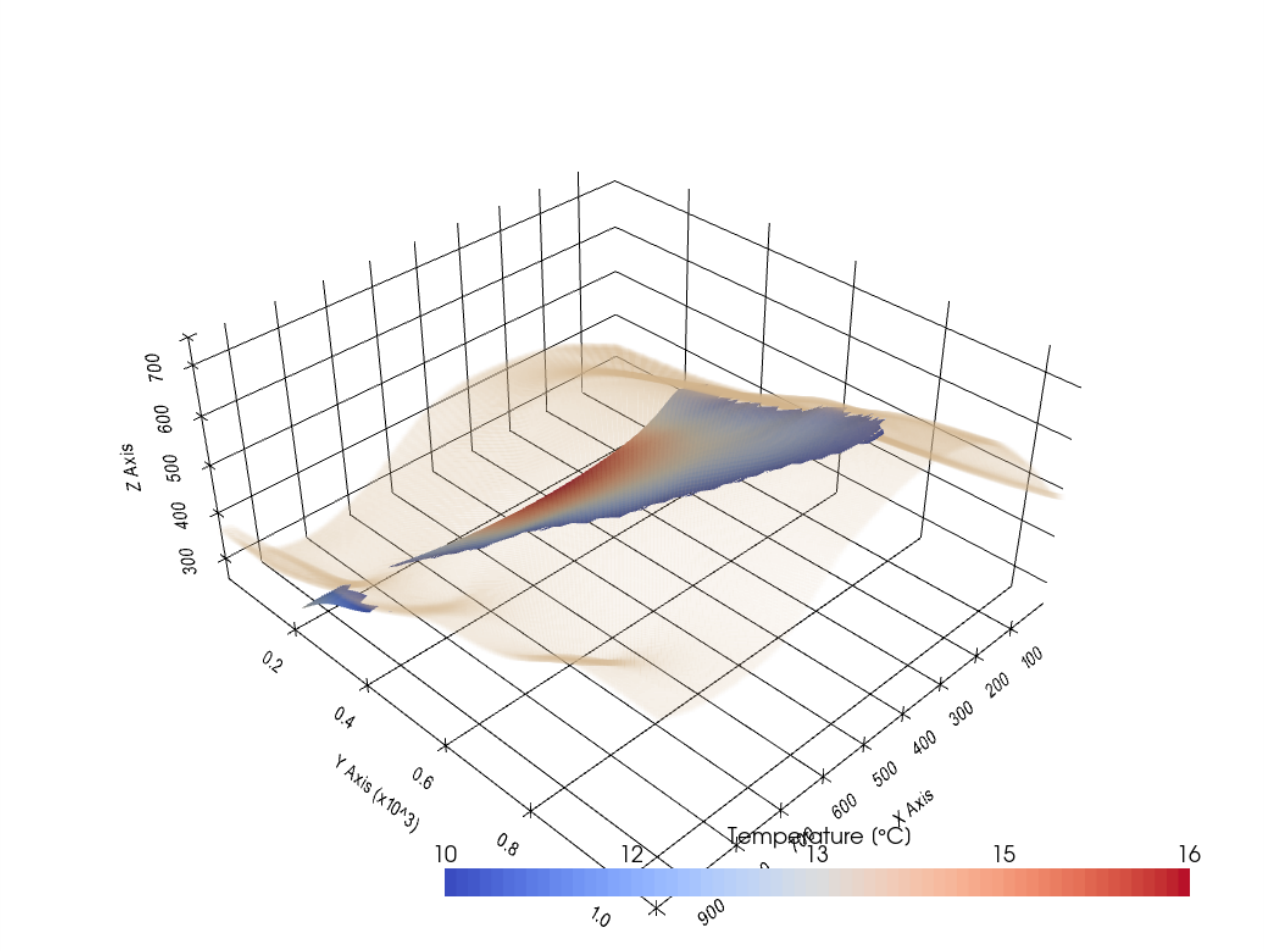
[27]:
```
mesh['Temperature [°C]']
```

[27]:
```
array([11.02852415, 11.25881073, 11.52602936, ..., 10.01756479,
       10.01284668, 10.01284668])
```

[28]:
```
import pyvista as pv
surf = pv.PolyData(geo_model._grid.topography.values)
```

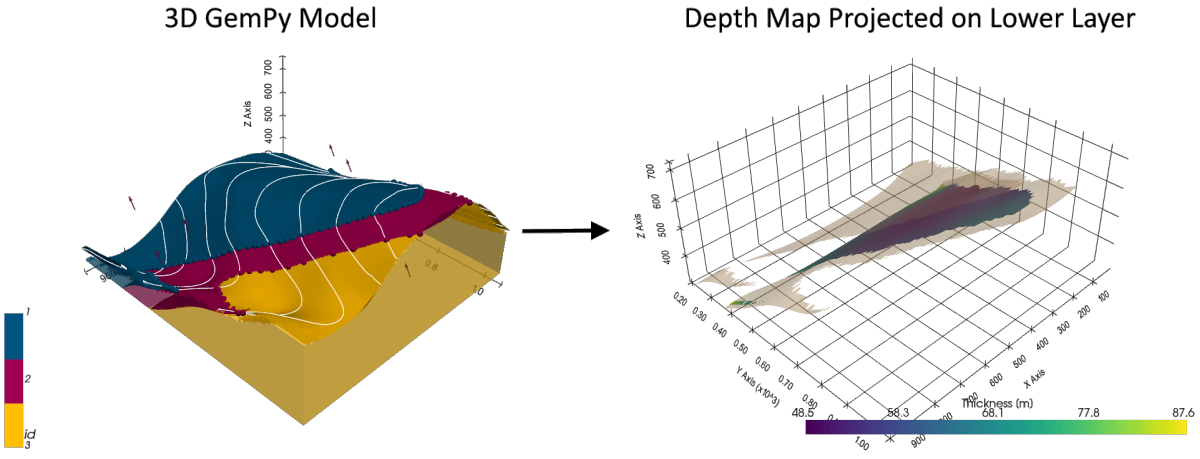
Plotting the temperature map and the topography. It can be seen that the temperatures at the edges are equal to 10 degrees C. These are the areas where the layer is outcropping at the surface. The temperatures are higher at the center of the surface where the layer has a higher thickness.

[32]:
```
sargs = dict(fmt="%.0f", color='black')


p = pv.Plotter(notebook=True)
p.add_mesh(mesh=mesh, scalars='Temperature [°C]', cmap='coolwarm', scalar_bar_args=sargs)
p.add_mesh(surf, opacity=0.025)

p.set_background('white')
p.show_grid(color='black')
p.show()
```



6.24 23 Calculating Thickness Maps with PyVista

Thickness maps are important tools for Geologists to investigate the spatial changes of thicknesses of a layer. These investigations could reveal depocenters of deposition or provide information about the thickness of a reservoir. A simple model (Example 1) is created of which a thickness map for the two existing layers is created.



6.24.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/23_calculating_thickness_maps/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="23_calculating_thickness_maps.
↪zip", dirpath=file_path)
```

6.24.2 Loading the data

[2]:
```
import geopandas as gpd
import rasterio

interfaces = gpd.read_file(file_path + 'interfaces.shp')
orientations = gpd.read_file(file_path + 'orientations.shp')
extent = [0,972,0,1069, 300, 800]
resolution = [50, 50, 50]
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[3]:
```
interfaces.head()
```

[3]:

	level_0	level_1	formation	X	Y	Z	geometry
0	0	0	Sand1	0.26	264.86	353.97	POINT (0.256 264.862)

(continues on next page)

(continued from previous page)

```
1        0        0     Sand1 10.59 276.73 359.04  POINT (10.593 276.734)
2        0        0     Sand1 17.13 289.09 364.28  POINT (17.135 289.090)
3        0        0     Sand1 19.15 293.31 364.99  POINT (19.150 293.313)
4        0        0     Sand1 27.80 310.57 372.81  POINT (27.795 310.572)
```

[4]:
```
orientations['polarity'] = 1
orientations.head()
```

[4]:
```
  formation   dip  azimuth      X      Y      Z                 geometry  \
0       Ton 30.50   180.00  96.47 451.56 477.73   POINT (96.471 451.564)
1       Ton 30.50   180.00 172.76 661.88 481.73  POINT (172.761 661.877)
2       Ton 30.50   180.00 383.07 957.76 444.45  POINT (383.074 957.758)
3       Ton 30.50   180.00 592.36 722.70 480.57  POINT (592.356 722.702)
4       Ton 30.50   180.00 766.59 348.47 498.96  POINT (766.586 348.469)

   polarity
0         1
1         1
2         1
3         1
4         1
```

6.24.3 Creating the GemPy Model

[5]:
```
import sys
sys.path.append('../../../../gempy-master')
import gempy as gp
```

[6]:
```
geo_model = gp.create_model('Model1')
geo_model
```

[6]:
```
Model1  2020-12-13 13:14
```

Initiating the Model

[7]:
```
import pandas as pd

gp.init_data(geo_model, extent, resolution,
             surface_points_df = interfaces,
             orientations_df = orientations,
             default_values=True)
geo_model.surfaces
```

```
Active grids: ['regular']
```

[7]:
```
  surface          series  order_surfaces    color  id
0   Sand1  Default series               1  #015482   1
1     Ton  Default series               2  #9f0052   2
```

The vertices and edges are currently `NaN` values, so no model has been computed so far.

[8]:
```
geo_model.surfaces.df
```

[8]:
```
  surface          series  order_surfaces  isBasement isFault  isActive  \
0   Sand1  Default series               1       False   False      True
1     Ton  Default series               2        True   False      True

   hasData    color vertices edges sfai  id
0     True  #015482      NaN   NaN  NaN   1
1     True  #9f0052      NaN   NaN  NaN   2
```

Mapping Stack to Surfaces

[9]:
```
gp.map_stack_to_surfaces(geo_model,
                         {"Strat_Series": ('Sand1', 'Ton')},
                         remove_unused_series=True)
geo_model.add_surfaces('basement')
```

[9]:
```
    surface        series  order_surfaces    color  id
0     Sand1  Strat_Series               1  #015482   1
1       Ton  Strat_Series               2  #9f0052   2
2  basement  Strat_Series               3  #ffbe00   3
```

Loading Topography

[10]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster1.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[10]:
```
Grid Object. Values:
array([[   9.72      ,   10.69      ,  305.        ],
       [   9.72      ,   10.69      ,  315.        ],
       [   9.72      ,   10.69      ,  325.        ],
       ...,
       [ 970.056     , 1059.28181818,  622.0892334 ],
       [ 970.056     , 1063.16909091,  622.06713867],
       [ 970.056     , 1067.05636364,  622.05786133]])
```

Setting Interpolator

[11]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                   values
range             1528.90
$C_o$            55655.83
drift equations      [3]
```

[11]:
```
<gempy.core.interpolator.InterpolatorModel at 0x17c25a90af0>
```

Computing Model

[12]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

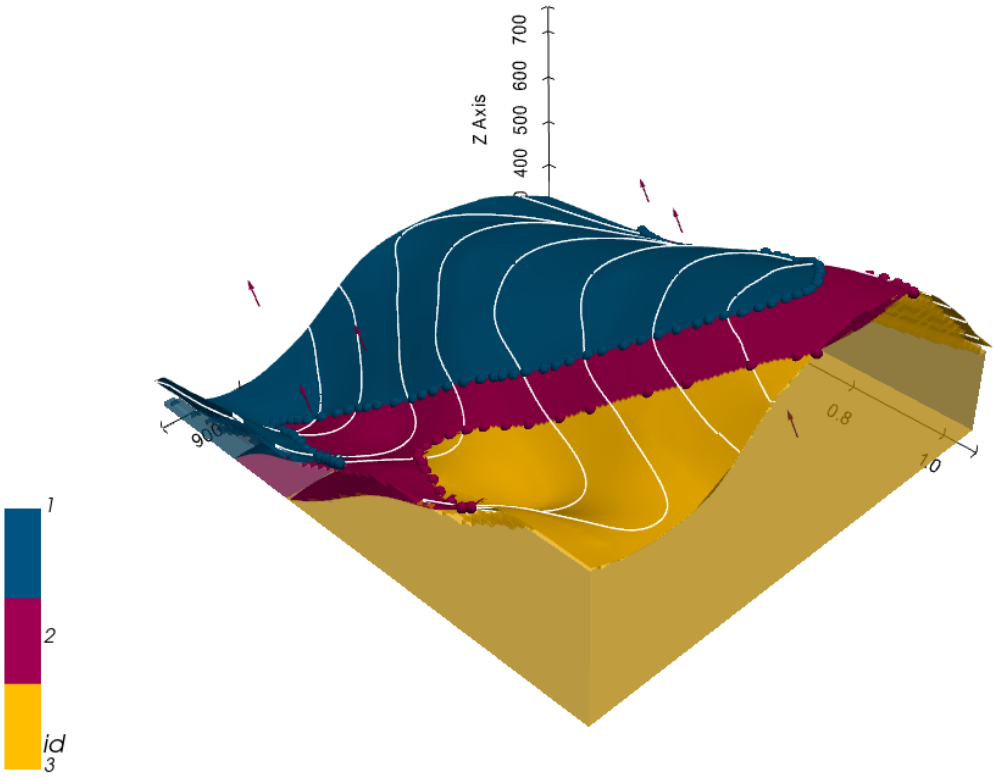
The surfaces DataFrame now contains values for vertices and edges.

[13]:
```
geo_model.surfaces.df
```

[13]:
```
    surface        series  order_surfaces  isBasement isFault  isActive  \
0     Sand1  Strat_Series               1       False   False      True
1       Ton  Strat_Series               2       False   False      True
2  basement  Strat_Series               3        True   False      True

   hasData    color                                              vertices  \
0     True  #015482  [[29.160000000000004, 194.27877317428587, 305...
1     True  #9f0052  [[29.160000000000004, 365.78652999877926, 305...
2     True  #ffbe00                                                NaN

                                              edges sfai  id
0  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.26   1
1  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.21   2
2                                                NaN  NaN   3
```

Plotting the 3D Model

[14]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



6.24.4 Creating Depth Maps

When creating the depth maps, a dict containing the mesh, the depth values and the color of the surface within the GemPy Model are returned.

[15]:
```
dict_sand1 = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model,
                                                           surfaces='Sand1')

dict_sand1
```

[15]:
```
{'Sand1': [PolyData (0x17c28731ca0)
    N Cells:    4174
    N Points:   2303
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   1.881e+02, 9.491e+02
    Z Bounds:   3.050e+02, 7.250e+02
    N Arrays:   0,
```

(continues on next page)

(continued from previous page)

```
  array([305.        , 305.        , 314.39133763, ..., 495.        ,
         495.        , 494.12916183]),
  '#015482']}
```

Plotting Depth Maps

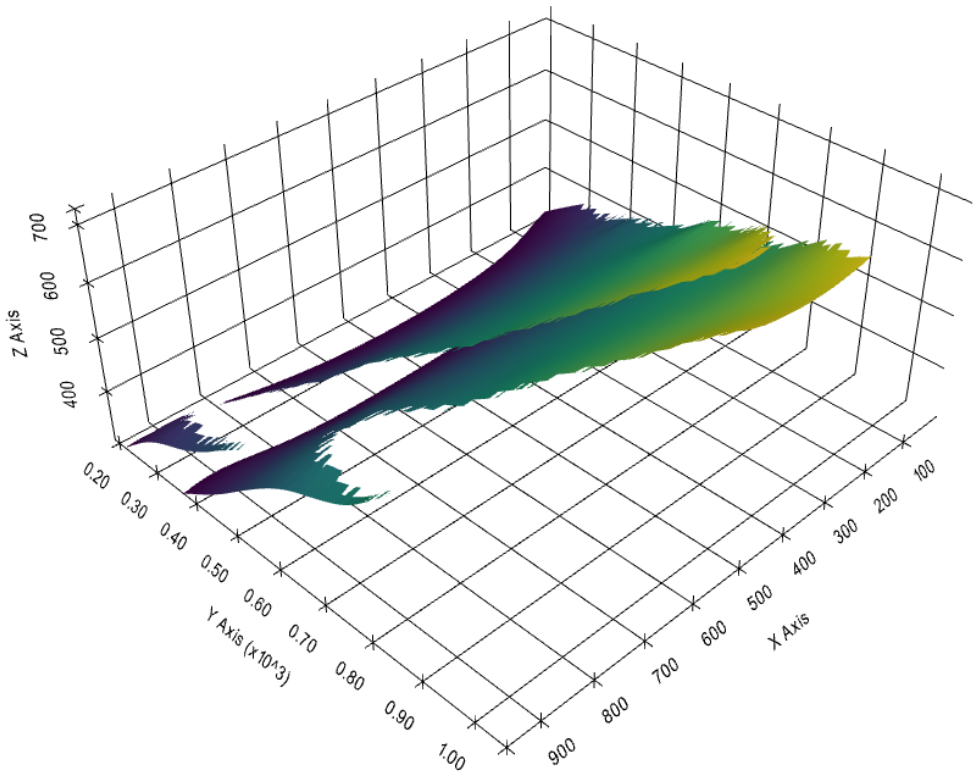
The depth maps can easily be plotted with PyVista.

[18]:
```
dict_all = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model,
                                                         surfaces=['Sand1', 'Ton'])

dict_all
```

[18]:
```
{'Sand1': [PolyData (0x17c28887100)
    N Cells:    4174
    N Points:   2303
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   1.881e+02, 9.491e+02
    Z Bounds:   3.050e+02, 7.250e+02
    N Arrays:   0,
  array([305.        , 305.        , 314.39133763, ..., 495.        ,
         495.        , 494.12916183]),
  '#015482'],
 'Ton': [PolyData (0x17c27ff8160)
    N Cells:    5111
    N Points:   2739
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   3.578e+02, 1.058e+03
    Z Bounds:   3.050e+02, 7.265e+02
    N Arrays:   0,
  array([305.        , 305.        , 314.91398156, ..., 595.        ,
         593.61738205, 595.        ]),
  '#9f0052']}
```

This data can be accessed as before to display both surfaces.

[19]:
```
import pyvista as pv

p = pv.Plotter(notebook=True)

p.add_mesh(dict_all['Sand1'][0], scalars='Depth [m]')
p.add_mesh(dict_all['Ton'][0], scalars='Depth [m]')

p.set_background('white')
p.show_grid(color='black')
p.show()
```



6.24.5 Creating Thickness Map

The thickness map can be calculated using `create_thickness_maps(..)`. The thickness is calculated using normal vectors of the bottom surface and intersecting them with the upper surface. In places where no upper surface is available, the value is set to 0.

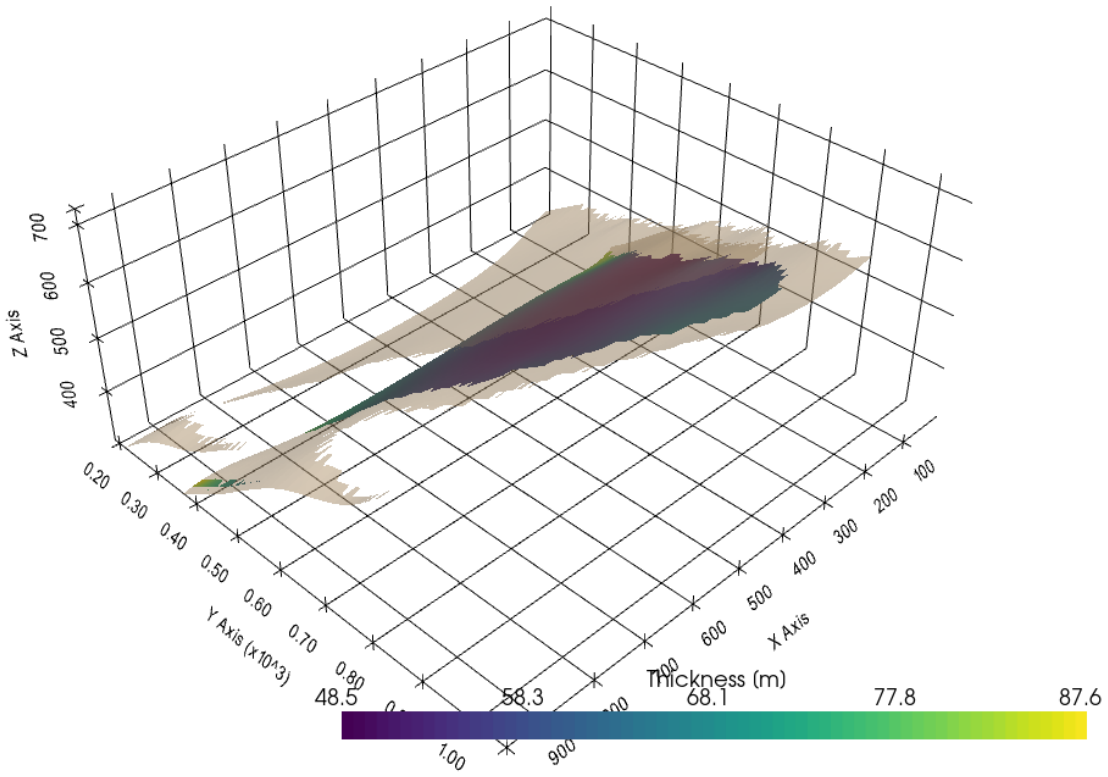
[23]:
```
thickness_map = gg.visualization.create_thickness_maps(top_surface=dict_all['Sand1'][0],
                                                       base_surface=dict_all['Ton'][0])
thickness_map
```

[23]:
```
PolyData (0x17c341a14c0)
  N Cells:  5111
  N Points: 2739
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 3.578e+02, 1.058e+03
  Z Bounds: 3.050e+02, 7.265e+02
  N Arrays: 3
```

Plotting the Thickness map

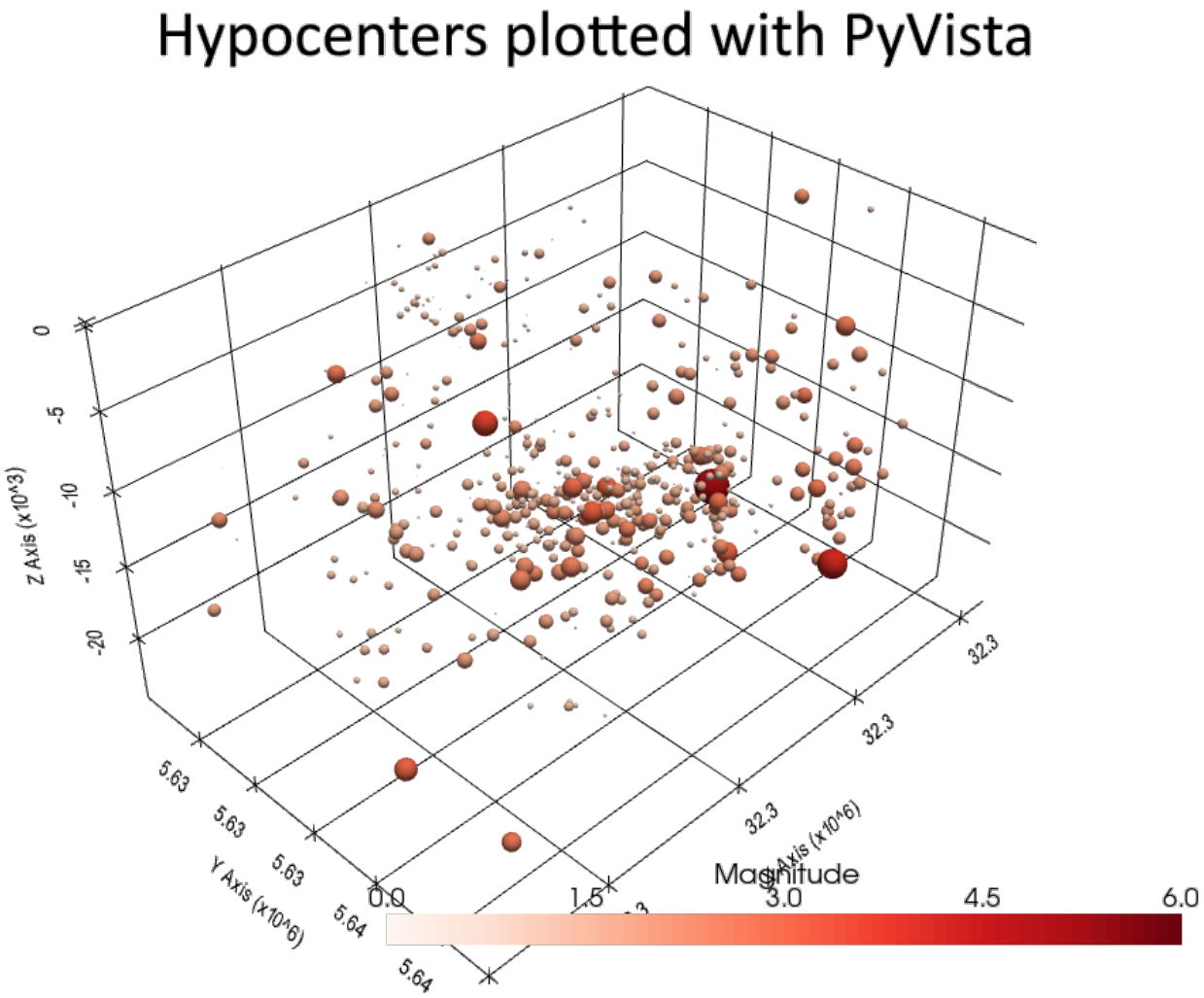
The thickness map can now be plotted using PyVista. The `nan_opacity` value is set to 0 so that `nan` values are not shown. The array `Thickness [m]` contains the thickness values and is plotted as `scalars` of the mesh. The thickness map is plotted at the location of the `base_surface`.

[38]:
```
import pyvista as pv

p = pv.Plotter(notebook=True)

p.add_mesh(thickness_map, scalars= 'Thickness [m]', nan_opacity=0)
p.add_mesh(dict_all['Sand1'][0], color=True, opacity=0.5)
p.add_mesh(dict_all['Ton'][0], color=True, opacity=0.5)


p.set_background('white')
p.show_grid(color='black')
p.show()
```



6.25 24 Plotting Hypocenters of Earthquakes with PyVista

Hypocenters of earthquake can be nicely visualized using GemGIS and PyVista.



6.25.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/24_plotting_hypocenters_of_earthquakes/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
```

(continues on next page)

(continued from previous page)

```
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="24_plotting_hypocenters_of_
↪earthquakes.zip", dirpath=file_path)
```

6.25.2 Load Earthquake Data

The data used here was obtained from the earthquake catalog of the Earthquake Station of the University of Cologne in Bensberg (http://www.seismo.uni-koeln.de/catalog/index.htm)

import gemgis as gg import pandas as pd

data = pd.read$_csv(file_path + 'earthquakes_achen.csv', delimiter = ';').dropna()data.head(15)$

data = data[data['Magnitude'] != '-'] data = data[data['Tiefe [km]'] != '-'] data.head(10)

data['Magnitude'] = pd.to$_numeric(data['Magnitude'])data['Tiefe[km]'] = pd.to_numeric(data['Tiefe[km]'])data['Date'] = pd.to_datetime(data['Date'])data['Year'] = pd.DatetimeIndex(data['Date']).year$

data['X'] = pd.to$_numeric(data['X'])data['Y'] = pd.to_numeric(data['Y'])data['Z'] = pd.to_numeric(data['Z'])$

6.25.3 Converting Pandas DataFrame to GeoDataFrame and reproject coordinates

[3]:
```
import geopandas as gpd
gdf = gpd.read_file(filename= file_path+'earthquake_data.shp')
```

[4]:
```
#gdf = gpd.GeoDataFrame(data, geometry=gpd.points_from_xy(data.X, data.Y), crs='EPSG:4326
↪').to_crs('EPSG:4647').reset_index()

gdf = gg.vector.extract_xy(gdf=gdf, reset_index=True)
gdf.head()
```

[4]:

	Y	X	Z	RASTERVALU	Tiefe [km]	Magnitude	\
0	5645741.63	32322660.15	-8249.25	150.75	8.40	1.50	
1	5645947.18	32323159.51	89.63	89.63	0.00	0.80	
2	5637979.08	32306261.57	-14056.14	143.86	14.20	2.20	
3	5645345.19	32321239.77	-8124.22	275.78	8.40	1.70	
4	5645698.15	32320689.25	-5329.58	270.42	5.60	1.90	

	Epizentrum	Year	geometry
0	STETTERNICH	2002	POINT (32322660.151 5645741.630)
1	SOPHIENHOEHE	2014	POINT (32323159.505 5645947.183)
2	ALSDORF	2007	POINT (32306261.571 5637979.080)
3	STETTERNICH	1999	POINT (32321239.770 5645345.189)
4	STETTERNICH	1999	POINT (32320689.249 5645698.148)

gdf.drop('Date', axis=1).to$_file(file_path + 'earthquake_data.shp')$

6.25.4 Loading WMS Data for Plotting

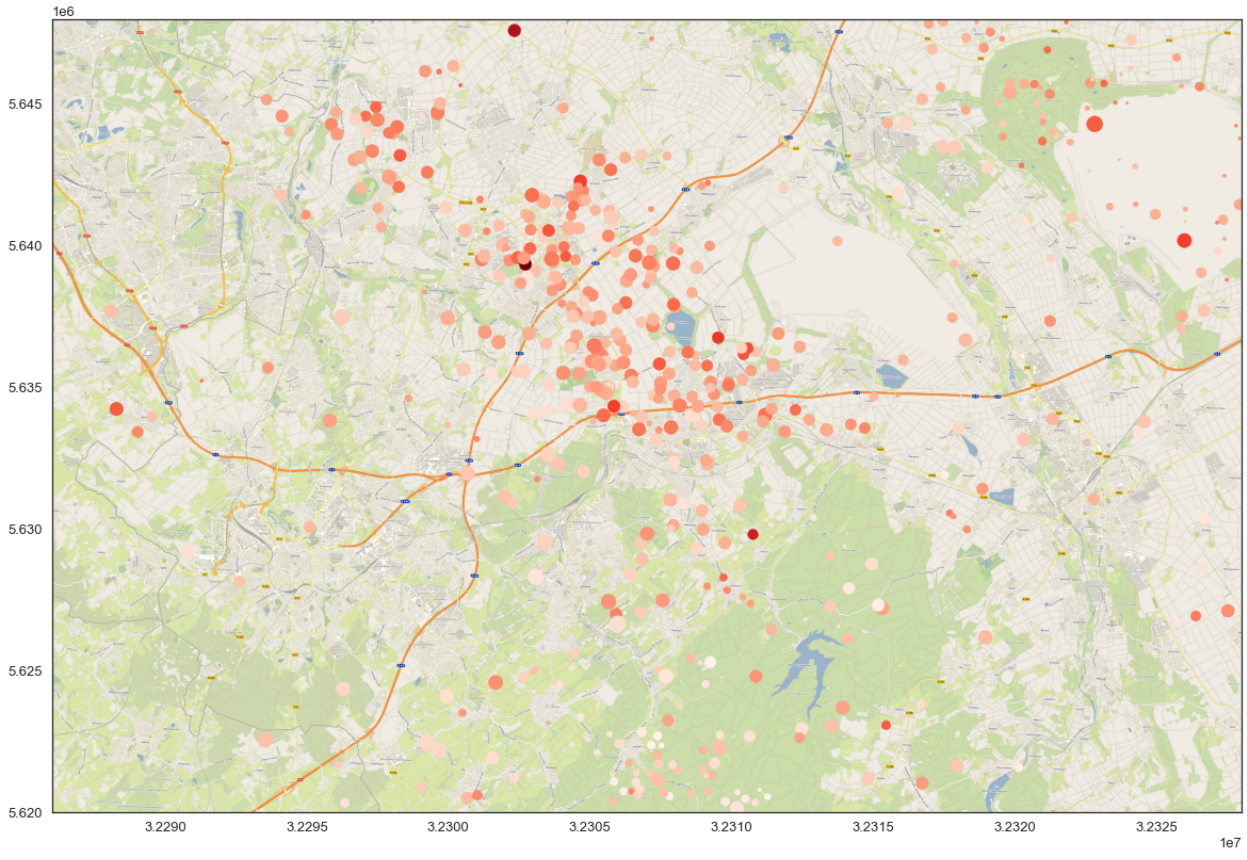
[5]:
```
wms = gg.web.load_wms(url='https://ows.terrestris.de/osm/service?')
```

[6]:
```
wms_map = gg.web.load_as_array(url=wms.url,
                               layer='OSM-WMS',
                               style='default',
                               crs='EPSG:4647',
                               bbox=[32286000,32328000, 5620000,5648000],
                               size=[4200, 2800],
                               filetype='image/png')
```

6.25.5 Plotting the data

[7]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1, figsize=(20,20))

ax.imshow(wms_map, extent=[32286000,32328000, 5620000,5648000])
gdf.plot(ax=ax, aspect='equal', column='Magnitude', cmap='Reds', markersize = gdf['Tiefe␣
↪[km]'].values*10)
```

[7]:
```
<AxesSubplot:>
```



6.25.6 Create PyVista Spheres

[8]:
```
import pyvista as pv

test = pv.Sphere(radius=1000, center=gdf.loc[0][['X', 'Y', 'Z']].tolist())
test
```

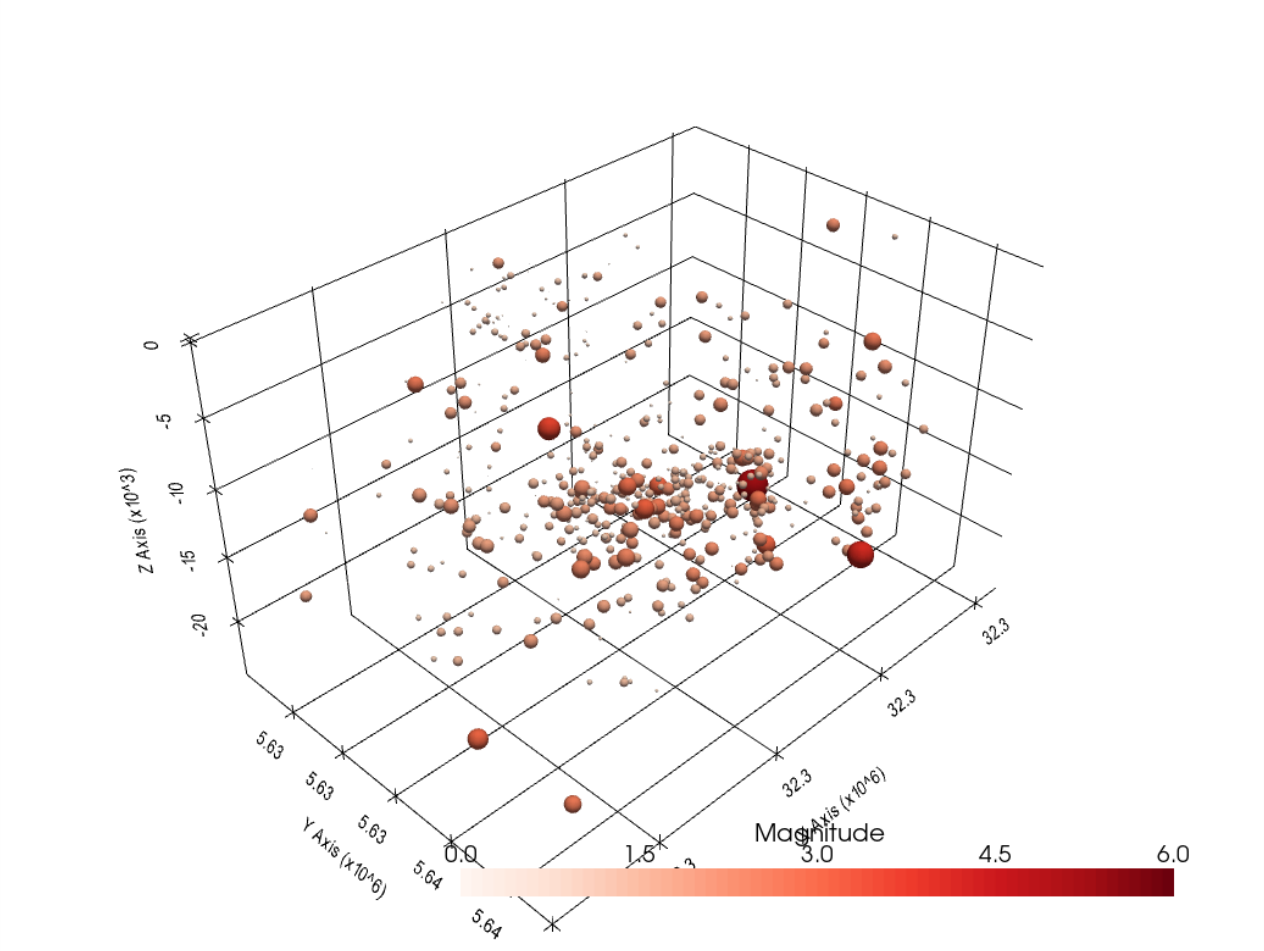
[8]:
```
PolyData (0x2cbf6ba3fa0)
  N Cells:  1680
  N Points: 842
  X Bounds: 3.232e+07, 3.232e+07
  Y Bounds: 5.645e+06, 5.647e+06
  Z Bounds: -9.249e+03, -7.249e+03
  N Arrays: 1
```

[9]:
```
import numpy as np

spheres = pv.MultiBlock([pv.Sphere(radius=gdf.loc[i]['Magnitude']*200, center=gdf.
↪loc[i][['X', 'Y', 'Z']].tolist()) for i in range(len(gdf))])

for i in range(len(spheres.keys())):
    spheres[spheres.keys()[i]]['Magnitude'] = np.zeros(len(spheres[spheres.keys()[i]].
↪points)) + gdf.loc[i]['Magnitude']
    spheres[spheres.keys()[i]]['Year'] = np.zeros(len(spheres[spheres.keys()[i]].
↪points)) + gdf.loc[i]['Year']
```

[10]:
```
type(spheres)
```

[10]:
```
pyvista.core.composite.MultiBlock
```

[11]:
```
sargs = dict(fmt="%.1f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(spheres,scalars='Magnitude', cmap='Reds', clim=[0,6], scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.show()
```

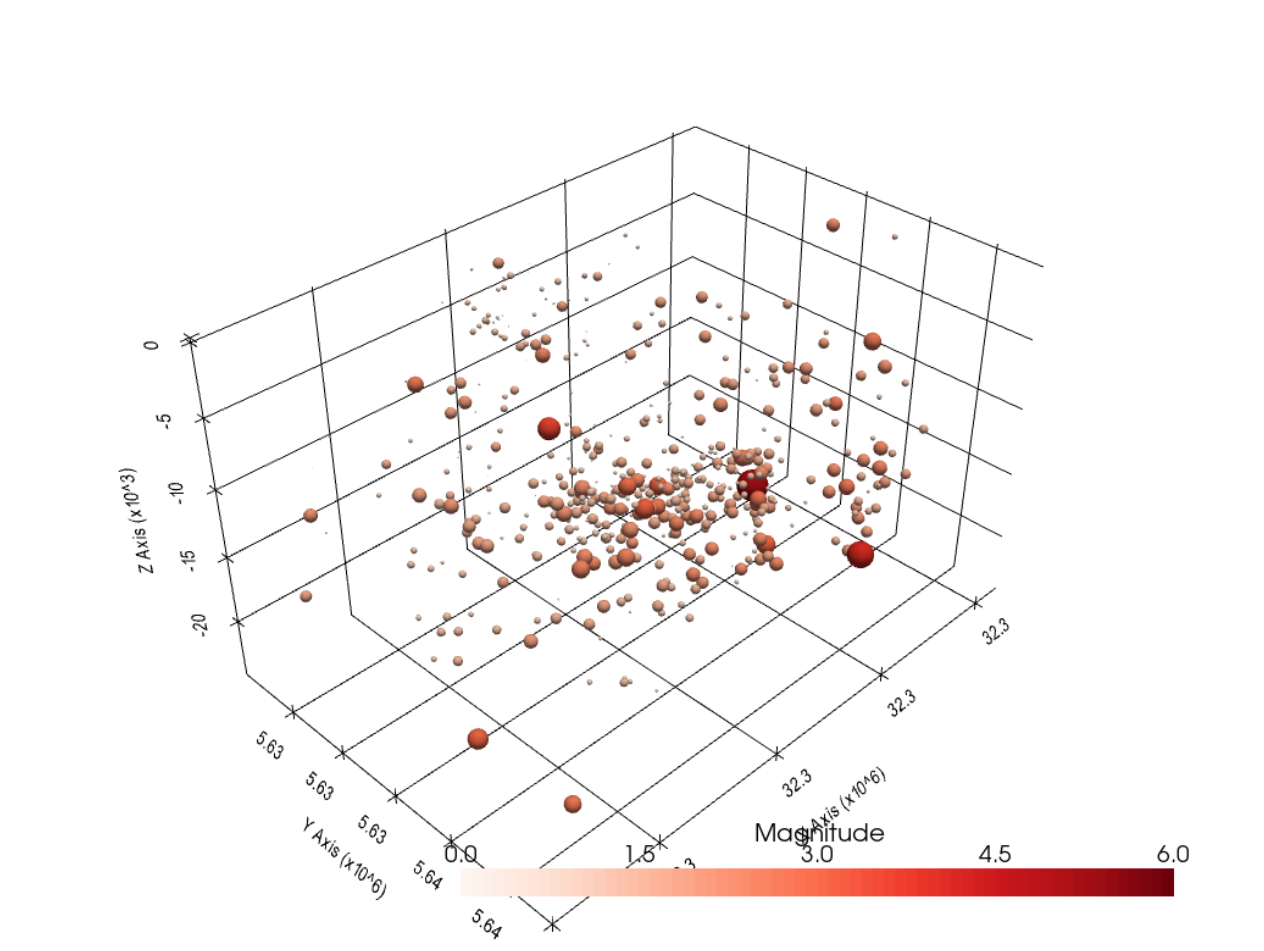


The above shown steps are also combined in the function `create_meshes_hypocenters(..)`.

[12]:
```
spheres = gg.visualization.create_meshes_hypocenters(gdf=gdf)
spheres
```

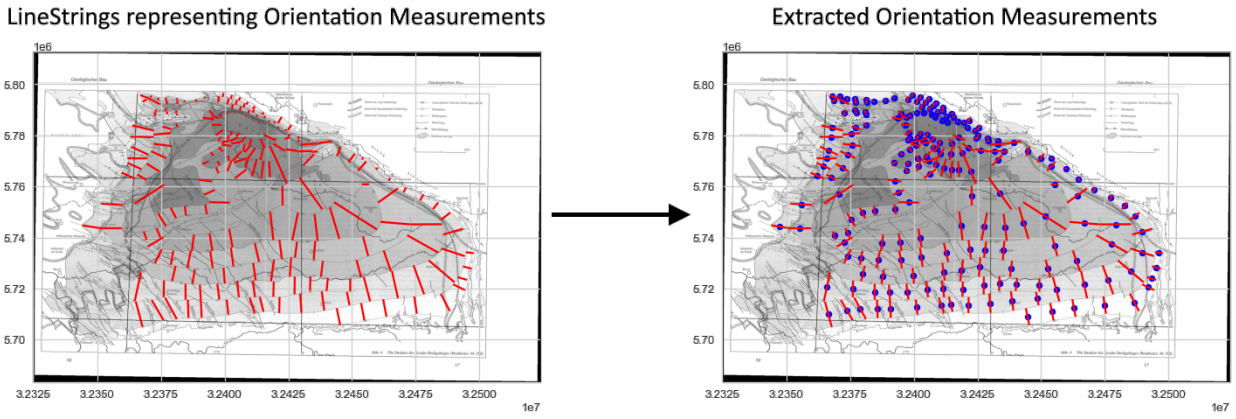
[12]:
```
MultiBlock (0x2cbf4b818e0)
  N Blocks: 497
  X Bounds: 32287780.000, 32328260.000
  Y Bounds: 5620074.000, 5648385.000
  Z Bounds: -24317.020, 309.130
```

[13]:
```
sargs = dict(fmt="%.1f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(spheres,scalars='Magnitude', cmap='Reds', clim=[0,6], scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.show()
```



6.26 25 Creating Orientations from Isolines on Maps

Orientations can be calculated from LineStrings representing the highest gradient between two or more isolines of a surface.



Source: Geologie im Münsterland (1995) by Günter Drozdzewski, Martin Hiss, Franziska Lehmann, Gert Michel, Klaus Skupin, Henner Staude, Arend Thiermann, Hildegard Dahm-Arens, Walter Finke.

6.26.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/25_creating_orientations_from_isolines_on_maps/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="25_creating_orientations_from_
↪isolines_on_maps.zip", dirpath=file_path)
```

6.26.2 Loading data

The data used here represents the base of the Münsterland Basin in northern Germany.

The map was extracted from 'Geologie im Münsterland' (1995) by Günter Drozdzewski, Martin Hiss, Franziska Lehmann, Gert Michel, Klaus Skupin, Henner Staude, Arend Thiermann, Hildegard Dahm-Arens, Walter Finke. The orientation values were digitized in QGIS.

[2]:
```
import rasterio
import geopandas as gpd

raster = rasterio.open(file_path + 'Tiefenlage_BasisKreide_georeferenziert.tif')
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

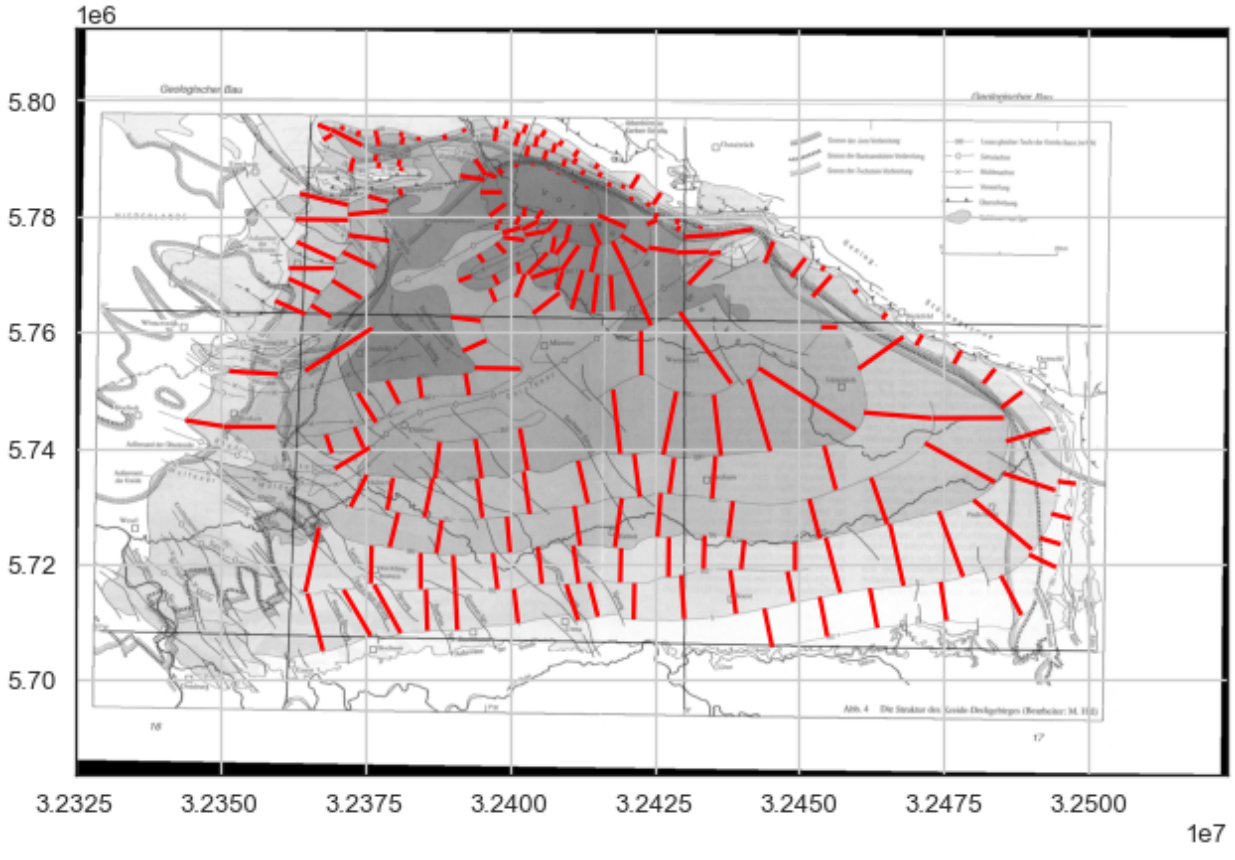
[3]:
```
orientations = gpd.read_file(file_path + 'bottom_cret_orient.shp')
orientations.head()
```

[3]:
```
     id   dZ                                                geometry
0  None  500  LINESTRING (32478299.585 5717761.236, 32474124...
1  None  500  LINESTRING (32468027.985 5716203.394, 32465662...
2  None  500  LINESTRING (32457110.971 5715004.084, 32454603...
3  None  250  LINESTRING (32449009.343 5713465.851, 32447984...
4  None  250  LINESTRING (32439984.702 5719017.270, 32440531...
```

6.26.3 Plotting the Data

The orientations consist of LineStrings consisting of two points and were digitized along the steepest gradient between two isolines/contour lines.

[4]:
```
import matplotlib.pyplot as plt
from rasterio.plot import reshape_as_image
fig, ax = plt.subplots(1,1)

orientations.plot(ax=ax, aspect='equal', color='red')
ax.imshow(reshape_as_image(raster.read()), extent=[raster.bounds[0], raster.bounds[2],␣
↪raster.bounds[1], raster.bounds[3]], cmap='gray')
plt.grid()
```



6.26.4 Extracting Orientations

The orientations can be extracted from the map using `extract_orientations_from_map(..)`.

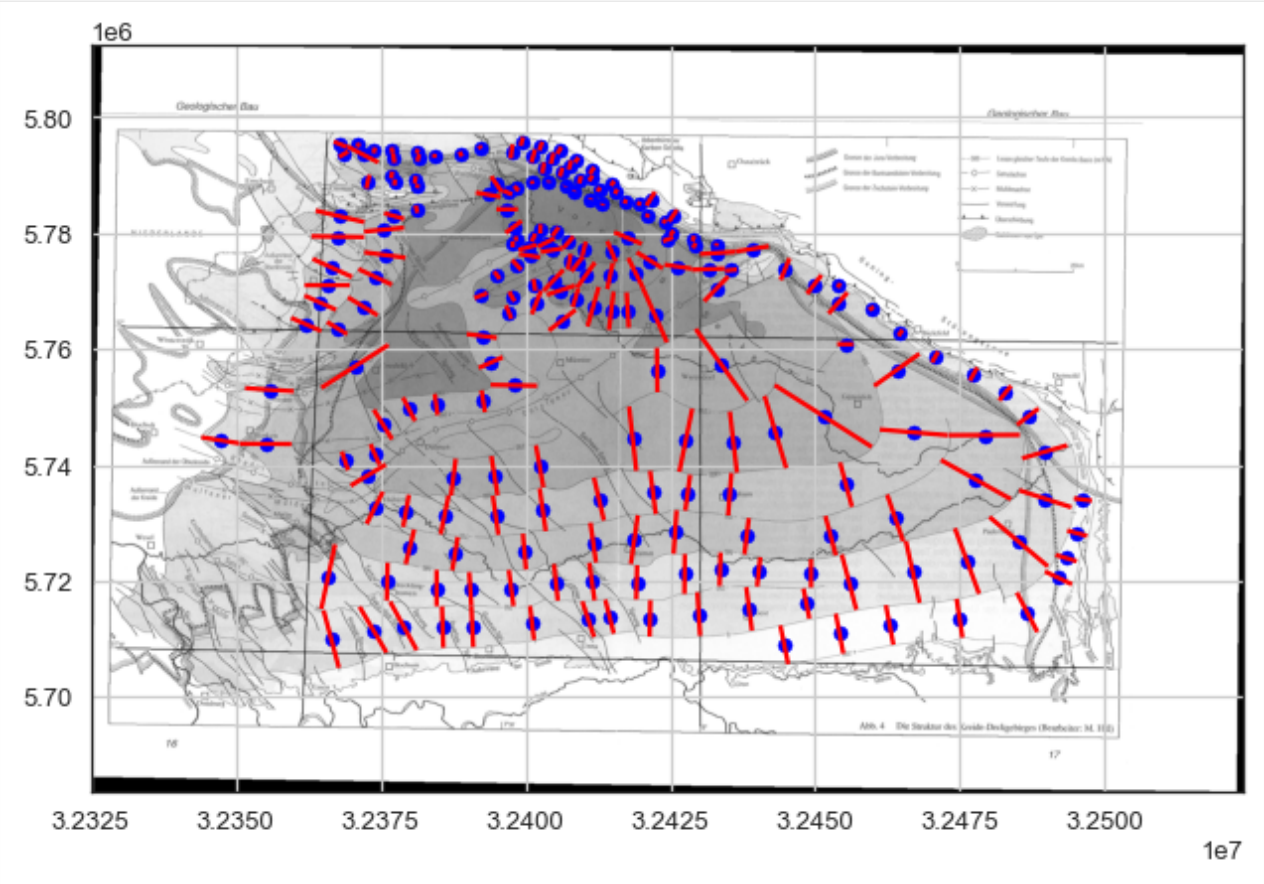
[5]:
```
gdf = gg.vector.extract_orientations_from_map(gdf=orientations)
gdf
```

[5]:
```
                            geometry  azimuth  dip           X          Y  \
0    POINT (32476212.059 5723406.886)   339.71 2.38 32476212.06 5723406.89
1    POINT (32466845.216 5721529.592)   347.48 2.62 32466845.22 5721529.59
2    POINT (32455857.358 5719760.010)   345.23 2.91 32455857.36 5719760.01
3    POINT (32448496.843 5716069.354)   348.86 2.70 32448496.84 5716069.35
4    POINT (32440258.151 5721545.726)     6.17 2.81 32440258.15 5721545.73
..                                ...      ...  ...         ...        ...
201  POINT (32397115.154 5766347.158)   337.75 5.77 32397115.15 5766347.16
202  POINT (32401900.665 5768063.700)    21.43 3.24 32401900.67 5768063.70
203  POINT (32406348.070 5764968.723)    52.21 2.44 32406348.07 5764968.72
204  POINT (32397999.433 5753733.174)   271.12 1.79 32397999.43 5753733.17
205  POINT (32377335.907 5782940.399)   105.95 7.91 32377335.91 5782940.40

     polarity
0           1
1           1
2           1
3           1
4           1
..        ...
201         1
202         1
203         1
204         1
205         1

[206 rows x 6 columns]
```

6.26.5 Plotting the results

It can be seen that the locations of the orientations are the midpoints of the LineStrings.

[6]:
```
import matplotlib.pyplot as plt
from rasterio.plot import reshape_as_image
fig, ax = plt.subplots(1,1)

orientations.plot(ax=ax, aspect='equal', color='red')
gdf.plot(ax=ax, aspect='equal',color='blue', markersize=50)
ax.imshow(reshape_as_image(raster.read()), extent=[raster.bounds[0], raster.bounds[2],␣
↪raster.bounds[1], raster.bounds[3]], cmap='gray')
plt.grid()
```



6.27 26 Working with Well Data from the Geological Survey NRW

This notebook presents the extraction of borehole data (location of wells and stratigraphy) from logs provided by the Geological Survey NRW.

Original Well Data

Stammdaten - 2521/ 5631/ 1 - Bnum: 196747

Objekt / Name :	B. 19 ESCHWEILER
Bohrungs- / Aufschluß-Nr. :	19
Archiv-Nr. :	
Endteufe [m] :	70.30
Stratigraphie der Endteufe :	Karbon
TK 25 :	Eschweiler [TK 5103]
Ort / Gemarkung :	Eschweiler/Weißweiler
GK Rechtswert/Hochwert [m] :	2521370.00 / 5631910.00
UTM East/North [m] :	32310019.32 / 5633520.32
Hoehe des Ansatzpunktes [mNN] :	130.00
Koordinatenbestimmung :	ungeprüfte Angabe aus dem Bohrarchiv
Hoehenbestimmung :	ungeprüfte Angabe aus dem Bohrarchiv
Hauptzweck des Aufschlusses :	Exploration, Lagerstättenerkundung
Aufschlussart :	Bohrung
Aufschlussverfahren :	
Vertraulichkeit :	vertraulich, offen nach Einzelfallprüfung;
Art der Aufnahme :	Übertragung eines alten Archivbestandes
Schichtenverzeichnis Version :	1
Qualität :	Schichtdaten von guter Qualität; genaue stratigrafische Einstufung
aufgestellt von :	NN am
geol./stratgr. bearbeitet von :	NN am
erster - letzter Bohrtag :	
Grundwasserstand [m] :	
Oberster Grundwasserstand [m] :	
Auftraggeber :	NN
Fachaufsicht :	
Bohrunternehmer :	
Bohrmeister :	
Gerät :	
Bemerkung :	
Originalschichtenverzeichnis :	Original-Schichtenverzeichnis liegt vor

Extracted Meta Data

	Index	DABO No.	Name	Number	Depth	X	Y	Z
0	GD0001	DABO_196747	B.19ESCHWEILER	19	70.30	32310019.32	5633520.32	130.00
1	GD0002	DABO_196748	B.16ESCHWEILER	16	37.61	32310327.14	5632967.35	122.00

6.27.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/26_working_with_well_data_from_GD_NRW/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="26_working_with_well_data_from_
↪GD_NRW.zip", dirpath=file_path)
```

6.27.2 Loading the Well Data

The data used for GemGIS is obtained from the Geological Survey NRW. It will be used under Datenlizenz Deutschland – Namensnennung – Version 2.0 (https://www.govdata.de/dl-de/by-2-0).

The PDF Files can be loaded as strings using PyPDF2.

[5]:
```
data = gg.misc.load_pdf(path=file_path + 'test_data.pdf')
data[:500]
```

```
100%|| 2/2 [00:00<00:00, 37.16it/s]
```

```
../../../../gemgis_data/data/26_working_with_well_data_from_GD_NRW/test_data.txt␣
↪successfully saved
```

[5]:
```
'Stammdaten    -     2521/ 5631/ 1          -          Bnum: 196747  .  .  Objekt / Name :
↪B. 19  ESCHWEILER\n\n  Bohrungs- / Aufschluß-Nr. :19\n\n  Archiv-Nr. :\n  Endteufe [m]␣
↪:70.30\n\n  Stratigraphie der Endteufe :Karbon\n  .  TK 25 :Eschweiler [TK 5103]\n\n ␣
↪Ort / Gemarkung :Eschweiler/Weißweiler\n\n  GK  Rechtswert/Hochwert [m] :2521370.00  ␣
↪/ 5631910.00\n\n  UTM  East/North  [m] :32310019.32 / 5633520.32\n\n  Hoehe des␣
↪Ansatzpunktes [mNN] :130.00\n\n  Koordinatenbestimmung :ungeprüfte Angabe aus dem␣
↪Bohrarch'
```

6.27.3 Extracting Meta Data From the Well Data

The meta data or 'Stammdaten' of the wells can be extracted using `get_meta_data_df(...)`. Any duplicate wells will be removed automatically.

[3]:
```
df = gg.misc.get_meta_data_df(data=data,
                              name='GD')
df
```

[3]:
```
    Index     DABO No.            Name Number  Depth           X          Y  \
0  GD0001  DABO_196747  B.19ESCHWEILER     19  70.30 32310019.32 5633520.32
1  GD0002  DABO_196748  B.16ESCHWEILER     16  37.61 32310327.14 5632967.35

       Z       X_GK       Y_GK  ...     Kind Procedure  \
0 130.00 2521370.00 5631910.00  ...  Bohrung
1 122.00 2521700.00 5631370.00  ...  Bohrung

                            Confidentiality  \
0  vertraulich, offen nach Einzelfallprüfung;
1  vertraulich, offen nach Einzelfallprüfung;

                             Record Type Lithlog Version  \
0  Übertragung eines alten Archivbestandes               1
1  Übertragung eines alten Archivbestandes               1

                                        Quality Drilling Period Remarks  \
0  Schichtdaten von guter Qualität; genaue strati...
1  Schichtdaten von guter Qualität; genaue strati...

                     Availability Lithlog                          geometry
```

(continues on next page)

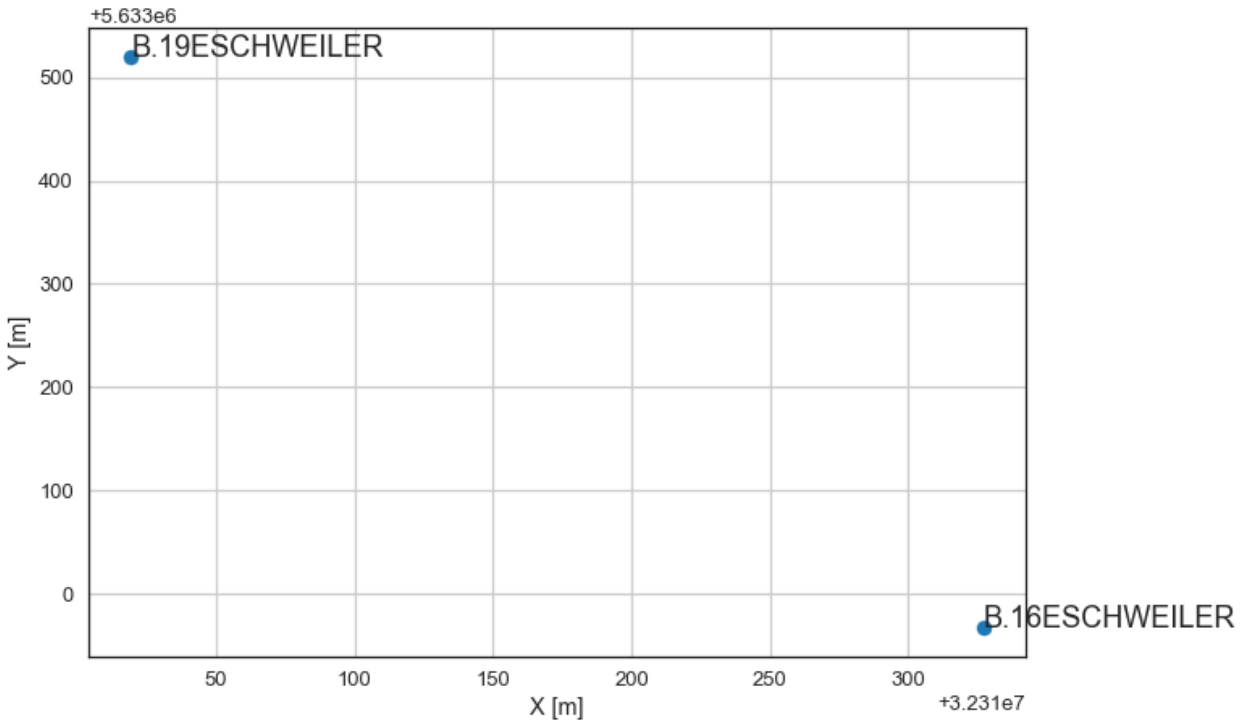
(continued from previous page)

```
0  Original-Schichtenverzeichnis liegt vor  POINT (32310019.320 5633520.320)
1  Original-Schichtenverzeichnis liegt vor  POINT (32310327.140 5632967.350)

[2 rows x 26 columns]
```

Plot Data

The locations of the wells can easily be plotted using Matplotlib or the built-in GeoPandas functions.

[4]:
```
import matplotlib.pyplot as plt

plt.scatter(df['X'], df['Y'])
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
for i in range(len(df)):
    plt.text(df['X'].loc[i], df['Y'].loc[i], df['Name'].loc[i])
```



6.27.4 Extracting Stratigraphic Data from Well Data

The stratigraphic data can be extracted using `get_stratigraphic_data_df(..)`. Different files have to be loaded beforehand to make the workflow work. This includes a file containing symbols that will be filtered out and the classification of the different formations.

[5]:
```
data = gg.misc.load_pdf(path=file_path + 'test_data.pdf',
                        save_as_txt=True)

data[:500]
```

```
100%|| 2/2 [00:00<00:00, 66.64it/s]
```

```
../../../../gemgis_data/data/26_working_with_well_data_from_GD_NRW/test_data.txt␣
↪successfully saved
```

[5]:
```
'Stammdaten    -     2521/ 5631/ 1         -          Bnum: 196747  .  .  Objekt / Name :
↪B. 19  ESCHWEILER\n\n  Bohrungs- / Aufschluß-Nr. :19\n\n  Archiv-Nr. :\n  Endteufe [m]␣
↪:70.30\n\n  Stratigraphie der Endteufe :Karbon\n  .  TK 25 :Eschweiler [TK 5103]\n\n ␣
↪Ort / Gemarkung :Eschweiler/Weißweiler\n\n  GK  Rechtswert/Hochwert [m] :2521370.00  ␣
↪/ 5631910.00\n\n  UTM  East/North  [m] :32310019.32 / 5633520.32\n\n  Hoehe des␣
↪Ansatzpunktes [mNN] :130.00\n\n  Koordinatenbestimmung :ungeprüfte Angabe aus dem␣
↪Bohrarch'
```

Load Well Data from txt-file

The data can be loaded from a text file so that the original PDF does not have to be reloaded again to save time.

[6]:
```
with open(file_path + 'test_data.txt', "r") as text_file:
    data = text_file.read()
```

Load Symbols from txt-file

Symbols that will be removed by default from the well data can be loaded from a text file.

[7]:
```
with open(file_path + 'symbols.txt', "r") as text_file:
    symbols = [(i, '') for i in text_file.read().splitlines()]

symbols
```

[7]:
```
[('.m ', ''),
 (', ', ''),
 ('; ', ''),
 (': ', ''),
 ('/ ', ''),
 ('? ', ''),
 ('! ', ''),
 ('-"- ', ''),
 ('" ', ''),
 ('% ', ''),
 ('< ', ''),
 ('> ', ''),
```

(continues on next page)

(continued from previous page)

```
 ('= ', ''),
 ('~ ', ''),
 ('_ ', ''),
 ('^A° ', ''),
 ("' ", '')]
```

Load Formations from txt-file

Classified formations can be loaded from a text file.

[8]:
```
with open(file_path + 'formations.txt', "rb") as text_file:
    formations = text_file.read().decode("UTF-8").split()

formations = [(formations[i], formations[i+1]) for i in range(0,len(formations)-1,2)]
formations[:10]
```

[8]:
```
[('UnterdevonKalltalFormation', 'KalltalFM'),
 ('nullLöss', 'Quaternary'),
 ('QuartärFlugsand', 'Quaternary'),
 ('QuartärHauptterrassen', 'Quaternary'),
 ('QuartärSandlöss', 'Quaternary'),
 ('QuartärHochflutablagerungen', 'Quaternary'),
 ('QuartärAnthropogeneBildungen(künstlicheAufschüttung)', 'Quaternary'),
 ('QuartärVerschwemmungsablagerungenFrostbodenbildungenundRutschmassen',
  'Quaternary'),
 ('QuartärLösslehm', 'Quaternary'),
 ('QuartärHochflutlehm', 'Quaternary')]
```

Extracting the Stratigraphic Data

After loading the symbols and formations, the stratigraphic data can be extracted. The (Geo-)DataFrame contains the index, the well name, X, Y and Z coordinates, the altitudes, the depths, the formations and a geometry column.

[9]:
```
df = gg.misc.get_stratigraphic_data_df(data=data,
                                       name='GD',
                                       symbols=symbols,
                                       formations=formations,
                                       return_gdf=True)
df
```

[9]:
```
    Index            Name           X          Y      Z  Altitude  Depth  \
0  GD0001  B.19ESCHWEILER 32310019.32 5633520.32 125.30    130.00  70.30
1  GD0001  B.19ESCHWEILER 32310019.32 5633520.32  66.50    130.00  70.30
2  GD0001  B.19ESCHWEILER 32310019.32 5633520.32  60.90    130.00  70.30
3  GD0001  B.19ESCHWEILER 32310019.32 5633520.32  59.70    130.00  70.30
4  GD0002  B.16ESCHWEILER 32310327.14 5632967.35 117.80    122.00  37.61
5  GD0002  B.16ESCHWEILER 32310327.14 5632967.35  84.40    122.00  37.61
6  GD0002  B.16ESCHWEILER 32310327.14 5632967.35  84.39    122.00  37.61

       formation                          geometry
0     Quaternary  POINT (32310019.320 5633520.320)
```

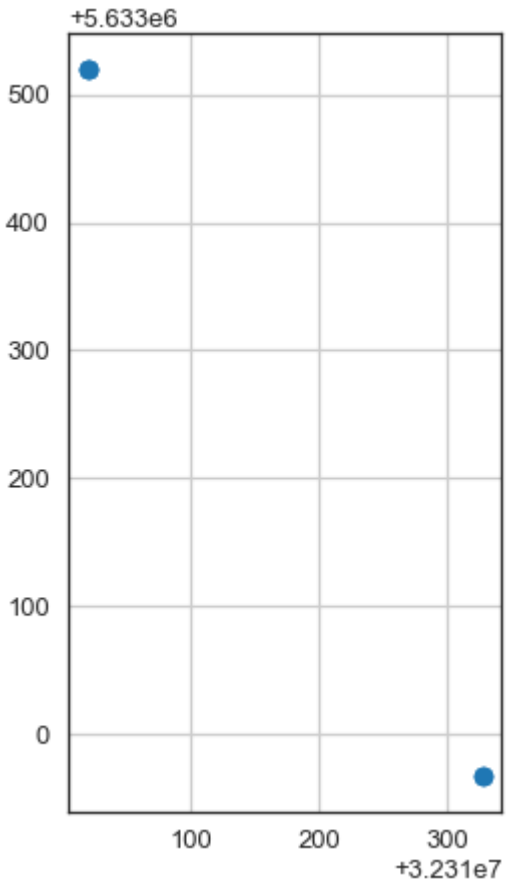
(continues on next page)

(continued from previous page)

```
1        Miocene  POINT (32310019.320 5633520.320)
2      Oligocene  POINT (32310019.320 5633520.320)
3  Carboniferous  POINT (32310019.320 5633520.320)
4     Quaternary  POINT (32310327.140 5632967.350)
5        Miocene  POINT (32310327.140 5632967.350)
6  Carboniferous  POINT (32310327.140 5632967.350)
```

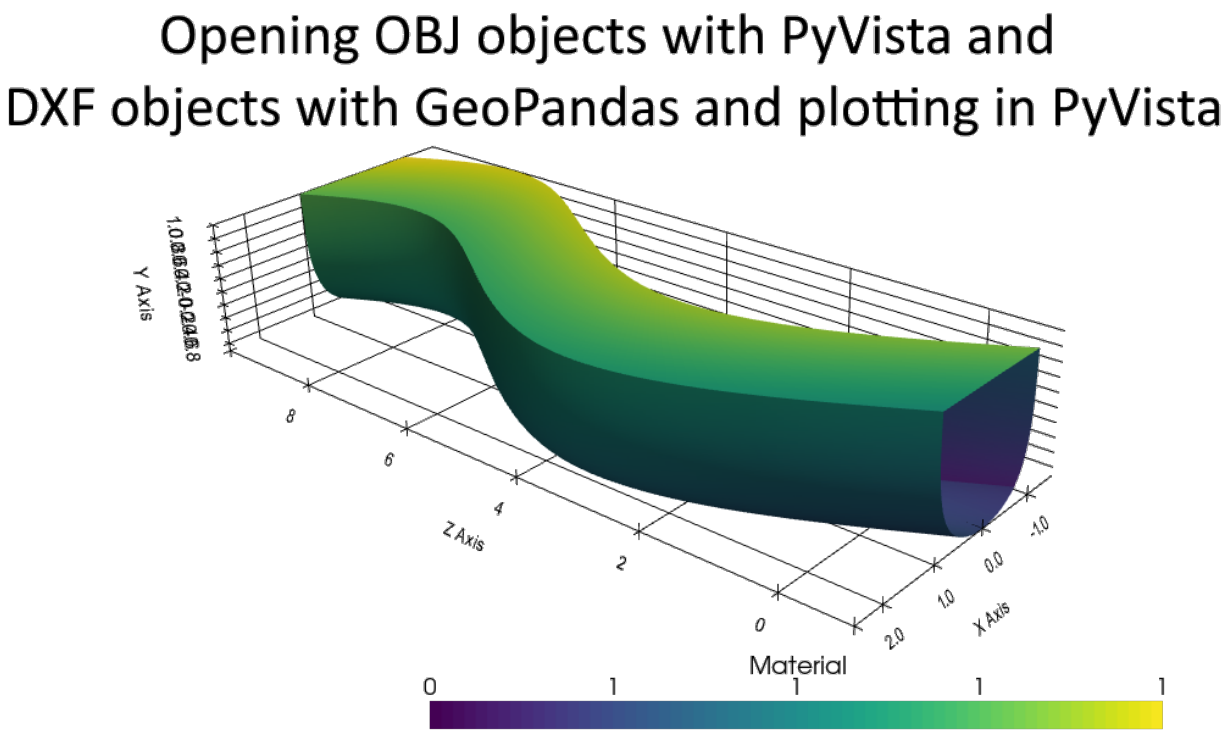
Plotting data

[10]:
```
df.plot()
plt.grid()
```



6.28 27 Opening OBJ and DXF Files with PyVista in GemGIS

OBJ and DXF Files can be opened with GemGIS and visualized with PyVista.



6.28.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/27_opening_obj_and_dxf_files/'
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\gemgis\gemgis.py:27:␣
↪UserWarning: Shapely 2.0 is installed, but because PyGEOS is also installed, GeoPandas␣
↪will still use PyGEOS by default for now. To force to use and test Shapely 2.0, you␣
↪have to set the environment variable USE_PYGEOS=0. You can do this before starting the␣
↪Python process, or in your code before importing geopandas:

import os
os.environ['USE_PYGEOS'] = '0'
import geopandas

In a future release, GeoPandas will switch to using Shapely by default. If you are using␣
↪PyGEOS directly (calling PyGEOS functions on geometries from GeoPandas), this will␣
↪then stop working and you are encouraged to migrate from PyGEOS to Shapely 2.0 (https:/
↪/shapely.readthedocs.io/en/latest/migration_pygeos.html).
```

(continues on next page)

(continued from previous page)

```
  import geopandas as gpd
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="27_opening_obj_and_dxf_files.zip
↪", dirpath=file_path)
```

```
Downloading file '27_opening_obj_and_dxf_files.zip' from 'https://rwth-aachen.sciebo.de/
↪s/AfXRsZywYDbUF34/download?path=%2F27_opening_obj_and_dxf_files.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\tutorial\data\27_opening_obj_and_dxf_
↪files'.
```

6.28.2 Loading OBJ File using PyVista

The obj file can easily be loaded and plotted using PyVista.

[3]:
```
import pyvista as pv

mesh = pv.read(file_path + 'Channel.obj')
mesh
```

[3]:
```
PolyData (0x23de4877d00)
  N Cells:  49152
  N Points: 196608
  N Strips: 0
  X Bounds: -1.576e+00, 2.530e+00
  Y Bounds: -9.167e-01, 1.000e+00
  Z Bounds: -1.000e+00, 9.751e+00
  N Arrays: 6
```

6.28.3 Plotting the mesh using PyVista

The loaded mesh can be plotted using PyVista again.

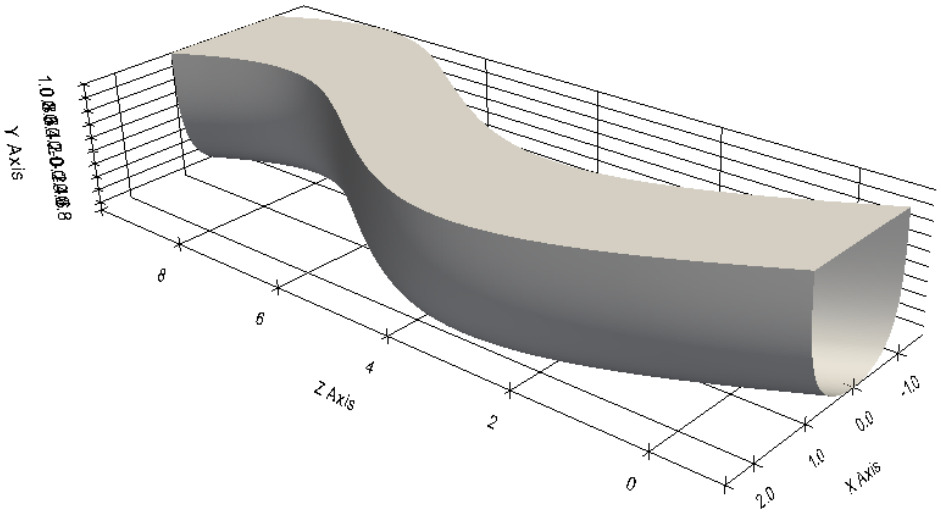
[4]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh,scalar_bar_args=sargs)

p.camera_position=[(15.33958702947096, 9.654312885616765, -9.581353852513592),
 (0.5404866466699564, -0.29141440140763164, 4.2033639107058445),
 (-0.3459193991987702, 0.8968436300281839, 0.2757014191763108)]

p.set_background('white')
p.show_grid(color='black')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:
↪ UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink
```

(continues on next page)

(continued from previous page)

```
Falling back to a static output.
  warnings.warn(
```



6.28.4 Loading the DXF File using GeoPandas

DXF Files can be loaded using GeoPandas. The faces are then stored as `POLYGON Z` objects containing not only the X and Y values but also a Z value. Each polygon is made of 3 vertices of which the start and endpoint are identical (3 different vertices). In order to build the mesh, the vertices need to be extracted. The faces are equal to the single polygons.

[5]:
```
import geopandas as gpd

gdf = gpd.read_file(file_path + 'Channel.dxf')
gdf.drop(['Layer', 'PaperSpace','SubClasses','Linetype', 'EntityHandle', 'Text'],␣
↪axis=1).head()
```

[5]:
```
                                            geometry
0  POLYGON Z ((1.00869 0.92852 1.00000, 0.97744 0...
1  POLYGON Z ((1.00869 0.92852 1.00000, 1.01735 0...
2  POLYGON Z ((0.97744 0.92853 1.00000, 0.94619 0...
3  POLYGON Z ((0.97744 0.92853 1.00000, 0.98610 0...
4  POLYGON Z ((0.94619 0.92853 1.00000, 0.91494 0...
```

Inspecting the Geometries

Each geometry object is a polygon consisting of three unique vertices.

[6]:
```
gdf.loc[0].geometry
```

[6]:

[7]:
```
type(gdf.loc[0].geometry)
```

[7]:
```
shapely.geometry.polygon.Polygon
```

[8]:
```
gdf.loc[0].geometry.wkt
```

[8]:
```
'POLYGON Z ((1.0086873769760132 0.9285249710083008 1, 0.9774374961853027 0.
↪9285261631011963 1, 1.0173505544662476 0.8570554852485657 1.0000001192092896, 1.
↪0086873769760132 0.9285249710083008 1))'
```

Extracting XYZ Coordinates of Polygons

The coordinates for each vertex of each Polygon can be extracted using the regular `extract_xy(..)` function again. The function was adapted to also work with geometries containing a Z component.

[9]:
```
gdf_lines = gg.vector.extract_xy(gdf)
gdf_lines.head()
```

[9]:
```
  Layer PaperSpace SubClasses Linetype EntityHandle  Text  \
0     0       None       None     None         None  None
1     0       None       None     None         None  None
2     0       None       None     None         None  None
3     0       None       None     None         None  None
4     0       None       None     None         None  None

                   geometry    X    Y    Z
0  POINT (1.00869 0.92852) 1.01 0.93 1.00
1  POINT (0.97744 0.92853) 0.98 0.93 1.00
2  POINT (1.01735 0.85706) 1.02 0.86 1.00
3  POINT (1.00869 0.92852) 1.01 0.93 1.00
4  POINT (1.00869 0.92852) 1.01 0.93 1.00
```

Showing vertices

The vertices to create a mesh are equal to the X, Y and Z values of the GeoDataFrame as NumPy array.

[10]:
```
vertices = gdf_lines[['X', 'Y', 'Z']].values
vertices
```

[10]:
```
array([[ 1.00868738,  0.92852497,  1.        ],
       [ 0.9774375 ,  0.92852616,  1.        ],
       [ 1.01735055,  0.85705549,  1.00000012],
       ...,
       [ 0.24864995, -7.21182299, -0.65079874],
       [ 0.23326781, -7.21196413, -0.66666669],
       [ 0.25142166, -7.16530943, -0.66666669]])
```

Showing Faces

The faces for the mesh are equal to the indices of the single points within the GeoDataFrame in the needed VTK format.

[11]:
```
import numpy as np
faces = np.pad(np.arange(0,len(gdf_lines[['X', 'Y', 'Z']].values)).reshape(int(len(gdf_
↪lines[['X', 'Y', 'Z']].values)/4), 4), ((0, 0), (1, 0)), 'constant', constant_values=4)
faces
```

[11]:
```
array([[     4,      0,      1,      2,      3],
       [     4,      4,      5,      6,      7],
       [     4,      8,      9,     10,     11],
       ...,
       [     4, 393204, 393205, 393206, 393207],
       [     4, 393208, 393209, 393210, 393211],
       [     4, 393212, 393213, 393214, 393215]])
```

Creating PolyData

A PyVista PolyData dataset can easily be created with the vertices and faces.

[12]:
```
poly = pv.PolyData(vertices, faces)
poly
```

[12]:
```
PolyData (0x23de4ecb0a0)
  N Cells:  98304
  N Points: 393216
  N Strips: 0
  X Bounds: -1.576e+00, 2.530e+00
  Y Bounds: -9.751e+00, 1.000e+00
  Z Bounds: -9.167e-01, 1.000e+00
  N Arrays: 0
```

Plotting the mesh

As usual, the mesh can be plotted using PyVista.

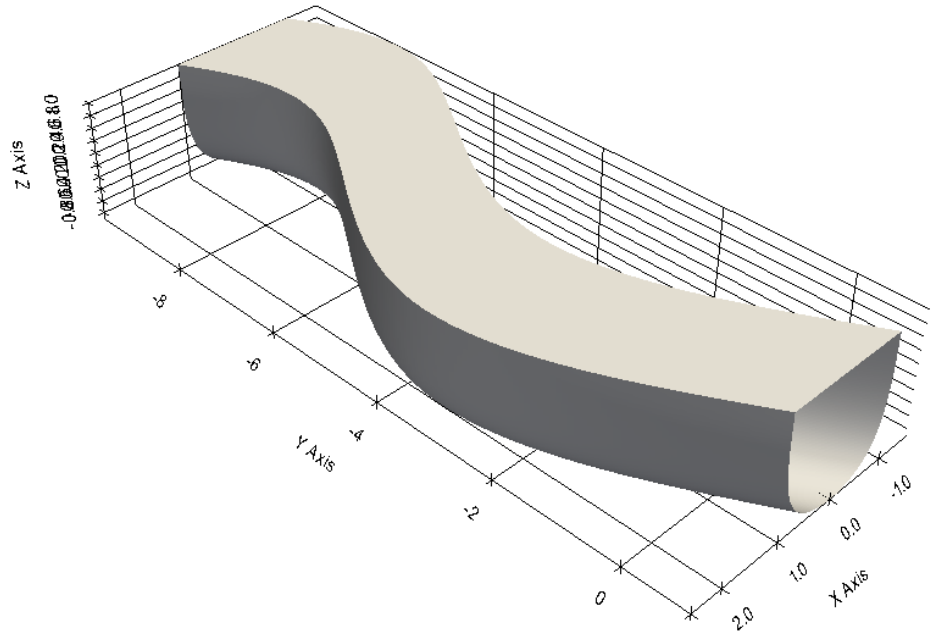
[13]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(poly,scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:
↪ UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink
```

(continues on next page)

(continued from previous page)

```
Falling back to a static output.
  warnings.warn(
```



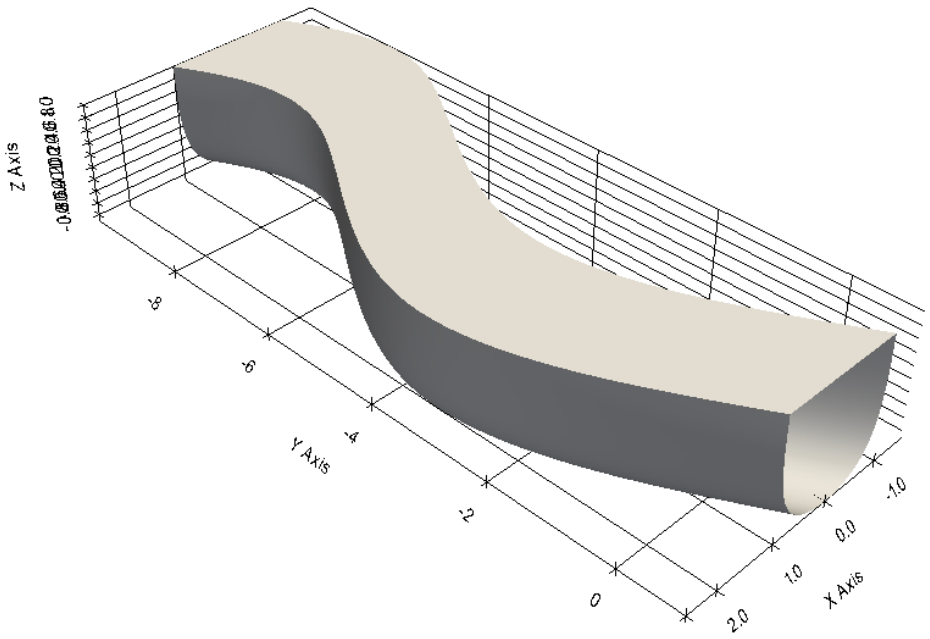
6.28.5 Using the built-in GemGIS Function

The PolyData dataset can also be created using the built-in GemGIS function `create_polydata_from_dxf(...)`.

[14]:
```
poly = gg.visualization.create_polydata_from_dxf(gdf=gdf)
```

[15]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(poly,scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.show()
```

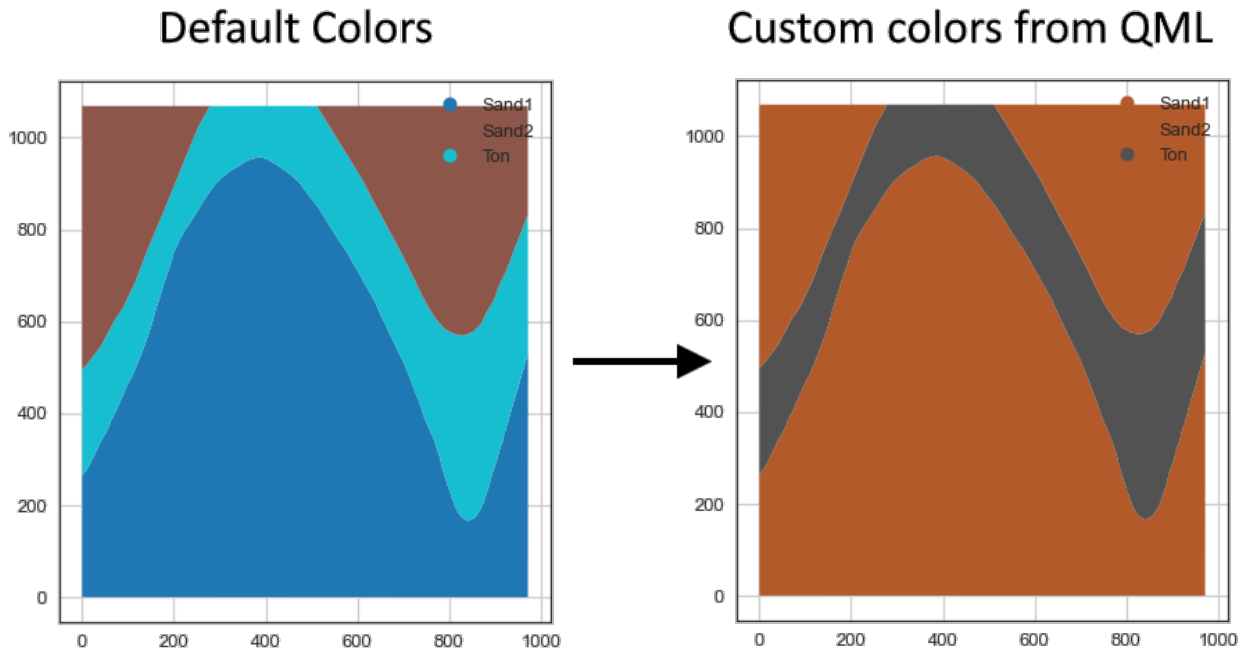
```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:
↪ UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.29 28 Parsing QGIS Style Files to GemGIS

The following will present how to load QGIS style files (QML files) into `GemGIS`, how to set the colors for GeoPandas plots and how to automatically set the surface colors for a GemPy model.

QML is an XML format for storing layer styling. A QML file contains all the information QGIS can handle for the rendering of feature geometries including symbol definitions, sizes and rotations, labelling, opacity and blend mode and more.



Sources: https://docs.qgis.org/3.4/en/docs/user_manual/appendices/qgis_file_formats.html#qml-the-qgis-style-file-format

6.29.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg
file_path ='data/28_parsing_QGIS_style_files/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="28_parsing_QGIS_style_files.zip
↪", dirpath=file_path)
```

```
Downloading file '28_parsing_QGIS_style_files.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F28_parsing_QGIS_style_files.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\tutorial\data\28_parsing_QGIS_style_
↪files'.
```

(continues on next page)

(continued from previous page)

6.29.2 Load Files

A shape file containing Polygons is loaded for demonstration.

[2]:
```
import gemgis as gg
import geopandas as gpd

polygons = gpd.read_file(file_path + 'interfaces_polygons.shp')
polygons
```

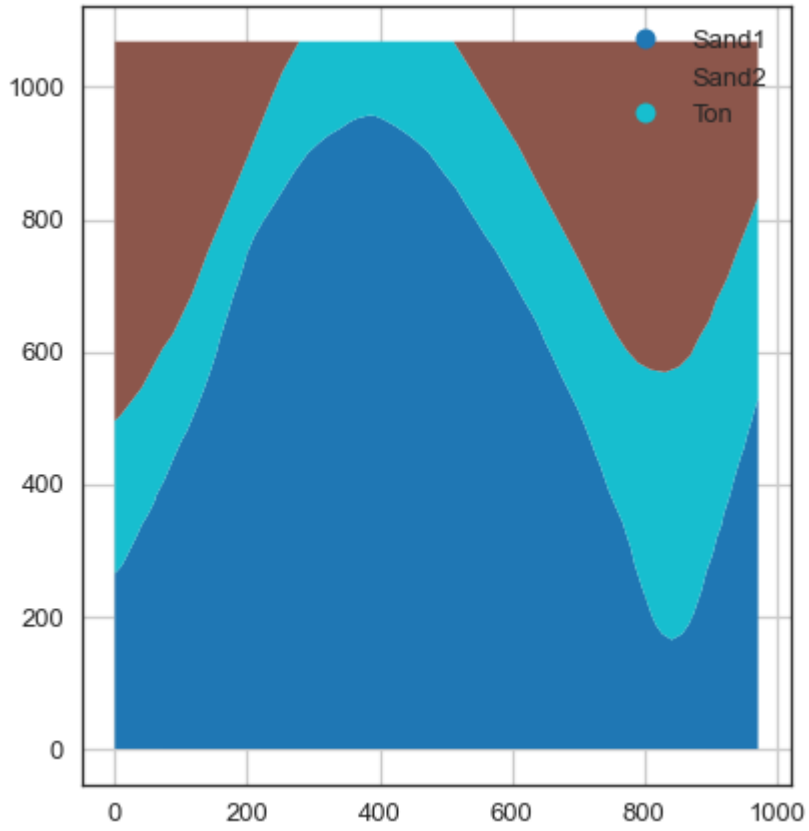
```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:

	id	formation	geometry
0	None	Sand1	POLYGON ((0.256 264.862, 10.593 276.734, 17.13...
1	None	Ton	POLYGON ((0.256 264.862, 0.188 495.787, 8.841 ...
2	None	Sand2	POLYGON ((0.188 495.787, 0.249 1068.760, 278.5...
3	None	Sand2	POLYGON ((511.675 1068.852, 971.698 1068.800, ...

6.29.3 Plotting the data with default colors

[3]:
```
import matplotlib.pyplot as plt

polygons.plot(column='formation', aspect='equal', legend=True)
plt.grid()
```



6.29.4 Loading the QML File

Loading a QML style file. The GeoDataFrame of the plotted polygons have to be provided to create a list of colors. These colors are equal to the `unique()` formations the geological map contains. The surface colors can be loaded with `load_surface_colors(..)` by providing the path to the style file and the corresponding GeoDataFrame.

[4]:
```
cols = gg.utils.load_surface_colors(path=file_path + 'style.qml',
                                    gdf=polygons)
cols
```

[4]:
```
['#b35a2a', '#b35a2a', '#525252']
```

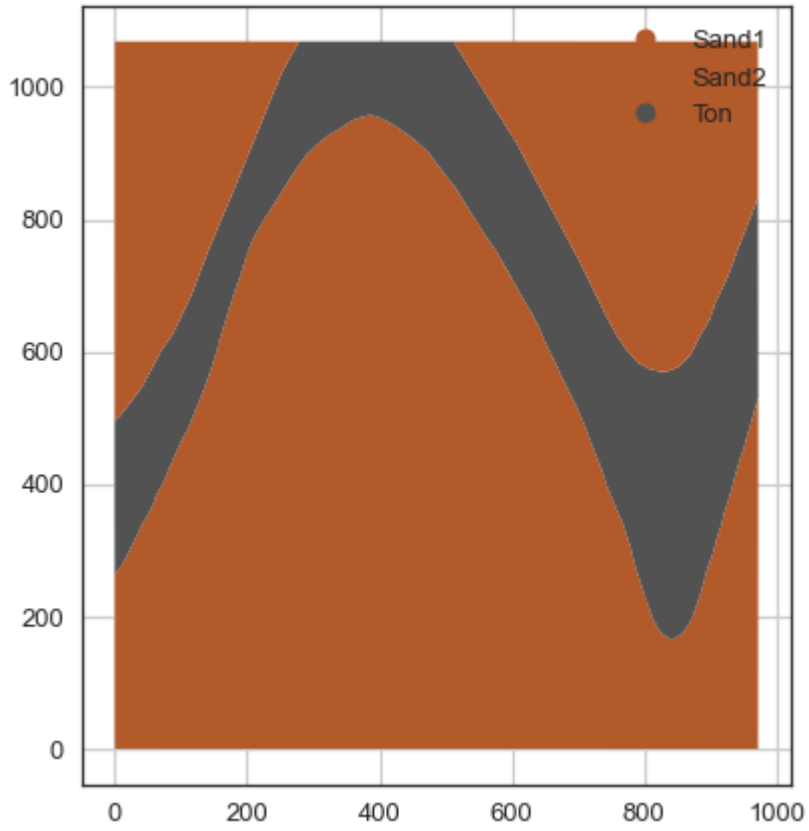
[5]:
```
polygons['formation'].unique()
```

[5]:
```
array(['Sand1', 'Ton', 'Sand2'], dtype=object)
```

6.29.5 Plotting the data with loaded colors

The Polygons can now be plotted using `matplotlib` ListedColormap feature.

[6]:
```
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

polygons.plot(column='formation', aspect='equal', legend=True, cmap=ListedColormap(cols))
plt.grid()
```



6.29.6 Creating surface color dictionary from Style File

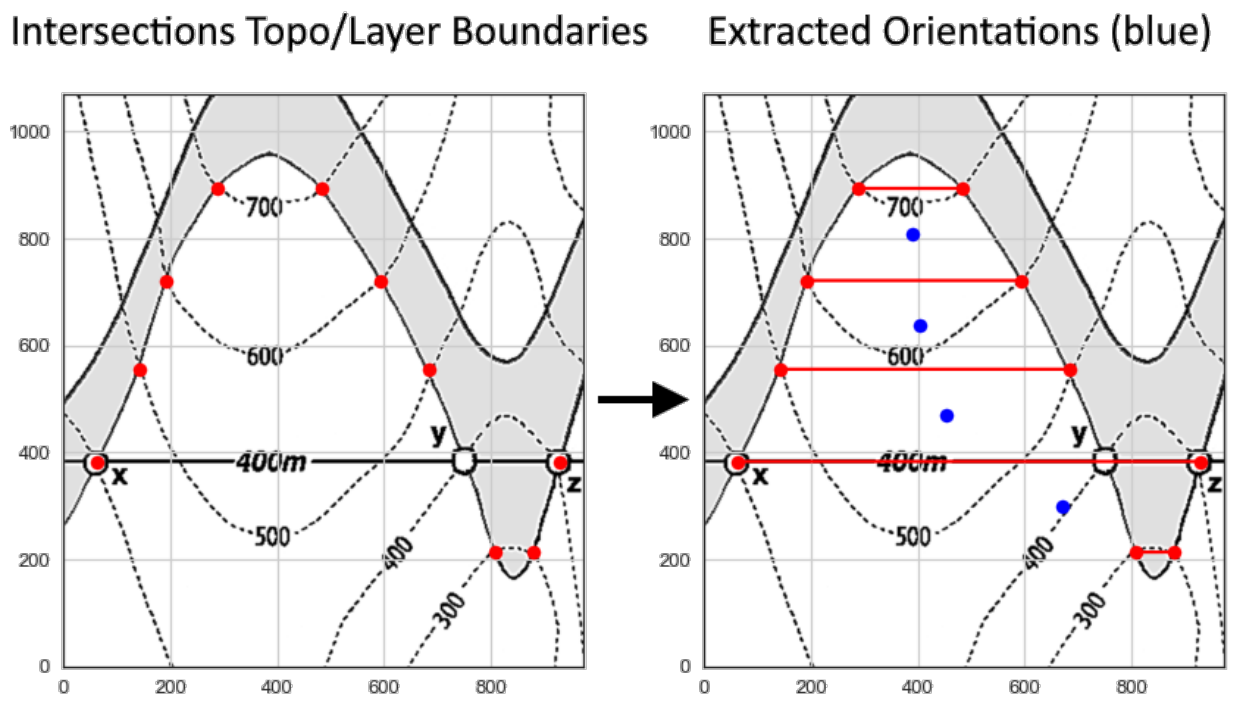
A surface color dict can be created to change the colors of a GemPy Model by using `create_surface_color_dict(...)`.

[7]:
```
surface_color_dict = gg.utils.create_surface_color_dict(path=file_path + 'style.qml')
surface_color_dict
```

[7]:
```
{'Sand1': '#b35a2a', 'Sand2': '#b35a2a', 'Ton': '#525252'}
```

6.30 29 Calculating Orientations from Strike Lines

The following will demonstrate how to calculate orietations based on strike lines on geological maps. These orientations can be used to create a GemPy model.



Source: Powell, D. (1995): Interpretation geologischer Strukturen durch Karten - Eine praktische Anleitung mit Aufgaben und Lösungen, page 15, figure 10 A, Springer Verlag Berlin, Heidelberg, New York, ISBN: 978-3-540-58607-4.

6.30.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/29_calculating_orientations_from_strike_lines/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
```

(continues on next page)

(continued from previous page)

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="29_calculating_orientations_
↪from_strike_lines.zip", dirpath=file_path)
```

```
Downloading file '29_calculating_orientations_from_strike_lines.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F29_calculating_orientations_from_
↪strike_lines.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\
↪data\29_calculating_orientations_from_strike_lines'.
```

6.30.2 Loading Data

A shape file containing the intersection points between topographic contours and layer boundaries is loaded as GeoDataFrame.

[2]:
```
import geopandas as gpd
import rasterio

gdf = gpd.read_file(file_path + 'points_strike.shp')
gdf
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:

	id	Z	formation	geometry
0	2	400	Ton	POINT (62.513 382.832)
1	3	500	Ton	POINT (141.489 555.155)
2	4	600	Ton	POINT (191.943 720.949)
3	5	700	Ton	POINT (287.541 893.302)
4	5	700	Ton	POINT (481.908 893.302)
5	4	600	Ton	POINT (594.500 720.942)
6	3	500	Ton	POINT (684.515 555.153)
7	1	300	Ton	POINT (807.441 213.514)
8	1	300	Ton	POINT (878.392 213.516)
9	2	400	Ton	POINT (927.489 382.827)

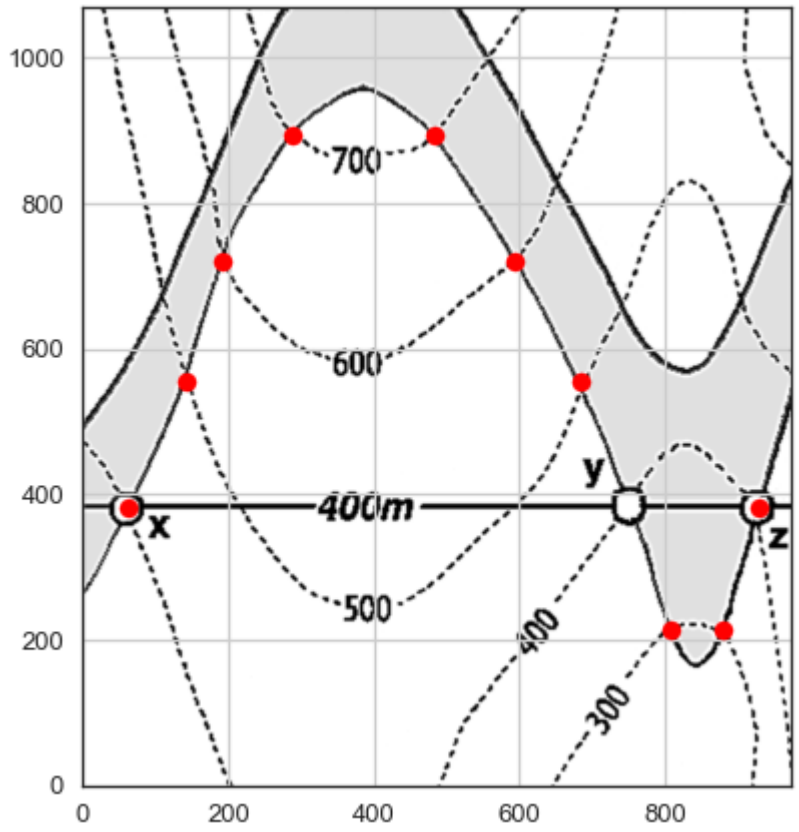
Loading the raster containing the geological map.

[3]:
```
raster = rasterio.open(file_path + 'raster.tif')
```

6.30.3 Plotting the data

It can be seen that the loaded points are equal to the intersections between contour lines and layer boundaries.

[4]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1)

ax.imshow(raster.read(1), extent=[0,972,0,1069], cmap='gray')

gdf.plot(ax=ax, aspect='equal', color='red')

plt.grid()
```



6.30.4 Creating LineStrings from Points

LineStrings are created from the points to create strike lines.

[5]:
```
gdf
```

[5]:
```
   id    Z formation                 geometry
0   2  400       Ton   POINT (62.513 382.832)
1   3  500       Ton  POINT (141.489 555.155)
2   4  600       Ton  POINT (191.943 720.949)
3   5  700       Ton  POINT (287.541 893.302)
4   5  700       Ton  POINT (481.908 893.302)
```

(continues on next page)

(continued from previous page)

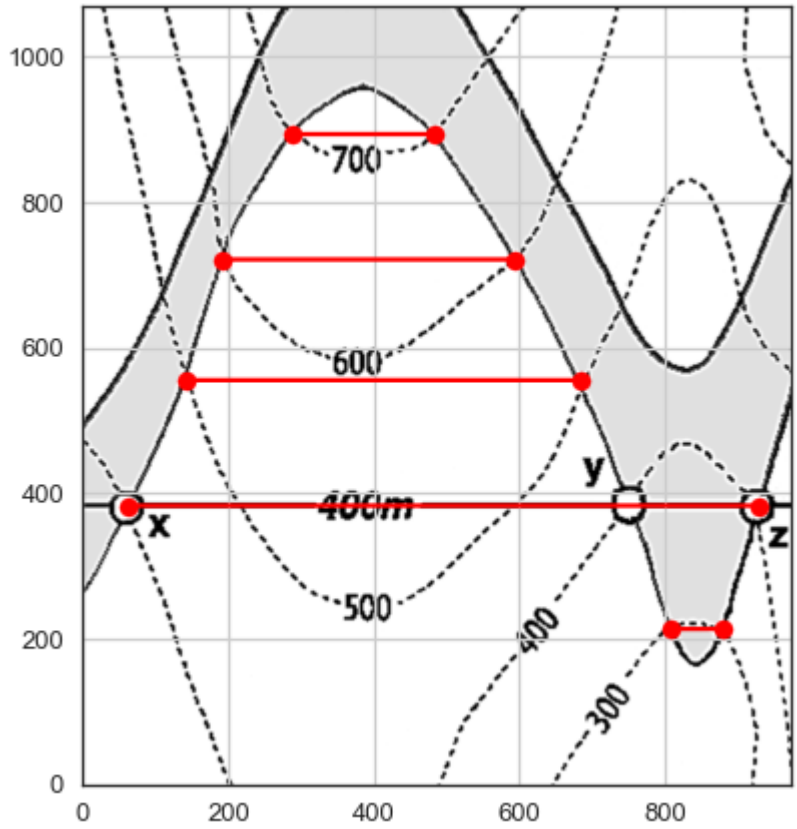
```
5   4  600       Ton  POINT (594.500 720.942)
6   3  500       Ton  POINT (684.515 555.153)
7   1  300       Ton  POINT (807.441 213.514)
8   1  300       Ton  POINT (878.392 213.516)
9   2  400       Ton  POINT (927.489 382.827)
```

[6]:
```
linestring_gdf = gg.vector.create_linestring_gdf(gdf=gdf)
linestring_gdf
```

[6]:

	index	id	Z	formation	geometry
0	7	1	300	Ton	LINESTRING (807.441 213.514, 878.392 213.516)
1	0	2	400	Ton	LINESTRING (62.513 382.832, 927.489 382.827)
2	1	3	500	Ton	LINESTRING (141.489 555.155, 684.515 555.153)
3	2	4	600	Ton	LINESTRING (191.943 720.949, 594.500 720.942)
4	3	5	700	Ton	LINESTRING (287.541 893.302, 481.908 893.302)

[7]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1)

ax.imshow(raster.read(1), extent=[0,972,0,1069], cmap='gray')

gdf.plot(ax=ax, aspect='equal', color='red')
linestring_gdf.plot(ax=ax, aspect='equal', color='red')

plt.grid()
```



6.30.5 Calculating Orientations

Orientations are calculated for one formation with eigenvector analysis. The orientations are then returned for the midpoint of each strike line and the DataFrame can directly be used for GemPy. Examples with multiple formations follow.

[8]:
```
linestring_gdf.is_valid
```

[8]:
```
0    True
1    True
2    True
3    True
4    True
dtype: bool
```

[9]:
```
orientations = gg.vector.calculate_orientations_from_strike_lines(gdf=linestring_gdf)
orientations
```

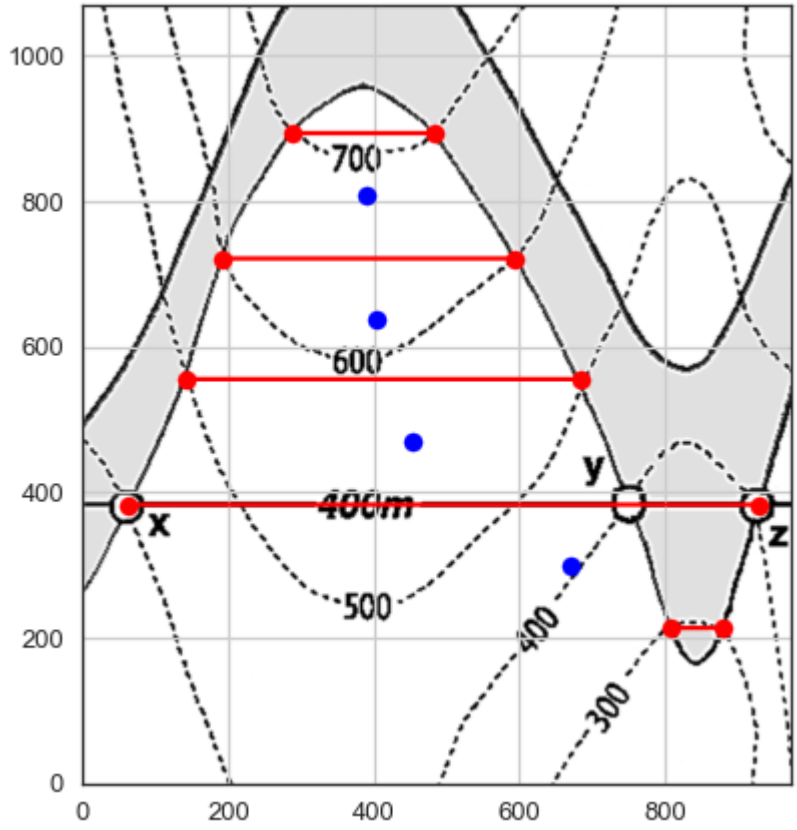
[9]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 30.57   180.00 350.00  POINT (668.959 298.172)      1.00       Ton 668.96
1 30.13   180.00 450.00  POINT (454.001 468.992)      1.00       Ton 454.00
2 31.10   180.00 550.00  POINT (403.112 638.050)      1.00       Ton 403.11
3 30.12   180.00 650.00  POINT (388.973 807.124)      1.00       Ton 388.97

       Y
```

(continues on next page)

(continued from previous page)

```
0 298.17
1 468.99
2 638.05
3 807.12
```

[10]:
```
import matplotlib.pyplot as plt

fig, ax = plt.subplots(1,1)

ax.imshow(raster.read(1), extent=[0,972,0,1069], cmap='gray')

gdf.plot(ax=ax, aspect='equal', color='red')
linestring_gdf.plot(ax=ax, aspect='equal', color='red')
orientations.plot(ax=ax, aspect='equal', color='blue')

plt.grid()
```



6.31 30 Opening GeoDataBases for GemGIS

A GeoDataBase folder containing raster or vector data or other type of formats can be opened using fiona and GeoPandas. The different layers can be listed using fiona and opened separately using GeoPandas to return GeoDataFrames for further processing.

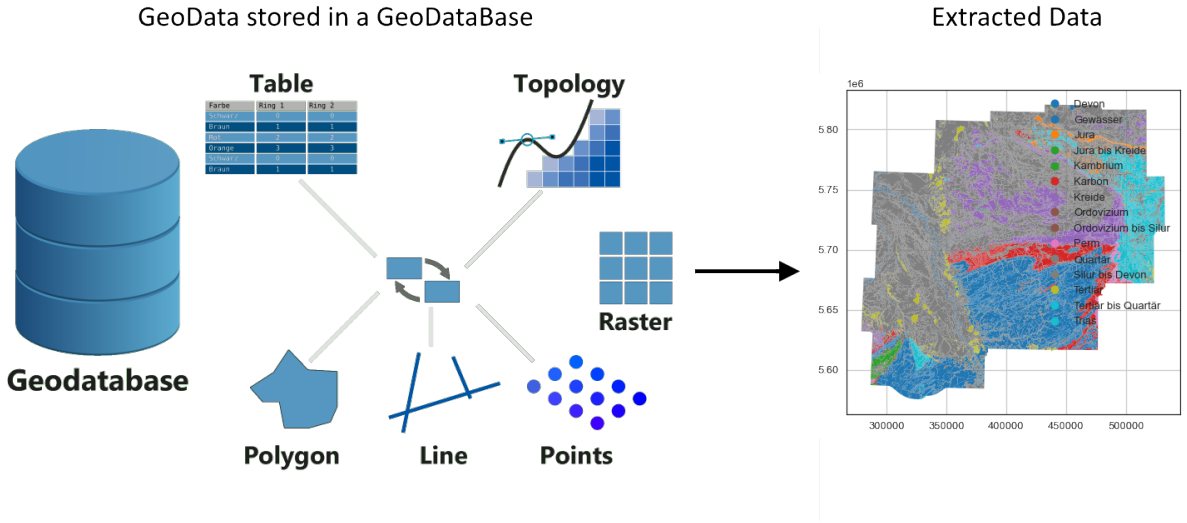


Image Source: https://gisgeography.com/geodatabase-personal-file/ Map Source: https://open.nrw/dataset/11ed9860-d540-4f77-8af8-97cee14ce91f

6.31.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg
file_path ='data/30_opening_geodatabases_for_gemgis/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="30_opening_geodatabases_for_
↪gemgis.zip", dirpath=file_path)
```

```
Downloading file '30_opening_geodatabases_for_gemgis.zip' from 'https://rwth-aachen.
↪sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F30_opening_geodatabases_for_gemgis.zip'␣
↪to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\data\30_opening_
↪geodatabases_for_gemgis'.
```

6.31.2 Load Layer Data

The used GeoDataBase was downloaded from https://www.opengeodata.nrw.de/produkte/geologie/geologie/GK/ISGK100/ISGK100vektor/ and will be used under Datenlizenz Deutschland Namensnennung 2.0.

A list of the different layers within the GeoDataBase can be obtained using fionas `listlayers(..)` function. Subsequently, the different layers in the GeoDataBase can be loaded as GeoDataFrames using GeoPandas.

[10]:
```
import geopandas as gpd
import fiona

layer_list = fiona.listlayers(file_path + 'ISGK100.gdb')
layer_list
```

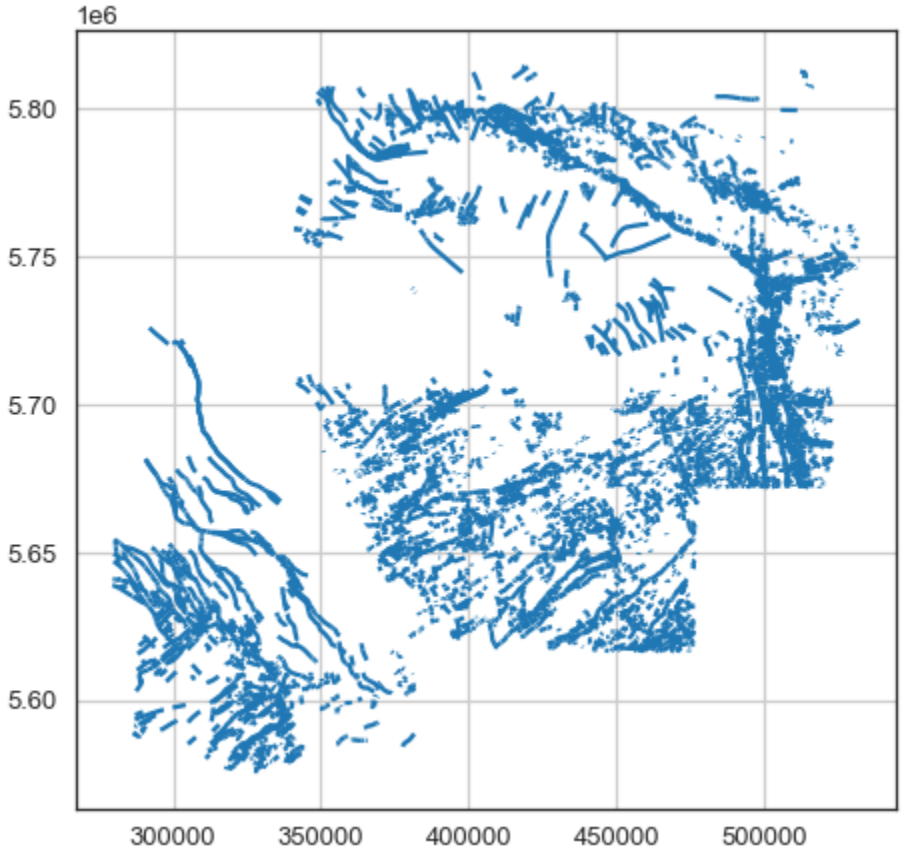
[10]:
```
['GK100_Tektonik', 'GK100_Hauptschichten', 'GK100_Deckschichten']
```

6.31.3 Loading and Plotting different Layers of GeoDataBase

[25]:
```
data = gpd.read_file(file_path + "ISGK100.gdb", driver='FileGDB', layer=layer_list[0])
data.head()
```

[25]:
```
                        TYP     DATUM  SHAPE_Length  \
0  Überschiebung, gesichert  20140911        809.45
1  Überschiebung, gesichert  20140911        695.09
2  Überschiebung, gesichert  20140911       2021.76
3  Überschiebung, gesichert  20140911       1016.71
4  Überschiebung, gesichert  20140911       1085.76

                                          geometry
0  MULTILINESTRING ((400321.058 5653696.880, 4001...
1  MULTILINESTRING ((399433.510 5653415.030, 3992...
2  MULTILINESTRING ((402635.674 5654612.215, 4023...
3  MULTILINESTRING ((403501.564 5651765.422, 4037...
4  MULTILINESTRING ((404140.075 5652650.414, 4050...
```

[24]:
```
import matplotlib.pyplot as plt

data.plot()
plt.grid()
```



[26]:
```
data = gpd.read_file(file_path + "ISGK100.gdb", driver='FileGDB', layer=layer_list[1])
data.head()
```

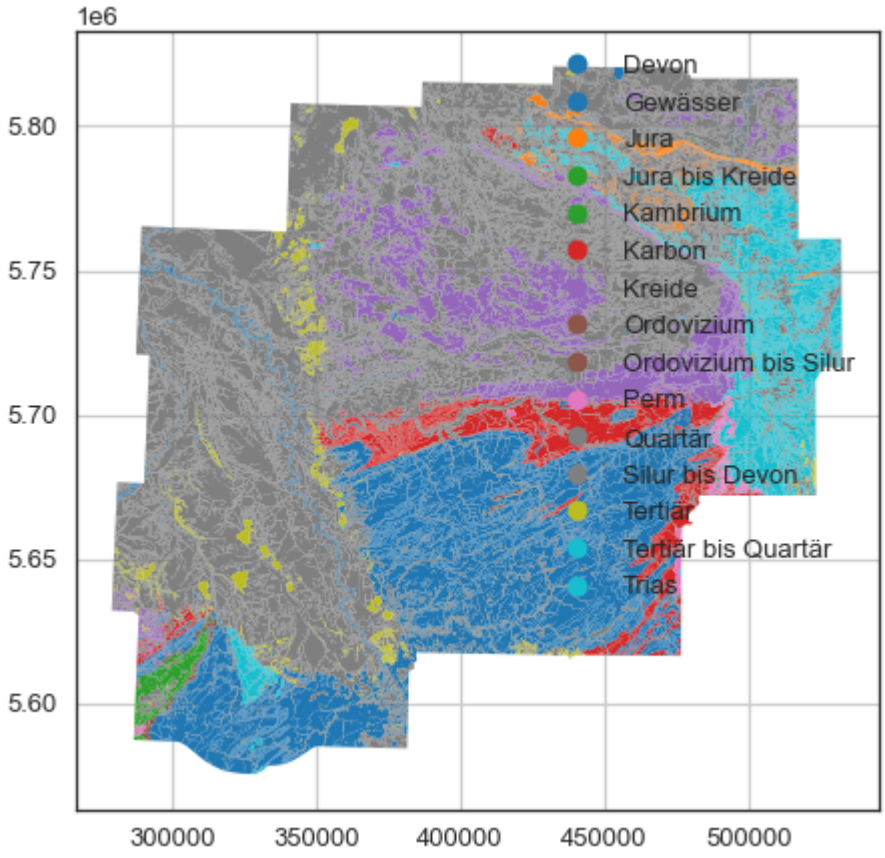
[26]:
```
     ID                EINHEIT SYMBOL  \
0  4109  Sprockhövel-Schichten    cnS
1  4109  Sprockhövel-Schichten    cnS
2  4109  Sprockhövel-Schichten    cnS
3  4109  Sprockhövel-Schichten    cnS
4  4109  Sprockhövel-Schichten    cnS

                                        LITHOLOGIE  \
0  Schluff- und Tonstein, schwach bis stark sandi...
1  Schluff- und Tonstein, schwach bis stark sandi...
2  Schluff- und Tonstein, schwach bis stark sandi...
3  Schluff- und Tonstein, schwach bis stark sandi...
4  Schluff- und Tonstein, schwach bis stark sandi...

                     LITHOLOGIE_KURZ     STEINART  SYSTEM       SERIE  \
0  Tonstein, Schluffstein, Steinkohle  Festgestein  Karbon  Oberkarbon
1  Tonstein, Schluffstein, Steinkohle  Festgestein  Karbon  Oberkarbon
2  Tonstein, Schluffstein, Steinkohle  Festgestein  Karbon  Oberkarbon
3  Tonstein, Schluffstein, Steinkohle  Festgestein  Karbon  Oberkarbon
4  Tonstein, Schluffstein, Steinkohle  Festgestein  Karbon  Oberkarbon
```

(continues on next page)

(continued from previous page)

```
     STUFE                        LEGENDE     DATUM  SHAPE_Length  SHAPE_Area  \
0  Namur C  4109, Sprockhövel-Schichten  20140911       3008.74   172425.08
1  Namur C  4109, Sprockhövel-Schichten  20140911       2585.06   200323.96
2  Namur C  4109, Sprockhövel-Schichten  20140911       4025.19   300520.77
3  Namur C  4109, Sprockhövel-Schichten  20140911       2634.66   448870.72
4  Namur C  4109, Sprockhövel-Schichten  20140911       2260.89   143180.71

                                              geometry
0  MULTIPOLYGON (((352879.326 5697940.476, 352924...
1  MULTIPOLYGON (((392856.502 5695841.097, 392721...
2  MULTIPOLYGON (((358582.349 5698046.058, 358435...
3  MULTIPOLYGON (((391719.954 5695960.055, 391591...
4  MULTIPOLYGON (((375148.274 5696852.140, 375087...
```

[22]:
```
import matplotlib.pyplot as plt

data.plot(column='SYSTEM', legend=True)
plt.grid()
```



[27]:
```
data = gpd.read_file(file_path + "ISGK100.gdb", driver='FileGDB', layer=layer_list[2])
data.head()
```

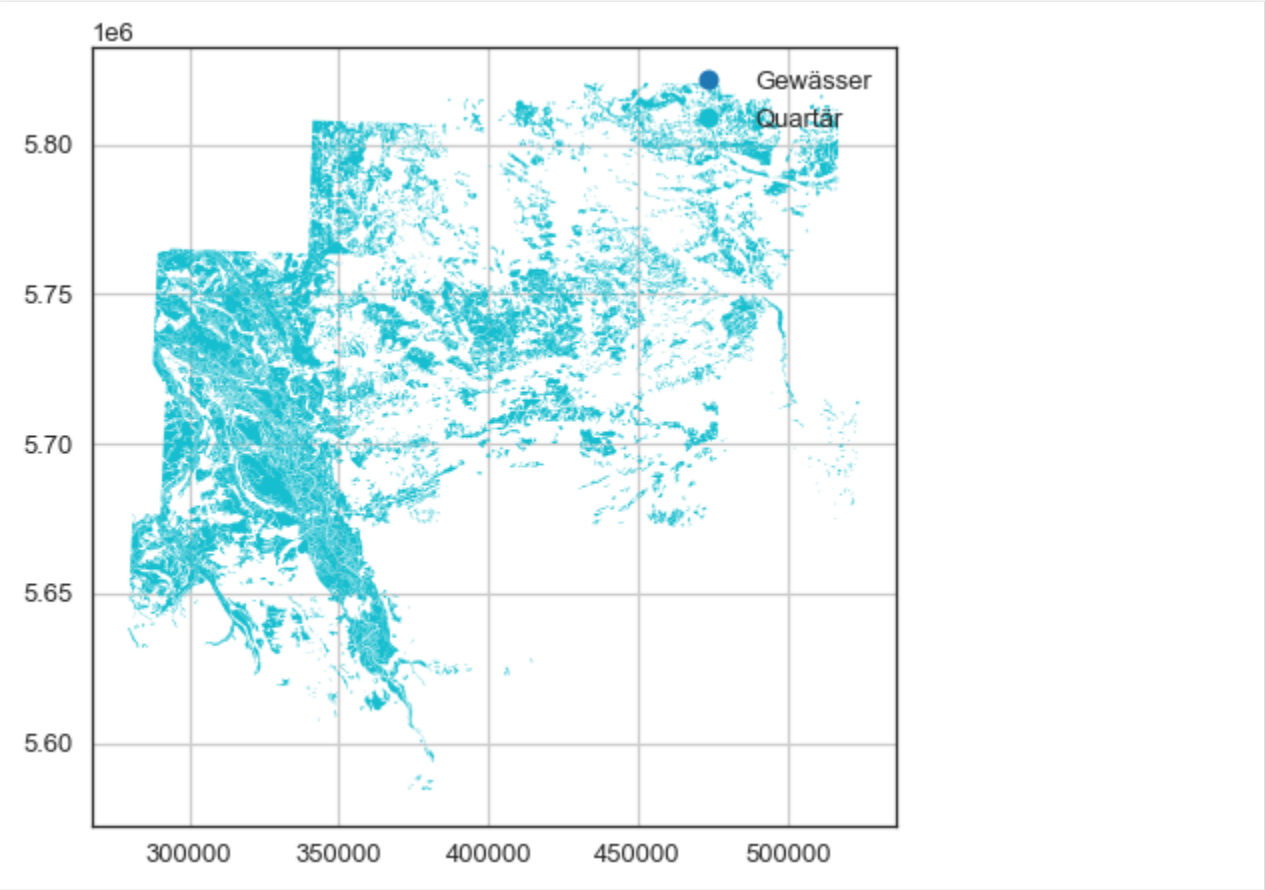
[27]:
```
     ID                              EINHEIT SYMBOL  \
0  1132                             Auenlehm     Lf
```

(continues on next page)

(continued from previous page)

```
1  1132                            Auenlehm     Lf
2  1132                            Auenlehm     Lf
3  1132                            Auenlehm     Lf
4  1203  Hanglehm, Hangschutt und Fließerde     hg

                                          LITHOLOGIE LITHOLOGIE_KURZ  \
0  Schluff und Ton, sandig, z.T. kalkhaltig, grau...    Schluff, Ton
1  Schluff und Ton, sandig, z.T. kalkhaltig, grau...    Schluff, Ton
2  Schluff und Ton, sandig, z.T. kalkhaltig, grau...    Schluff, Ton
3  Schluff und Ton, sandig, z.T. kalkhaltig, grau...    Schluff, Ton
4  Schluff, tonig, sandig, grusig, steinig, braun...         Schluff

        STEINART   SYSTEM                   SERIE STUFE  \
0  Lockergestein  Quartär                Holozän  None
1  Lockergestein  Quartär                Holozän  None
2  Lockergestein  Quartär                Holozän  None
3  Lockergestein  Quartär                Holozän  None
4  Lockergestein  Quartär  Pleistozän bis Holozän  None

                                   LEGENDE     DATUM  SHAPE_Length  \
0                           1132, Auenlehm  20140911      19229.40
1                           1132, Auenlehm  20140911       3177.70
2                           1132, Auenlehm  20140911       5298.13
3                           1132, Auenlehm  20140911       2120.11
4  1203, Hanglehm, Hangschutt und Fließerde  20140911       3788.10

   SHAPE_Area                                           geometry
0  1569937.10  MULTIPOLYGON (((322257.871 5728988.642, 322320...
1   348742.93  MULTIPOLYGON (((323863.620 5728974.644, 323892...
2   755121.53  MULTIPOLYGON (((323481.893 5729792.005, 323501...
3   274024.58  MULTIPOLYGON (((322560.368 5730041.085, 322782...
4   859667.50  MULTIPOLYGON (((472748.270 5713137.549, 472740...
```

[19]:
```
import matplotlib.pyplot as plt

data.plot(column='SYSTEM', legend=True)
plt.grid()
```



6.32 31 Obtaining City Locations

Locations of cities for easy plotting can be obtained using GeoPy with GemGIS.

6.32.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/31_obtaining_location_information/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
```

(continues on next page)

(continued from previous page)

```
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.32.2 Getting Location Information

Location information from different cities can be obtained using GeoPy. These contain a description of the location as well as longitude and latitude values.

[2]:
```
aachen = gg.utils.get_location_coordinate(name='Aachen')
aachen
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
Location(Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland, (50.776351, 6.
↪083862, 0.0))
```

[3]:
```
berlin = gg.utils.get_location_coordinate(name='Berlin')
berlin
```

[3]:
```
Location(Berlin, 10117, Deutschland, (52.5170365, 13.3888599, 0.0))
```

6.32.3 Reprojecting Coordinates

The WGS 84 coordinates can be reprojected using `transform_location_coordinate(..)`.

[4]:
```
aachen_repr = gg.utils.transform_location_coordinate(coordinates=aachen,
                                                     crs='EPSG:4647')
aachen_repr
```

[4]:
```
{'Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland': (32294411.33488576,
  5629009.357074926)}
```

[5]:
```
berlin_repr = gg.utils.transform_location_coordinate(coordinates=berlin,
                                                     crs='EPSG:4647')
berlin_repr
```

[5]:
```
{'Berlin, 10117, Deutschland': (32797738.56053437, 5827603.740024588)}
```

6.32.4 Creating Bounding Polygon from Location

Each location also consists of a bounding box which can be used to create a Shapely Polygon from it using `create_polygon_from_location(..)`.

[6]:
```
aachen_bbox = gg.utils.create_polygon_from_location(coordinates=aachen)

print(aachen_bbox.bounds)
aachen_bbox
```

```
(50.6621373, 5.9748624, 50.8572449, 6.2180747)
```

[6]:

[7]:
```
berlin_bbox = gg.utils.create_polygon_from_location(coordinates=berlin)

print(berlin_bbox.bounds)
berlin_bbox
```

```
(52.3570365, 13.2288599, 52.6770365, 13.5488599)
```

[7]:

6.32.5 Getting the locations of multiple cities

The function `get_locations(..)` allows to pass a list of cities and returns a dict with the city names and the corresponding coordinates.

[8]:
```
coordinates_dict = gg.utils.get_locations(names = ['Aachen', 'Berlin', 'München',
↪'Hamburg', 'Köln'], crs='EPSG:4647')
coordinates_dict
```

[8]:
```
{'Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland': (32294411.33488576,
  5629009.357074926),
 'Berlin, 10117, Deutschland': (32797738.56053437, 5827603.740024588),
 'München, Bayern, Deutschland': (32691595.356409974, 5334747.274305081),
 'Hamburg, Deutschland': (32566296.251301013, 5933959.964708338),
 'Köln, Nordrhein-Westfalen, Deutschland': (32356668.818424627,
  5644952.099932303)}
```

6.32.6 Converting Location Dict into GeoDataFrame

In order to work with the coordinates and location names, the dict can be converted to a GeoDataFrame.

[9]:
```
import geopandas as gpd
gdf = gg.utils.convert_location_dict_to_gdf(location_dict=coordinates_dict)
gdf
```

[9]:

	City	X	Y	geometry
0	Aachen	32294411.33	5629009.36	POINT (32294411.335 5629009.357)
1	Berlin	32797738.56	5827603.74	POINT (32797738.561 5827603.740)
2	München	32691595.36	5334747.27	POINT (32691595.356 5334747.274)
3	Hamburg	32566296.25	5933959.96	POINT (32566296.251 5933959.965)
4	Köln	32356668.82	5644952.10	POINT (32356668.818 5644952.100)

6.33 32 Using ipyvtk with PyVista for Visualization

6.33.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/32_using_ipyvtk_with_pyvista_for_visualization/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="32_using_ipyvtk_with_pyvista_
↪for_visualization.zip", dirpath=file_path)
```

```
Downloading file '32_using_ipyvtk_with_pyvista_for_visualization.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F32_using_ipyvtk_with_pyvista_for_
↪visualization.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\
↪tutorial\data\32_using_ipyvtk_with_pyvista_for_visualization'.
```

6.33.2 Load Data

[3]:
```
import pyvista as pv

mesh1 = pv.read(file_path + 'mesh1.vtk')
mesh2 = pv.read(file_path + 'mesh2.vtk')
```

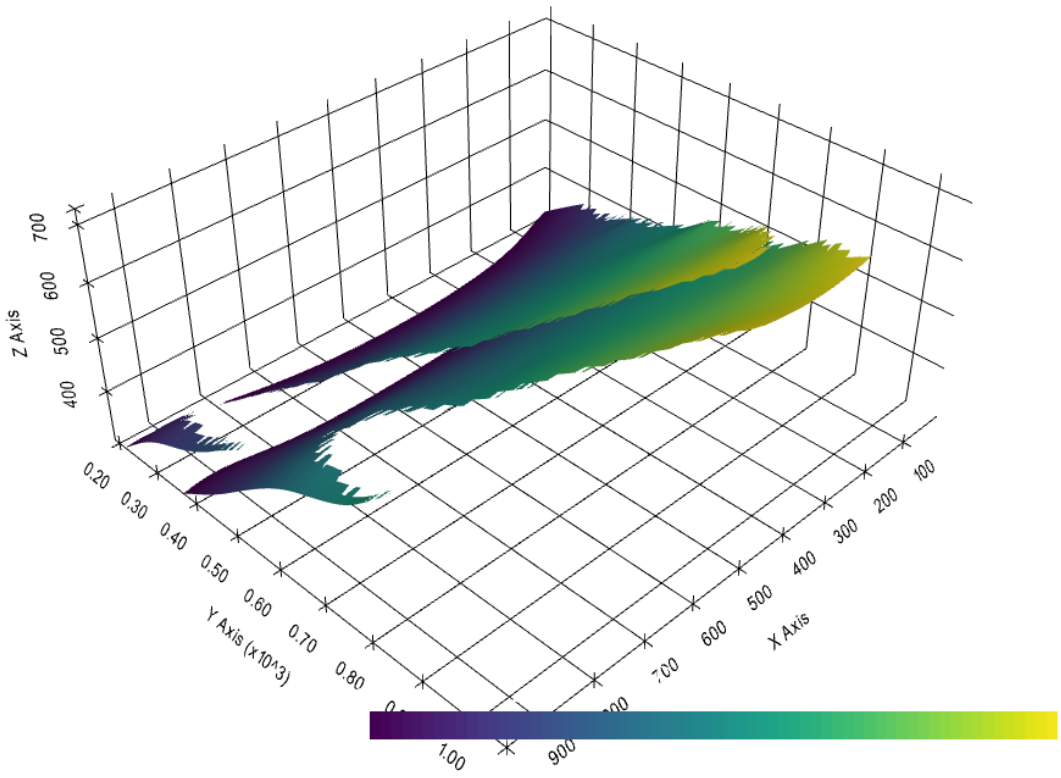
6.33.3 Plotting Data with PyVista

[7]:
```
p = pv.Plotter(notebook=True)

p.add_mesh(mesh1)
p.add_mesh(mesh2)

p.set_background('white')
```

(continues on next page)

(continued from previous page)

```
p.show_grid(color='black')
p.show()
```



[11]:
```
meshes = pv.MultiBlock([mesh1, mesh2])
```

[ ]:
```
p = pv.Plotter(notebook=False)

p.add_mesh_clip_plane(meshes)

p.set_background('white')
p.show_grid(color='black')
p.show()
```

6.34 33 Slicing Geological Models with PyVista

6.34.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[2]:
```
import gemgis as gg
file_path ='data/33_slicing_geological_models_with_pyvista/'
```

6.34.2 TUTORIAL NOT AVAILABLE

6.35 34 Interpolating Strike Lines with GemGIS

6.35.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/34_interpolating_strike_lines_with_gemgis/'
```

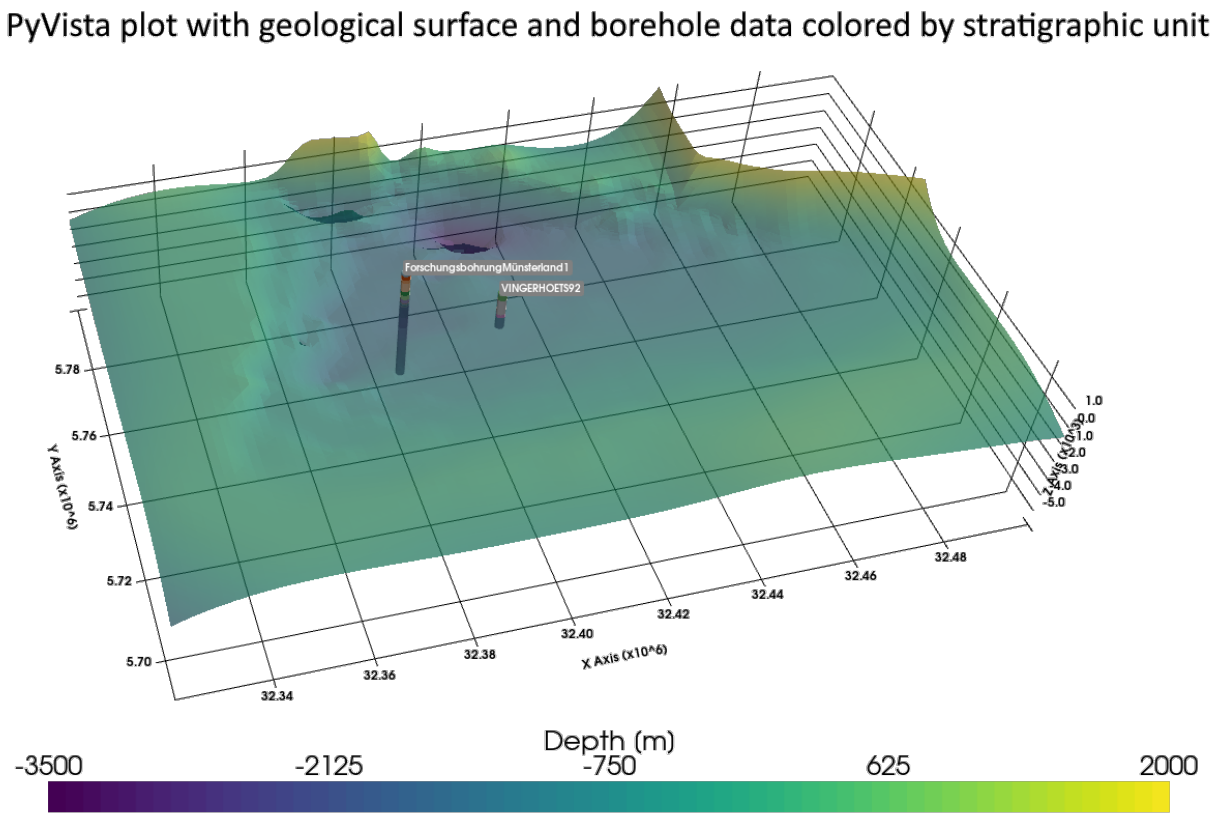
```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.35.2 TUTORIAL NOT AVAILABLE

6.36 35 Plotting Borehole Data with PyVista

Borehole data or stratigraphic data can be plotted using GemGIS and PyVista.

Well data provided by the Geological Survey NRW.



6.36.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/35_plotting_borehole_data_with_pyvista/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
```

(continues on next page)

(continued from previous page)

```
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="35_plotting_borehole_data_with_
↪pyvista.zip", dirpath=file_path)
```

6.36.2 Loading Data

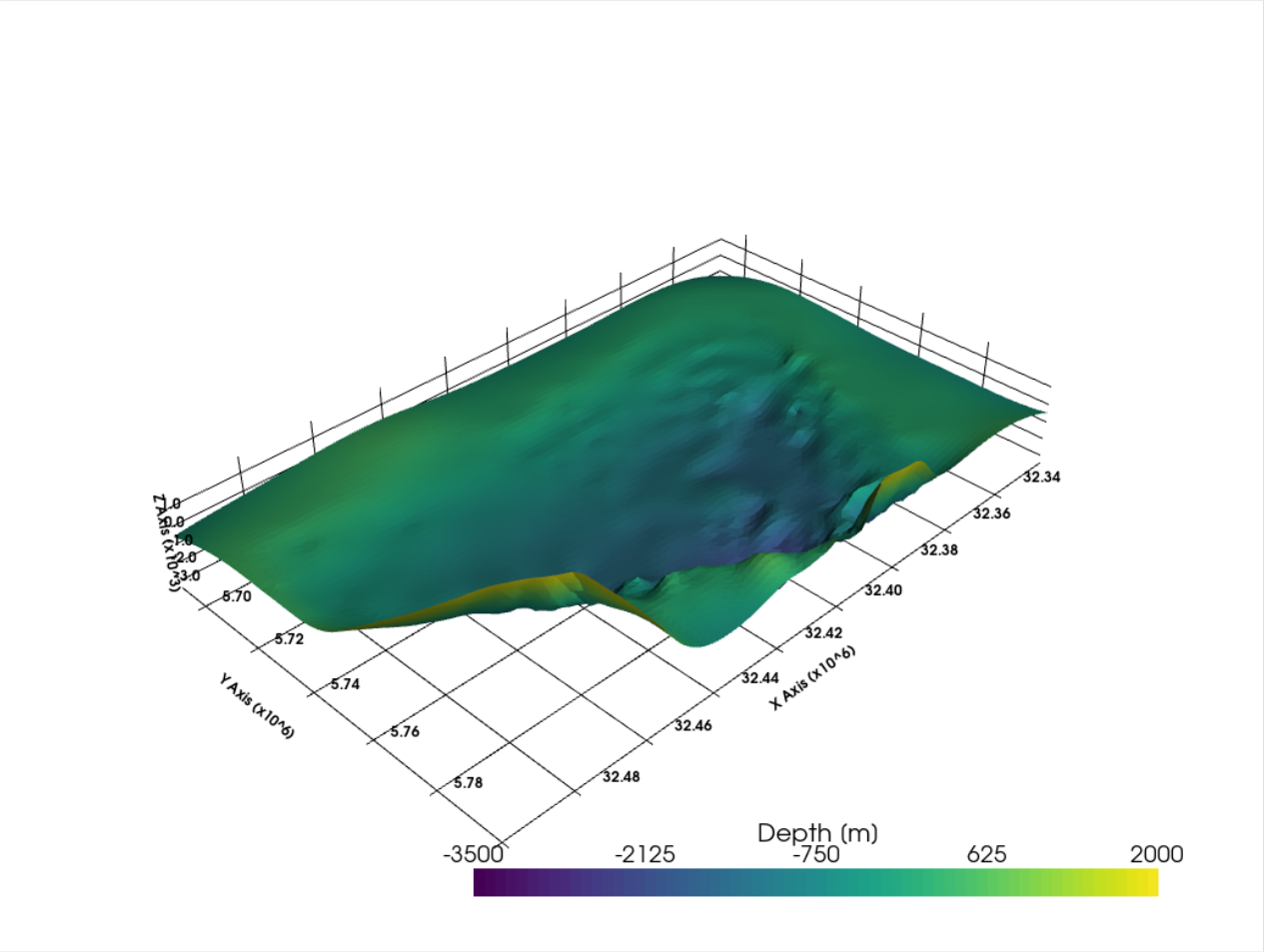
For better visualization, a mesh is loaded displaying a surface in the subsurface. In addition, sample well data is loaded to demonstrate the visualization of well data.

[2]:
```
import pyvista as pv
import pandas as pd

mesh = pv.read(file_path + 'base.vtk')
mesh
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
PolyData (0x1f76053edc0)
  N Cells:  23095
  N Points: 11782
  X Bounds: 3.232e+07, 3.250e+07
  Y Bounds: 5.691e+06, 5.799e+06
  Z Bounds: -3.445e+03, 1.945e+03
  N Arrays: 1
```

[3]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh,scalar_bar_args=sargs, clim=[-3500, 2000])

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,5)
p.show()
```



[4]:
```
data = pd.read_csv(file_path + 'Borehole_Data.csv')
data.head()
```

[4]:

	Unnamed: 0	Index	Name	X	Y	\
0	2091	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
1	2092	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
2	2093	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
3	2094	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
4	2095	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	

	Z	Altitude	Depth	formation	geometry
0	27.00	107.00	5956.00	OberCampanium	POINT (32386176.36 5763283.15)
1	-193.00	107.00	5956.00	UnterCampanium	POINT (32386176.36 5763283.15)
2	-268.00	107.00	5956.00	OberSantonium	POINT (32386176.36 5763283.15)
3	-828.00	107.00	5956.00	MittelSantonium	POINT (32386176.36 5763283.15)
4	-988.00	107.00	5956.00	UnterSantonium	POINT (32386176.36 5763283.15)

[5]:
```
data['formation'].unique()
```

[5]:
```
array(['OberCampanium', 'UnterCampanium', 'OberSantonium',
       'MittelSantonium', 'UnterSantonium', 'OberConiacium',
       'UnterConiacium', 'MittelTuronium', 'UnterTuronium',
```

(continues on next page)

(continued from previous page)

```
       'OberCenomanium', 'MittelCenomanium', 'UnterCenomanium',
       'OberAlbium', 'MittelAlbium', 'EssenFM', 'BochumFM', 'WittenFM',
       'Carboniferous', 'Devonian', 'Quaternary', 'MittelConiacium'],
      dtype=object)
```

6.36.3 Defining Color Dict

[6]:
```
color_dict = {
        'OberCampanium':'#3182bd', 'UnterCampanium':'#9ecae1',
        'OberSantonium': '#e6550d', 'MittelSantonium': '#fdae6b', 'UnterSantonium': '
↪#fdd0a2',
        'OberConiacium': '#31a354', 'MittelConiacium': '#74c476', 'UnterConiacium': '
↪#a1d99b',
        'OberTuronium': '#756bb1', 'MittelTuronium': '#9e9ac8', 'UnterTuronium': '#9e9ac8
↪',
        'OberCenomanium': '#636363', 'MittelCenomanium': '#969696', 'UnterCenomanium': '
↪#d9d9d9',
        'OberAlbium': '#637939', 'MittelAlbium': '#8ca252',
        'EssenFM': '#e7969c', 'BochumFM': '#7b4173', 'WittenFM': '#a55194',
        'Carboniferous': '#de9ed6', 'Devonian': '#de9ed6', 'Quaternary': '#de9ed6',
        }
```

6.36.4 Adding a row to the wells

[7]:
```
grouped = data.groupby(['Index'])
df_groups = [grouped.get_group(x) for x in grouped.groups]

list_df = gg.visualization.add_row_to_boreholes(df_groups)

list_df[0].head()
```

```
100%|| 2/2 [00:00<00:00, 105.22it/s]
```

[7]:

	Unnamed: 0	Index	Name	X	Y	\
0	nan	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
0	2091.00	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
1	2092.00	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
2	2093.00	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	
3	2094.00	GD1017	ForschungsbohrungMünsterland1	32386176.36	5763283.15	

	Z	Altitude	Depth	formation	geometry
0	107.00	107.00	5956.00		NaN
0	27.00	107.00	5956.00	OberCampanium	POINT (32386176.36 5763283.15)
1	-193.00	107.00	5956.00	UnterCampanium	POINT (32386176.36 5763283.15)
2	-268.00	107.00	5956.00	OberSantonium	POINT (32386176.36 5763283.15)
3	-828.00	107.00	5956.00	MittelSantonium	POINT (32386176.36 5763283.15)

6.36.5 Creating Lines From Points

[8]:
```
lines = gg.visualization.create_lines_from_points(df=list_df[0])
lines
```

[8]:
```
PolyData (0x1f7615a2ee0)
  N Cells:  39
  N Points: 20
  X Bounds: 3.239e+07, 3.239e+07
  Y Bounds: 5.763e+06, 5.763e+06
  Z Bounds: -5.849e+03, 1.070e+02
  N Arrays: 0
```

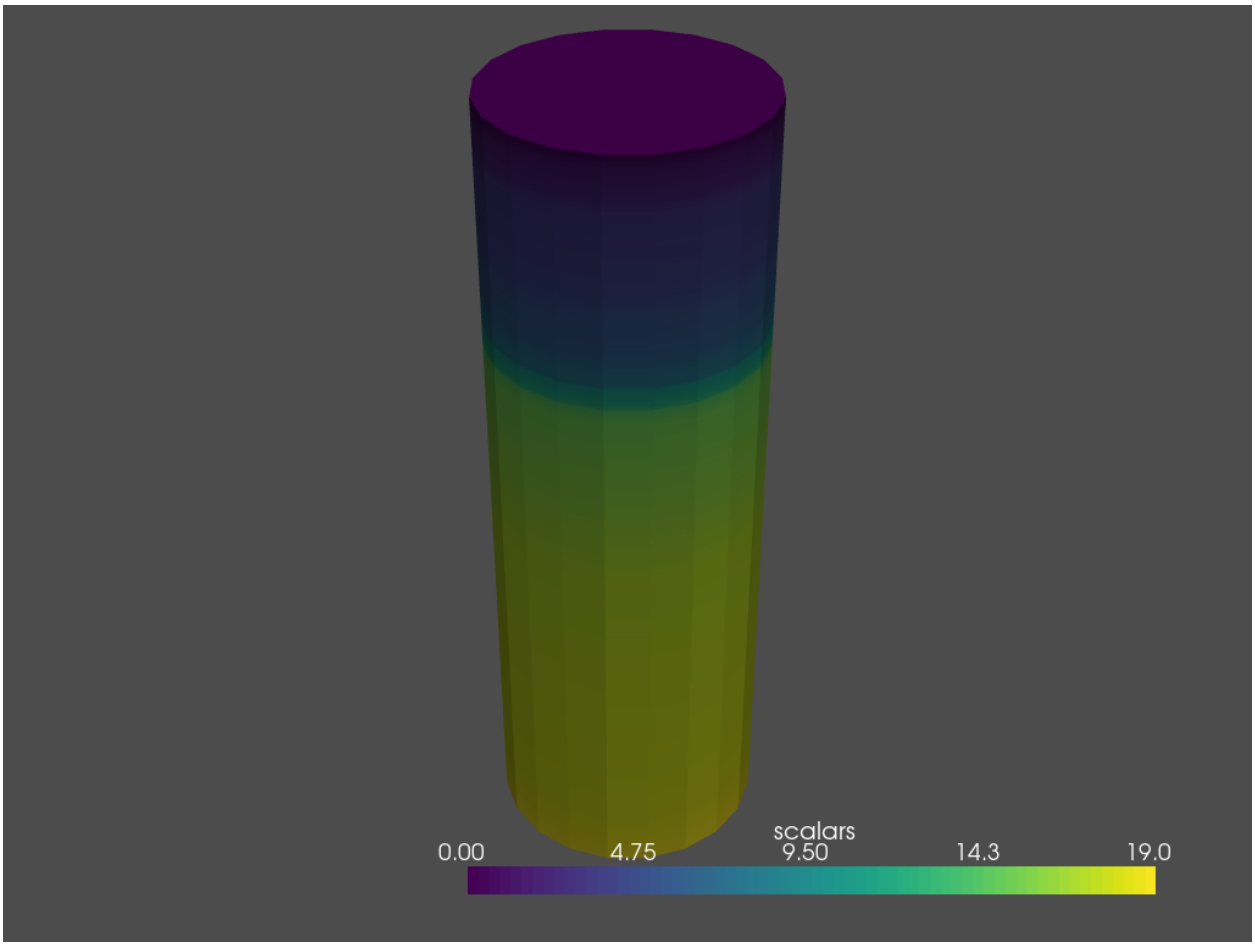
6.36.6 Creating Borehole Tubes

[9]:
```
tubes, df_groups = gg.visualization.create_borehole_tubes(df=data,
                                                          min_length= 10,
                                                          radius=1000)
tubes[0]
```

```
100%|| 2/2 [00:00<00:00, 153.78it/s]
```

[9]:
```
PolyData (0x1f7615db820)
  N Cells:  418
  N Points: 1520
  X Bounds: 3.239e+07, 3.239e+07
  Y Bounds: 5.762e+06, 5.764e+06
  Z Bounds: -5.849e+03, 1.070e+02
  N Arrays: 2
```

[10]:
```
tubes[0].plot()
```



[11]:
```
labels = gg.visualization.create_borehole_labels(df=data)
labels
```

[11]:
```
PolyData (0x1f7615dbfa0)
  N Cells:  2
  N Points: 2
  X Bounds: 3.239e+07, 3.240e+07
  Y Bounds: 5.753e+06, 5.763e+06
  Z Bounds: 6.000e+01, 1.070e+02
  N Arrays: 1
```

6.36.7 Creating Borehole Tube and Labels

[12]:
```
tubes, labels, df_groups = gg.visualization.create_boreholes_3d(df=data,
                                                                min_length=10,
                                                                color_dict=color_dict,
                                                                radius=1000)
tubes
```

```
100%|| 2/2 [00:00<00:00, 250.10it/s]
```

[12]:
```
MultiBlock (0x1f7616040a0)
  N Blocks: 2
  X Bounds: 32385176.360, 32404939.830
  Y Bounds: 5751889.550, 5764283.150
  Z Bounds: -5849.000, 107.000
```

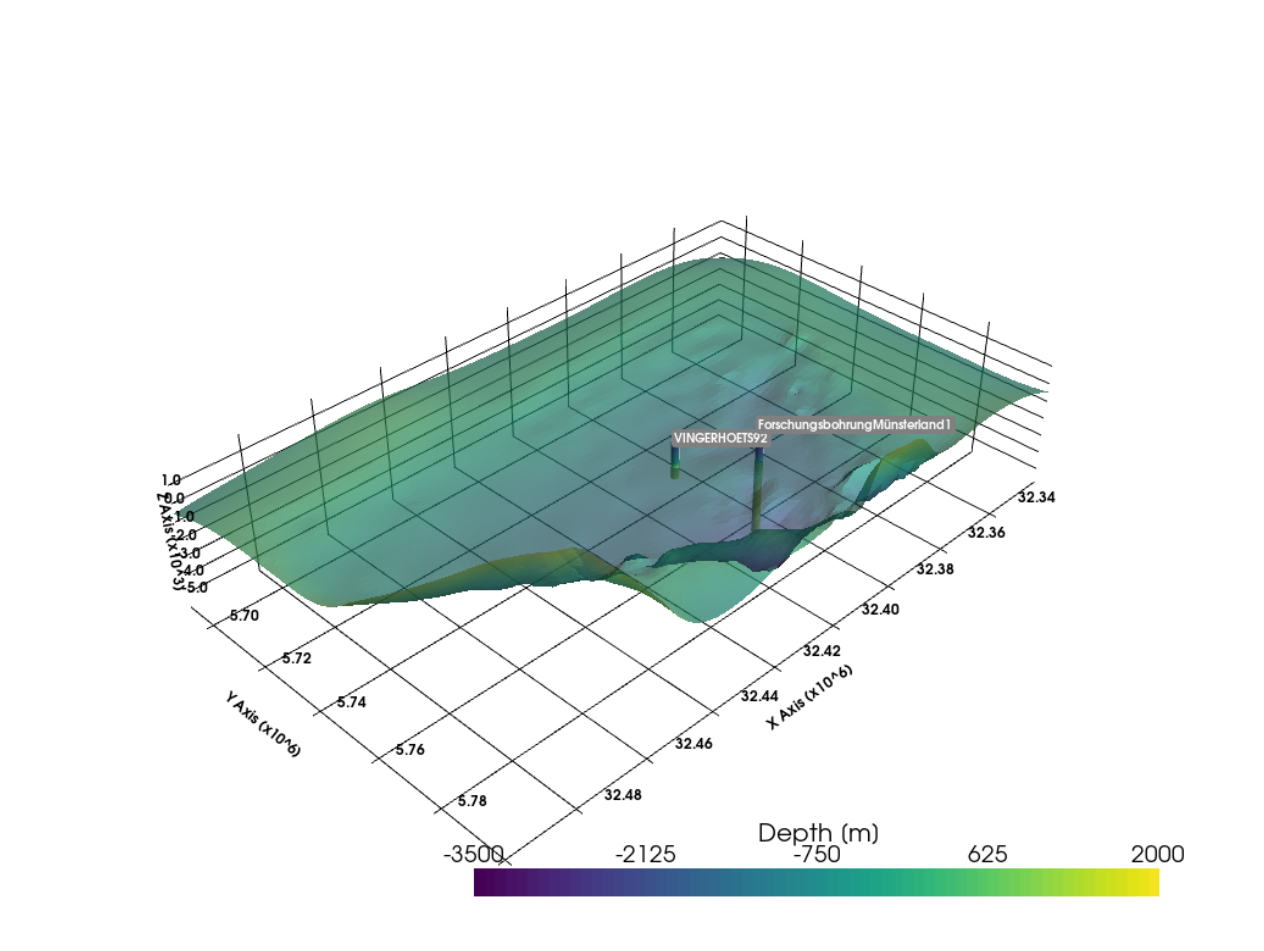
[13]:
```
tubes[0]
```

[13]:
```
PolyData (0x1f761604520)
  N Cells:  418
  N Points: 1520
  X Bounds: 3.239e+07, 3.239e+07
  Y Bounds: 5.762e+06, 5.764e+06
  Z Bounds: -5.849e+03, 1.070e+02
  N Arrays: 2
```

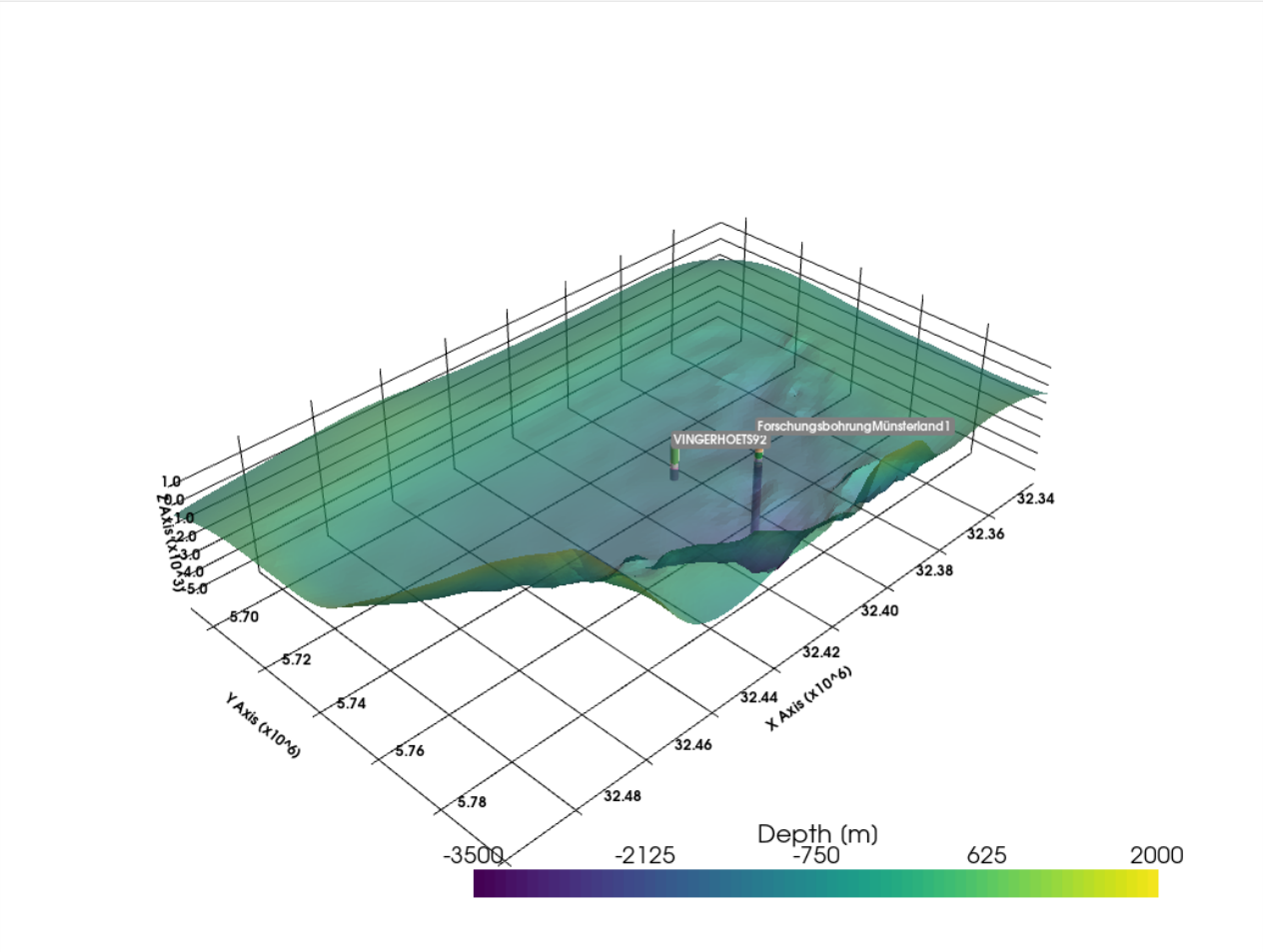
[14]:
```
labels
```

[14]:
```
PolyData (0x1f761604100)
  N Cells:  2
  N Points: 2
  X Bounds: 3.239e+07, 3.240e+07
  Y Bounds: 5.753e+06, 5.763e+06
  Z Bounds: 6.000e+01, 1.070e+02
  N Arrays: 1
```

[15]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh,scalar_bar_args=sargs, clim=[-3500, 2000], opacity=0.7)
p.add_point_labels(labels, "Labels", point_size=5, font_size=10)
p.add_mesh(tubes, show_scalar_bar=False)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,5)
p.show()
```



[16]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh,scalar_bar_args=sargs, clim=[-3500, 2000], opacity=0.7)
p.add_point_labels(labels, "Labels", point_size=5, font_size=10)

for j in range(len(tubes)):
        df_groups[j] = df_groups[j][1:]
        p.add_mesh(mesh=tubes[j], cmap=[color_dict[i] for i in df_groups[j]['formation'].
↪unique()], show_scalar_bar=False)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,5)
p.show()
```



6.37 36 Creating proj.crs.crs.CRS Objects for GemGIS

`proj.crs.crs.CRS` can be used in `GemGIS` to define the coordinate reference system (CRS) of `GeoDataFrames` or to specify the target CRS for coordinate transformations. The underlying `pyproj` package (https://pyproj4.github.io/pyproj/dev/index.html) is a Python interface to the `PROJ` package/library for cartographic projections and coordinate transformations (https://proj.org/).

6.37.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
file_path ='data/36_creating_proj_crs_object_for_gemgis/'
```

6.37.2 Creating CRS Objects

CRS objects/the CRS class can easily be initiated from a variety of inputs. The following shows a selection of these inputs. These objects can then be passed to functions that take a `proj.crs.crs.CRS` object as argument.

[2]:
```
from pyproj import CRS
```

From Authority

Making a CRS from an authority name and authority code.

[3]:
```
CRS.from_authority(auth_name='EPSG', code=4647)
```

[3]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

[4]:
```
CRS.from_authority(auth_name='EPSG', code='4647')
```

[4]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

From a dict

Making a CRS from a dictionary of PROJ parameters. The parameters were created using the `to_dict` function in the first place. You will likely lose important projection information when converting to a PROJ dict from another format!

[5]:
```
CRS.from_dict({'proj': 'tmerc',
               'lat_0': 0,
               'lon_0': 9,
               'k': 0.9996,
               'x_0': 32500000,
               'y_0': 0,
               'ellps': 'GRS80',
               'units': 'm',
               'no_defs': None,
               'type': 'crs'})
```

[5]:
```
<Projected CRS: +proj=tmerc +lat_0=0 +lon_0=9 +k=0.9996 +x_0=32500 ...>
Name: unknown
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- undefined
Coordinate Operation:
- name: unknown
- method: Transverse Mercator
Datum: Unknown based on GRS80 ellipsoid
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

From EPSG Code

Making a CRS from an EPSG code.

[6]:
```
CRS.from_epsg(4647)
```

[6]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

[7]:
```
CRS.from_epsg('4647')
```

[7]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

From User Input

Initialize a CRS class instance with: - PROJ string - Dictionary of PROJ parameters - PROJ keyword arguments for parameters - JSON string with PROJ parameters - CRS WKT string - An authority string [i.e. 'epsg:4326'] - An EPSG integer code [i.e. 4326] - A tuple of ("auth_name": "auth_code") [i.e ('epsg', '4326')] - An object with a `to_wkt` method. - A :class:`pyproj.crs.CRS` class

[8]:
```
CRS.from_user_input('EPSG:4647')
```

[8]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

[9]:
```
CRS.from_user_input(('EPSG', '4647'))
```

[9]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
```

(continues on next page)

(continued from previous page)

```
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

[10]:
```
CRS.from_user_input(4647)
```

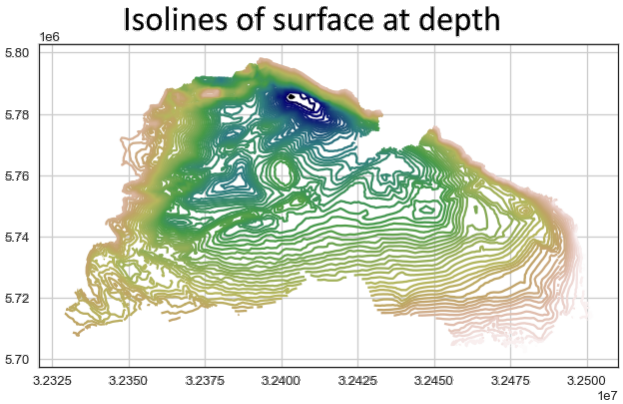
[10]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

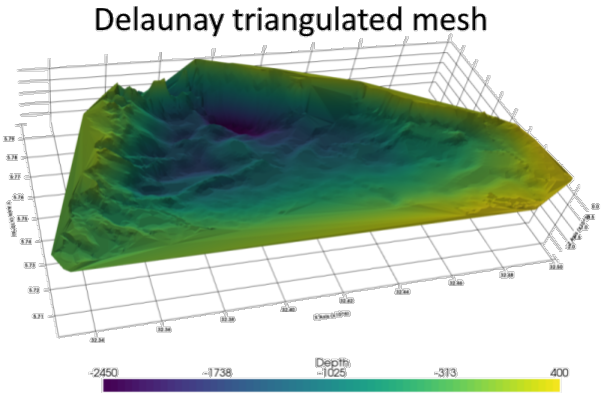
[11]:
```
crs = CRS.from_user_input('EPSG:4647')
type(crs)
```

[11]:
```
pyproj.crs.crs.CRS
```

6.38 37 Delaunay Triangulation for Isoline Maps

Isolines can be extracted from meshes to represent the depth of a surface in the subsurface. However, the original mesh may not be available for usage anymore. In that case, it may be useful to recreate a quick representation of the original mesh using Delaunay Triangulation.





6.38.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/37_delaunay_triangulation_for_isoline_maps/'
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\gemgis\gemgis.py:27:␣
↪UserWarning: Shapely 2.0 is installed, but because PyGEOS is also installed, GeoPandas␣
↪will still use PyGEOS by default for now. To force to use and test Shapely 2.0, you␣
↪have to set the environment variable USE_PYGEOS=0. You can do this before starting the␣
↪Python process, or in your code before importing geopandas:

import os
os.environ['USE_PYGEOS'] = '0'
import geopandas

In a future release, GeoPandas will switch to using Shapely by default. If you are using␣
↪PyGEOS directly (calling PyGEOS functions on geometries from GeoPandas), this will␣
↪then stop working and you are encouraged to migrate from PyGEOS to Shapely 2.0 (https:/
↪/shapely.readthedocs.io/en/latest/migration_pygeos.html).
  import geopandas as gpd
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="37_delaunay_triangulation_for_
↪isoline_maps.zip", dirpath=file_path)
```

```
Downloading file '37_delaunay_triangulation_for_isoline_maps.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F37_delaunay_triangulation_for_
↪isoline_maps.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\
↪data\37_delaunay_triangulation_for_isoline_maps'.
```

6.38.2 Loading Data

The data used for GemGIS is obtained from the *Geological Survey NRW*. It will be used under Datenlizenz Deutschland – Namensnennung – Version 2.0 (https://www.govdata.de/dl-de/by-2-0) with © Geowissenschaftliche Daten: Untergrundmodell NRW (2020).

[3]:
```
import geopandas as gpd

gdf = gpd.read_file(file_path + 'gg_kru_b_l_Z50m.shp')
gdf.head()
```

[3]:

	OBJECTID	Z	EINHEIT	Shape_Leng
0	1.00	-2450	gg_kru_b_l_Z50m	3924.67
1	2.00	-2400	gg_kru_b_l_Z50m	26332.90
2	3.00	-2350	gg_kru_b_l_Z50m	31104.28
3	4.00	-2300	gg_kru_b_l_Z50m	35631.73
4	5.00	-2250	gg_kru_b_l_Z50m	41702.52

(continues on next page)

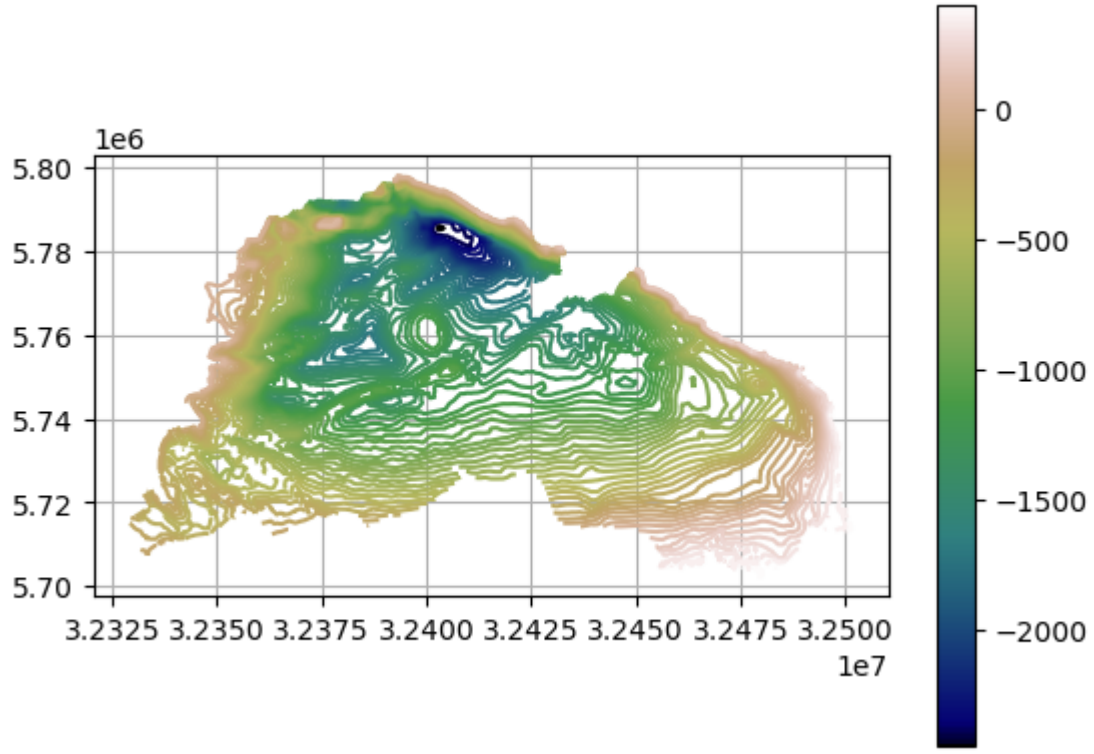
(continued from previous page)

```
                                              geometry
0  LINESTRING (32403313.109 5785053.637, 32402917...
1  LINESTRING (32410198.859 5781110.785, 32409807...
2  LINESTRING (32409587.930 5780538.824, 32408824...
3  LINESTRING (32408977.008 5779966.863, 32408808...
4  LINESTRING (32407319.922 5779788.672, 32407246...
```

6.38.3 Plotting the Data

The LineStrings of the GeoDataFrame can be plotted to illustrate the depth variations within the displayed basin.

[4]:
```
import matplotlib.pyplot as plt

gdf.plot(aspect='equal', column = 'Z', cmap = 'gist_earth', legend=True)
plt.grid()
```



6.38.4 Create Mesh using Delaunay Triangulation

A PyVista PolyData object can be created using `create_delaunay_mesh_from_gdf(..)`.

[5]:
```
mesh = gg.visualization.create_delaunay_mesh_from_gdf(gdf=gdf)
mesh
```

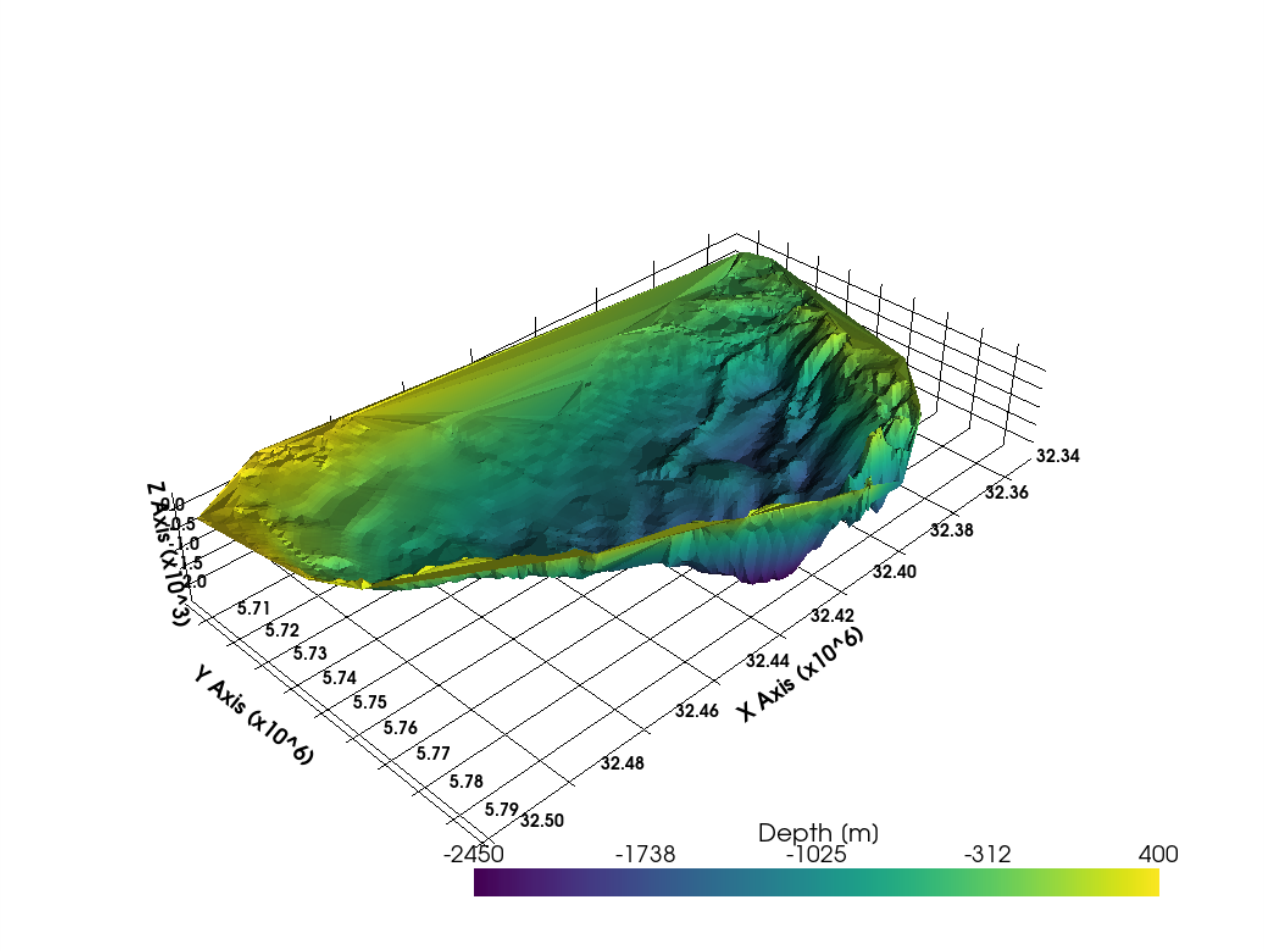
[5]:
```
PolyData (0x25ab3ceba00)
  N Cells:  45651
  N Points: 23009
  N Strips: 0
  X Bounds: 3.233e+07, 3.250e+07
  Y Bounds: 5.702e+06, 5.798e+06
  Z Bounds: -2.450e+03, 4.000e+02
  N Arrays: 1
```

6.38.5 Plotting the mesh

The created PolyData object can be plotted using PyVista.

[6]:
```
import pyvista as pv

sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh, scalars='Depth [m]', scalar_bar_args=sargs,)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:
↪ UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.38.6 Creating Mesh with PyVista

A triangulated surface can also be created using the built-in `delaunay_2d()` function of PyVista. See also https://docs.pyvista.org/examples/00-load/create-tri-surface.html.

[7]:
```
gdf_xyz = gg.vector.extract_xy(gdf=gdf)
gdf_xyz
```

[7]:

	OBJECTID	Z	EINHEIT	Shape_Leng	\
0	1.00	-2450.00	gg_kru_b_l_Z50m	3924.67	
1	1.00	-2450.00	gg_kru_b_l_Z50m	3924.67	
2	1.00	-2450.00	gg_kru_b_l_Z50m	3924.67	
3	1.00	-2450.00	gg_kru_b_l_Z50m	3924.67	
4	1.00	-2450.00	gg_kru_b_l_Z50m	3924.67	
...	...	...	...	...	
23004	733.00	400.00	gg_kru_b_l_Z50m	153.94	
23005	733.00	400.00	gg_kru_b_l_Z50m	153.94	
23006	733.00	400.00	gg_kru_b_l_Z50m	153.94	
23007	734.00	400.00	gg_kru_b_l_Z50m	94.02	
23008	734.00	400.00	gg_kru_b_l_Z50m	94.02	

	geometry	X	Y

(continues on next page)

(continued from previous page)

```
0      POINT (32403313.109 5785053.637) 32403313.11 5785053.64
1      POINT (32402917.820 5785157.270) 32402917.82 5785157.27
2      POINT (32402585.375 5785498.707) 32402585.38 5785498.71
3      POINT (32402870.727 5785657.359) 32402870.73 5785657.36
4      POINT (32403086.758 5786086.730) 32403086.76 5786086.73
...                                 ...         ...         ...
23004  POINT (32474651.758 5704841.910) 32474651.76 5704841.91
23005  POINT (32474603.586 5704848.586) 32474603.59 5704848.59
23006  POINT (32474597.918 5704823.645) 32474597.92 5704823.64
23007  POINT (32479877.676 5703950.059) 32479877.68 5703950.06
23008  POINT (32479970.945 5703938.168) 32479970.95 5703938.17

[23009 rows x 7 columns]
```

[8]:
```
import pyvista as pv
poly = pv.PolyData(gdf_xyz[['X', 'Y', 'Z']].values)
poly
```

[8]:
```
PolyData (0x25ab3caf1c0)
  N Cells:  23009
  N Points: 23009
  N Strips: 0
  X Bounds: 3.233e+07, 3.250e+07
  Y Bounds: 5.702e+06, 5.798e+06
  Z Bounds: -2.450e+03, 4.000e+02
  N Arrays: 0
```

[9]:
```
surf = poly.delaunay_2d()
surf
```

[9]:
```
PolyData (0x25ab3cae800)
  N Cells:  45536
  N Points: 23009
  N Strips: 0
  X Bounds: 3.233e+07, 3.250e+07
  Y Bounds: 5.702e+06, 5.798e+06
  Z Bounds: -2.450e+03, 4.000e+02
  N Arrays: 0
```

[10]:
```
surf ['Depth [m]'] = gdf_xyz['Z'].values
```

[11]:
```
surf.save(file_path + 'surf.vtk')
```

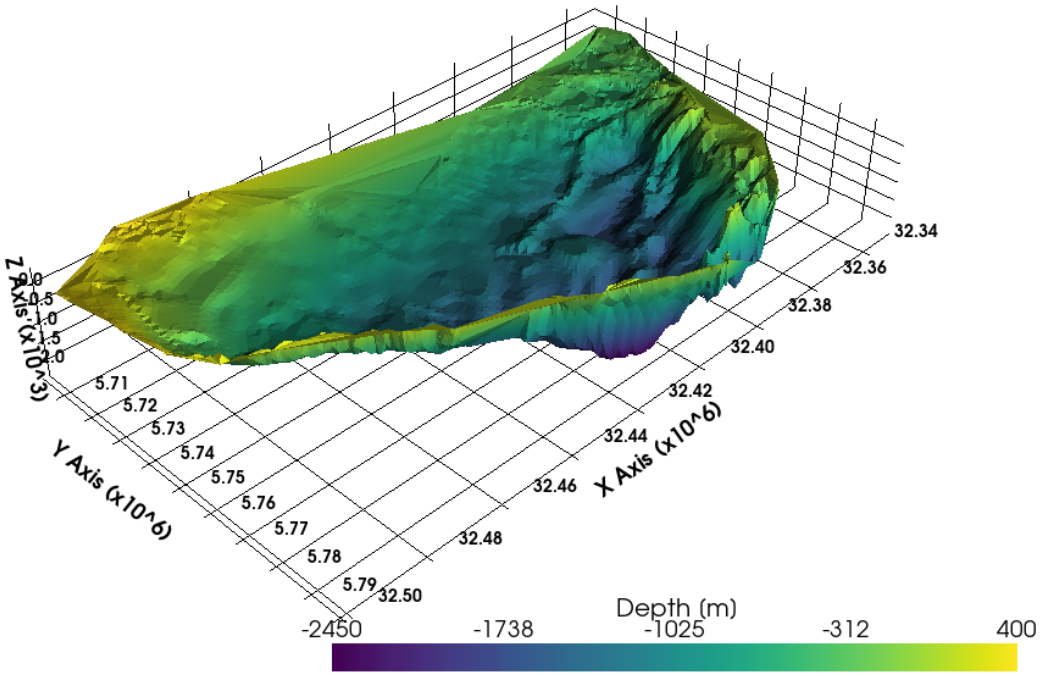
[12]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(surf, scalars='Depth [m]', scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
```

(continues on next page)

(continued from previous page)

```
p.set_scale(1,1,10)
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:
↪ UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



Creating Contour Lines

[13]:
```
import numpy as np
contours = surf.contour(isosurfaces=np.arange(-2450, 400,50))
contours
```

[13]:
```
PolyData (0x25ab3cadc00)
  N Cells:  36337
  N Points: 36178
  N Strips: 0
  X Bounds: 3.233e+07, 3.250e+07
```

(continues on next page)

(continued from previous page)

```
  Y Bounds: 5.704e+06, 5.798e+06
  Z Bounds: -2.400e+03, 3.500e+02
  N Arrays: 1
```

[14]:
```
from shapely.geometry import LineString

linestrings = []
number_of_previous_points=0

for i in range(contours.number_of_cells):

    index_to_find_length_of_line = i + number_of_previous_points

    number_of_points_of_line = contours.lines[index_to_find_length_of_line]

    values = [contours.lines[index_to_find_length_of_line+i+1] for i in range(number_of_
↪points_of_line)]

    vertices = [contours.points[value] for value in values]

    number_of_previous_points = number_of_previous_points + number_of_points_of_line

    linestrings.append(LineString(np.array(vertices)))


linestrings[:10]
```

[14]:
```
[<LINESTRING Z (32409587.93 5780538.824 -2350, 32410238.562 5780366.676 -2350)>,
 <LINESTRING Z (32407304.336 5777048.086 -2050, 32408334.289 5776648.801 -2050)>,
 <LINESTRING Z (32408748.977 5778005.047 -2200, 32409538.211 5777784.203 -2200)>,
 <LINESTRING Z (32403693.547 5786613.994 -2400, 32403174.859 5787003.531 -2400)>,
 <LINESTRING Z (32404738.664 5782672.48 -2350, 32405538.297 5782346.84 -2350)>,
 <LINESTRING Z (32404086.734 5783669.289 -2400, 32405073.039 5783625.348 -2400)>,
 <LINESTRING Z (32405876.141 5782105.867 -2350, 32406595.562 5781902.203 -2350)>,
 <LINESTRING Z (32403750.453 5783786.059 -2400, 32404086.734 5783669.289 -2400)>,
 <LINESTRING Z (32402643.008 5783912.356 -2400, 32403750.453 5783786.059 -2400)>,
 <LINESTRING Z (32402263.211 5784114.785 -2400, 32402643.008 5783912.356 -2400)>]
```

[15]:
```
gdf_countours = gpd.GeoDataFrame(geometry=linestrings)
gdf_countours['Z'] = [list(gdf_countours.loc[i].geometry.coords)[0][2] for i in␣
↪range(len(gdf_countours))]
gdf_countours
```

[15]:
```
                                              geometry        Z
0      LINESTRING Z (32409587.930 5780538.824 -2350.0... -2350.00
1      LINESTRING Z (32407304.336 5777048.086 -2050.0... -2050.00
2      LINESTRING Z (32408748.977 5778005.047 -2200.0... -2200.00
3      LINESTRING Z (32403693.547 5786613.994 -2400.0... -2400.00
4      LINESTRING Z (32404738.664 5782672.480 -2350.0... -2350.00
...                                                  ...       ...
36332  LINESTRING Z (32472712.875 5705828.297 350.000...   350.00
36333  LINESTRING Z (32476371.684 5706520.105 350.000...   350.00
```

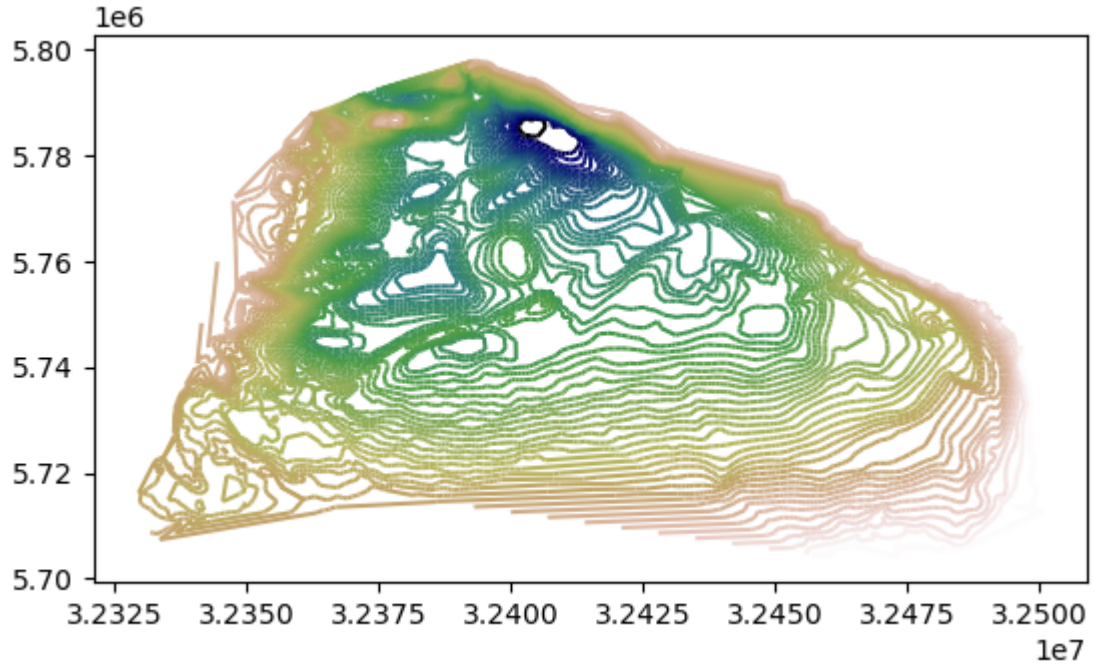
(continues on next page)

(continued from previous page)

```
36334  LINESTRING Z (32476542.145 5706398.895 350.000...   350.00
36335  LINESTRING Z (32476538.465 5706556.277 350.000...   350.00
36336  LINESTRING Z (32476540.793 5706593.047 350.000...   350.00

[36337 rows x 2 columns]
```

[16]:
```
gdf_countours.plot(column='Z', cmap='gist_earth')
```

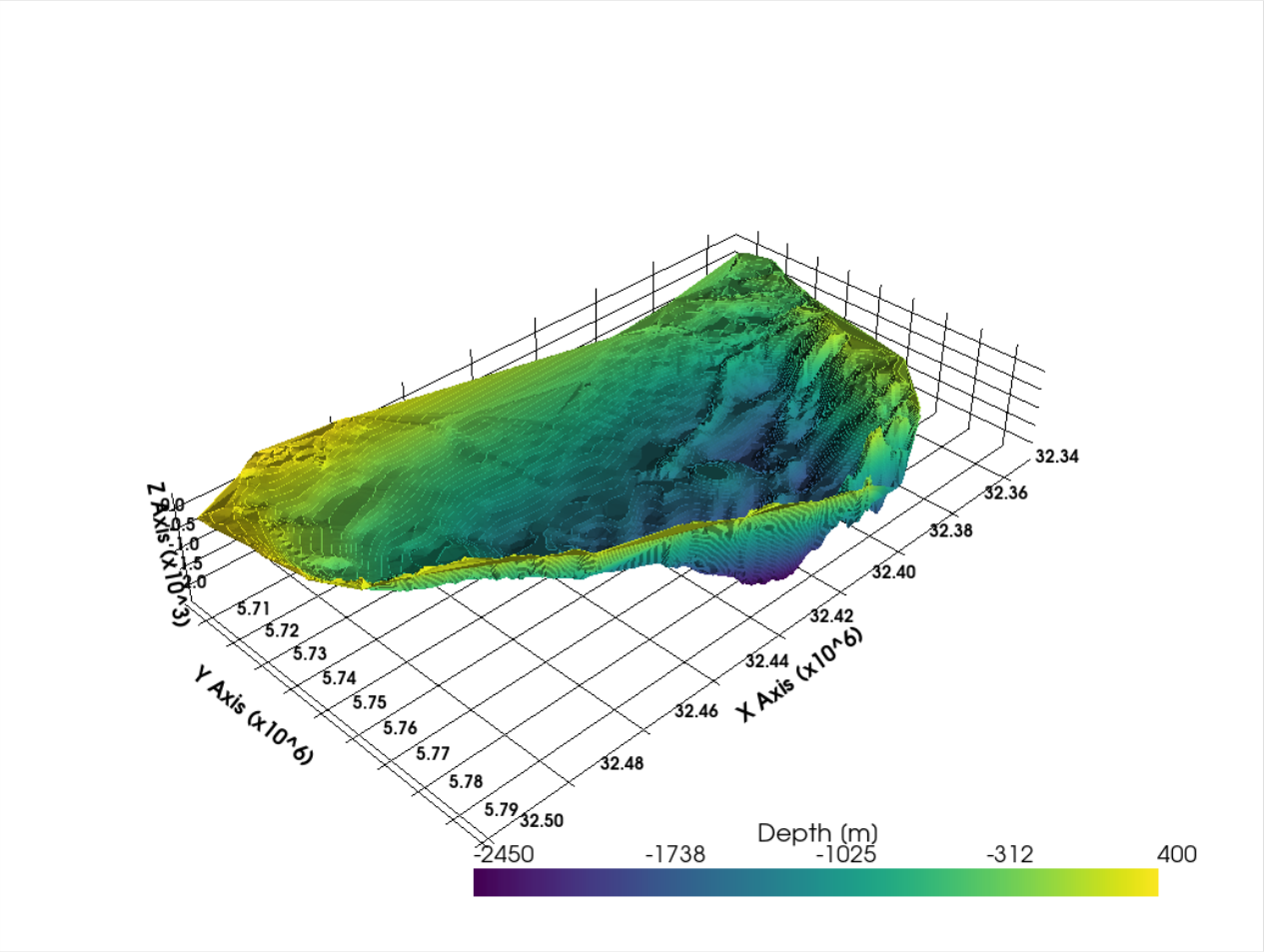
[16]:
```
<AxesSubplot: >
```



[17]:
```
contours.save(file_path + 'contours.vtk')
```

[18]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(surf, scalars='Depth [m]', scalar_bar_args=sargs)
p.add_mesh(contours, scalars='Depth [m]', scalar_bar_args=sargs)
p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:
↪ UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



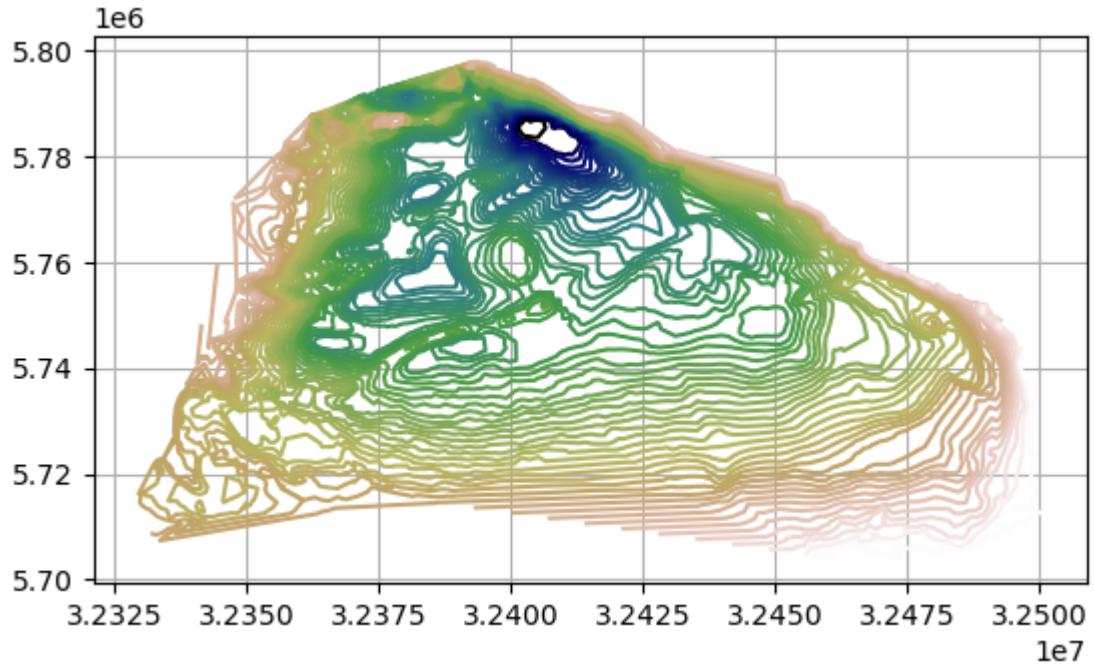
6.38.7 Merging Contour Lines

The single LineString elements can also be unified to larger LineStrings using `unify_linestrings(..)`.

[19]:
```
gdf = gg.vector.unify_linestrings(linestrings=gdf_countours)
gdf
```

[19]:
```
                                              geometry       Z
0    LINESTRING Z (32331825.641 5708789.973 -200.00... -200.00
1    LINESTRING Z (32334315.359 5723032.766 -250.00... -250.00
2    LINESTRING Z (32332516.312 5722028.768 -250.00... -250.00
3    LINESTRING Z (32332712.750 5721717.561 -250.00... -250.00
4    LINESTRING Z (32332516.312 5722028.768 -250.00... -250.00
..                                                 ...      ...
628  LINESTRING Z (32480957.211 5755586.262 200.000...  200.00
629  LINESTRING Z (32488784.838 5750893.873 250.000...  250.00
630  LINESTRING Z (32489046.086 5750945.794 300.000...  300.00
631  LINESTRING Z (32490632.693 5749614.501 250.000...  250.00
632  LINESTRING Z (32492043.488 5748252.048 300.000...  300.00

[633 rows x 2 columns]
```

[20]:
```
gdf.plot(column='Z', cmap='gist_earth')
plt.grid()
```



6.39 38 Interactive plotting with Bokeh in GemGIS

The bokeh package allows it to create interactive plots of the available geo data.

6.39.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='../../../../gemgis_data/data/38_interactive_plotting_with_bokeh_in gemgis/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
```

(continues on next page)

(continued from previous page)

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[3]:
```
gg.download_gemgis_data.download_tutorial_data(filename="38_interactive_plotting_with_
↪bokeh_in gemgis.zip", dirpath=file_path)
```

```
Downloading file '38_interactive_plotting_with_bokeh_in gemgis.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F38_interactive_plotting_with_bokeh_
↪in gemgis.zip' to 'C:\Users\ale93371\Documents\gemgis_data\data\38_interactive_
↪plotting_with_bokeh_in gemgis'.
```

6.39.2 Loading Libraries

[2]:
```
import geopandas as gpd
import rasterio
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.39.3 Importing Bokeh

Bokeh is being imported and the output of the plots is set to be in the notebooks.

[3]:
```
from bokeh.plotting import figure, show
from bokeh.io import output_notebook, show
output_notebook()
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_load.v0+json

6.39.4 Plotting OSM Data

The first data to be integrated is a Open Street Base Map. It can be downloaded by accessing the tile provider `OSM` via `get_provider(..)`.

NB: When using a tile provider, all other data has to be reprojected to ```'EPSG:3857'``` (https://wiki.openstreetmap.org/wiki/EPSG:3857)

[4]:
```
from bokeh.tile_providers import OSM, get_provider
```

(continues on next page)

(continued from previous page)

```
tile_provider = get_provider(OSM)
tile_provider
```

```
[4]: WMTSTileSource(id='1002', ...)
```

In order to display a plot, a figure needs to be created using `figure(..)` and the provided arguments. The range of the map was limited to the rough extent of the Münsterland Basin in Northern Germany.

```
[5]: p = figure(title="Test",
            x_axis_label='X [m]',
            y_axis_label='Y [m]',
            match_aspect=True,
            plot_width=800,
            plot_height=600,
            x_range=(7.5e5, 10.5e5),
            y_range=(6.65e6, 6.90e6))
```

The OSM Map is now added to the figure using `p.add_tile(..)`.

```
[6]: p.add_tile(tile_provider)
```

```
[6]: TileRenderer(id='1038', ...)
```

Showing the plot

The plot is shown using `show(p)`.

```
[7]: show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

6.39.5 Plotting Raster Data

Loading Data

The data used for GemGIS is obtained from OpenDataNRW. It will be used under Datenlizenz Deutschland – Namensnennung – Version 2.0 (https://www.govdata.de/dl-de/by-2-0) with © Geowissenschaftliche Daten: Digitales Höhenmodell NRW 1m (2020).

```
[8]: data = rasterio.open(file_path + 'DEM50_reproj.tif')
```

The CRS of the raster was reprojected using `gg.raster.reproject_raster(..)` to match the CRS of the Open Street Map layer.

```
[9]: data.crs
```

```
[9]: CRS.from_epsg(3857)
```

A second figure was created.

[10]:
```
p = figure(title="Test",
           x_axis_label='X [m]',
           y_axis_label='Y [m]',
           match_aspect=True,
           plot_width=800,
           plot_height=600,
           x_range=(7.5e5, 10.5e5),
           y_range=(6.65e6, 6.90e6))
```

The OSM Layer will be added as well.

[11]:
```
p.add_tile(tile_provider)
```

[11]:
```
TileRenderer(id='1111', ...)
```

Create GlypRenderer for Raster

For a better representation of the raster, the nodata value of the raster will be replaced with the min value of the array/band.

[12]:
```
data.nodata
```

[12]:
```
9999.0
```

[13]:
```
data_cleaned = data.read(1)

data_cleaned[data_cleaned == data.nodata] = data.read(1).min()
```

An image will be added to the figure using the dimensions of the rasterio object.

[14]:
```
from bokeh.models.mappers import ContinuousColorMapper
import numpy as np

dem_renderer = p.image(image = [np.flipud(data_cleaned)],
                       x=data.bounds[0],
                       y=data.bounds[1],
                       dw=data.bounds[2]-data.bounds[0],
                       dh=data.bounds[3]-data.bounds[1],
                       legend_label="DEM",
                       palette = 'RdYlGn11'
                      )
```

Create Hover Tool for Bokeh Plot

A HoverTool is created so that the height of the DEM will be returned at the respective cursor position when moving across the raster.

[15]:
```
from bokeh.models import HoverTool
dem_tool = HoverTool(tooltips=[('Height', '@image{0.00} m')],
                     renderers=[dem_renderer])
```

Showing the plot

The plot can now be displayed. In addition, the HoverTool and and the option to hide the data is being added.

[16]:
```
p.add_tools(dem_tool)
p.legend.click_policy="hide"
show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

6.39.6 Plotting Vector Data - LineString Data

Loading data

Vector data can also be added to a Bokeh plot.

[17]:
```
faults = gpd.read_file(file_path + 'gg_nrw_geotekst_l.shp')
faults.head()
```

[17]:
```
   OBJECTID_1  OBJECTID_2                                      Layer_Name  \
0        1.00           1                    Störung001_Swist-Sprung_SW
1        2.00           2               Störung002_Müggenhausener Sprung_SW
2        3.00           3                 Störung003_Strassfelder Sprung_SW
3        4.00           4                 Störung003_Strassfelder Sprung_SW
4        5.00           5  Störung004_Ludendorfer Sprung_Meckenheimer Sprung

                         Quelle                                ST_NAME  \
0  Erft Scholle RWE Projekt 2015                        Swist-Sprung
1  Erft Scholle RWE Projekt 2015                   Müggenhausener Sprung
2  Erft Scholle RWE Projekt 2015                     Strassfelder Sprung
3  Erft Scholle RWE Projekt 2015                     Strassfelder Sprung
4  Erft Scholle RWE Projekt 2015  Ludendorfer Sprung_Meckenheimer Sprung

   OBJECTID  Id ssymbol_QB  ST_NR  ST_ART_ID  ... DIP_3D  \
0         0   0       None   0.00          0  ...  62.50
1         0   0       None   0.00          0  ...  62.50
2         0   0       None   0.00          0  ...  62.50
3         0   0       None   0.00          0  ...  62.50
4         0   0       None   0.00          0  ...  62.50

               STOE_UEB_3  STOE_TOP_H                     STOE_BASIS  \
0  überregionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
1      regionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
2      regionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
3         lokale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
4      regionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t

   ST_ART_3D  Shape_Le_2  Versatz_TB  Doku_Versa Shape_Le_3  \
0       None    37635.89        0.00        None   37635.89
1       None    21797.50        0.00        None   21797.50
```

(continues on next page)

(continued from previous page)

```
2       None    23928.16        0.00        None   23928.16
3       None       56.58        0.00        None      56.58
4       None    18578.06        0.00        None   18578.06

                                           geometry
0  LINESTRING (32361016.617 5608960.850, 32360902...
1  LINESTRING (32358412.934 5611983.080, 32358293...
2  LINESTRING (32360075.838 5610943.615, 32359971...
3  LINESTRING (32341010.296 5623832.070, 32340983...
4  LINESTRING (32360936.158 5609018.641, 32360846...

[5 rows x 47 columns]
```

Empty and invalid entries are removed.

[18]:
```
faults = faults[~faults.is_empty]
faults = faults[faults.is_valid]
```

The data is reprojected to match the CRS of the OSM Layer.

[19]:
```
faults_reproj = faults.to_crs(epsg='3857')
faults_reproj.head()
```

[19]:
```
   OBJECTID_1  OBJECTID_2                                        Layer_Name  \
0        1.00           1                      Störung001_Swist-Sprung_SW
1        2.00           2                Störung002_Müggenhausener Sprung_SW
2        3.00           3                  Störung003_Strassfelder Sprung_SW
3        4.00           4                  Störung003_Strassfelder Sprung_SW
4        5.00           5  Störung004_Ludendorfer Sprung_Meckenheimer Sprung

                         Quelle                                ST_NAME  \
0  Erft Scholle RWE Projekt 2015                          Swist-Sprung
1  Erft Scholle RWE Projekt 2015                 Müggenhausener Sprung
2  Erft Scholle RWE Projekt 2015                   Strassfelder Sprung
3  Erft Scholle RWE Projekt 2015                   Strassfelder Sprung
4  Erft Scholle RWE Projekt 2015  Ludendorfer Sprung_Meckenheimer Sprung

   OBJECTID  Id ssymbol_QB  ST_NR  ST_ART_ID  ... DIP_3D  \
0         0   0       None   0.00          0  ...  62.50
1         0   0       None   0.00          0  ...  62.50
2         0   0       None   0.00          0  ...  62.50
3         0   0       None   0.00          0  ...  62.50
4         0   0       None   0.00          0  ...  62.50

               STOE_UEB_3  STOE_TOP_H                     STOE_BASIS  \
0  überregionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
1      regionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
2      regionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
3         lokale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t
4      regionale Bedeutung         DGM  Tertiaer_b/Praeperm_t/Karbon_t

   ST_ART_3D  Shape_Le_2  Versatz_TB  Doku_Versa Shape_Le_3  \
0       None    37635.89        0.00        None   37635.89
```

(continues on next page)

(continued from previous page)

```
1       None    21797.50        0.00        None   21797.50
2       None    23928.16        0.00        None   23928.16
3       None       56.58        0.00        None      56.58
4       None    18578.06        0.00        None   18578.06

                                           geometry
0  LINESTRING (783179.405 6553639.429, 782995.936...
1  LINESTRING (778956.585 6558296.279, 778762.946...
2  LINESTRING (781616.721 6556726.982, 781451.743...
3  LINESTRING (751011.105 6576218.560, 750967.199...
4  LINESTRING (783050.468 6553727.191, 782907.509...

[5 rows x 47 columns]
```

The data is now being exploded to single LineStrings and the X and Y coordinates are being extracted.

[20]:
```
faults_reproj = gg.vector.explode_multilinestrings(gdf=faults_reproj)
faults_reproj = gg.vector.extract_xy_linestring(gdf=faults_reproj)
```

Creating ColumnDataSource

The data is being converted to a ColumnDataSource. The geometry column of the GeoDataFrame must be dropped.

[33]:
```
faults_reproj = faults_reproj[['Shape_Le_2', 'X', 'Y', 'geometry']]
faults_reproj.head()
```

[33]:
```
   Shape_Le_2                                                  X  \
0    37635.89  [783179.4051572308, 782995.935648582, 782902.3...
1    21797.50  [778956.5848617974, 778762.9461795501, 778554...
2    23928.16  [781616.7214601886, 781451.7425967428, 781293...
3       56.58  [751011.1047211702, 750967.1987568956, 750960...
4    18578.06  [783050.4681690165, 782907.5090723939, 782818...

                                                    Y  \
0  [6553639.429076611, 6553792.618358821, 6553877...
1  [6558296.2788241105, 6558496.939616477, 655871...
2  [6556726.982401831, 6556744.251795183, 6556763...
3  [6576218.559684751, 6576283.0530930245, 657629...
4  [6553727.191026189, 6553824.610592816, 6553889...

                                           geometry
0  LINESTRING (783179.405 6553639.429, 782995.936...
1  LINESTRING (778956.585 6558296.279, 778762.946...
2  LINESTRING (781616.721 6556726.982, 781451.743...
3  LINESTRING (751011.105 6576218.560, 750967.199...
4  LINESTRING (783050.468 6553727.191, 782907.509...
```

[34]:
```
from bokeh.models import ColumnDataSource
geo_data = ColumnDataSource(faults_reproj.drop('geometry', axis=1).sample(n=10))
geo_data
```

[34]:
```
ColumnDataSource(id='1522', ...)
```

[35]:
```
p = figure(title="Test",
           x_axis_label='X [m]',
           y_axis_label='Y [m]',
           match_aspect=True,
           plot_width=800,
           plot_height=600,
           x_range=(7.5e5, 10.5e5),
           y_range=(6.65e6, 6.90e6))
```

[36]:
```
p.add_tile(tile_provider)
```

[36]:
```
TileRenderer(id='1560', ...)
```

The data will now be added as `multi_line` to the the figure.

[37]:
```
faults_renderer = p.multi_line(xs='X',
                               ys='Y',
                               source=geo_data,
                               color='red',
                               line_width=3,
                               legend_label="Faults",
                               hover_line_alpha=1.0)
faults_renderer
```

[37]:
```
GlyphRenderer(id='1568', ...)
```

[38]:
```
fault_hover_tool = HoverTool(tooltips=[('X_value', '$xs')],
                             renderers=[faults_renderer],
                             line_policy='nearest',
                             show_arrow=False,
                             mode='vline'
    )
#('FaultName', '@ST_NAME')
# ('LayerName', '@Layer_Name'),
# ('Dip', '@DIP_3D'),
# ('Length', '@Shape_Le_2')
```

Showing the plot

The plot can now be displayed. In addition, the HoverTool and and the option to hide the data is being added.

[39]:
```
p.add_tools(fault_hover_tool)
#p.add_tools(dem_tool)
p.legend.click_policy="hide"
show(p)
```

Data type cannot be displayed: application/javascript, application/vnd.bokehjs_exec.v0+json

6.40 39 Working with Shapely Base Geometries containing Z components

Shapely Base Geometries can not only have an X and Y coordinate but also a Z component. The following introduces how GemGIS is handling Shapely Base Geometries with an additional Z component.

6.40.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/39_shapely_base_geometries_with_z_component/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.40.2 Point Data

Points can easily be defined using an X, Y and Z coordinate.

[2]:
```
import gemgis as gg
from shapely.geometry import Point

point = Point(1,2,3)
point
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:

[3]:
```
point.wkt
```

[3]:
```
'POINT Z (1 2 3)'
```

Despite the Z component, the point is still a regular Shapely Point.

[4]:
```
type(point)
```

[4]:
```
shapely.geometry.point.Point
```

But the attribute that an object has a z component is now set to `True`.

[5]:
```
point.has_z
```

[5]:
```
True
```

Accessing the coordinates

The coordinates can be accessed via attributes.

[6]:
```
point.x
```

[6]:
```
1.0
```

[7]:
```
point.y
```

[7]:
```
2.0
```

[8]:
```
point.z
```

[8]:
```
3.0
```

Creating a GeoDataFrame and extracting coordinates

[9]:
```
import geopandas as gpd

gdf = gpd.GeoDataFrame(geometry=[point, point])
gdf
```

[9]:
```
                           geometry
0  POINT Z (1.00000 2.00000 3.00000)
1  POINT Z (1.00000 2.00000 3.00000)
```

Extracting Coordinates

[10]:
```
gdf_xyz = gg.vector.extract_xyz_points(gdf=gdf)
gdf_xyz
```

[10]:
```
                            geometry    X    Y    Z
0  POINT Z (1.00000 2.00000 3.00000) 1.00 2.00 3.00
1  POINT Z (1.00000 2.00000 3.00000) 1.00 2.00 3.00
```

[11]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf)
gdf_xyz
```

[11]:
```
                            geometry    X    Y    Z
0  POINT Z (1.00000 2.00000 3.00000) 1.00 2.00 3.00
1  POINT Z (1.00000 2.00000 3.00000) 1.00 2.00 3.00
```

6.40.3 LineString Data

[12]:
```
import gemgis as gg
from shapely.geometry import LineString

linestring = LineString(([1,2,3], [4,5,6]))
linestring
```

[12]:

[13]:
```
linestring.wkt
```

[13]:
```
'LINESTRING Z (1 2 3, 4 5 6)'
```

Despite the Z component, the line is still a regular Shapely LineString.

[14]:
```
type(linestring)
```

[14]:
```
shapely.geometry.linestring.LineString
```

But the attribute that an object has a z component is now set to `True`.

[15]:
```
linestring.has_z
```

[15]:
```
True
```

Accessing the coordinates

The coordinates can be accessed via attributes.

[16]:
```
list(linestring.coords)
```

[16]:
```
[(1.0, 2.0, 3.0), (4.0, 5.0, 6.0)]
```

Creating a GeoDataFrame and extracting coordinates

[17]:
```
import geopandas as gpd

gdf = gpd.GeoDataFrame(geometry=[linestring, linestring])
gdf
```

[17]:
```
                                             geometry
0  LINESTRING Z (1.00000 2.00000 3.00000, 4.00000...
1  LINESTRING Z (1.00000 2.00000 3.00000, 4.00000...
```

Extracting Coordinates

`extract_xy_linestrings(..)` will also work for extracting the xyz coordinates as the used Pandas `explode(..)` function also extracts the Z component of the LineString object. In the subsequent definition of the columns for the coordinates, an additional Z column will be created if a Value error would have been returned before that.

[18]:
```
gdf_xyz = gg.vector.extract_xy_linestrings(gdf=gdf)
gdf_xyz
```

[18]:
```
                  geometry    X    Y    Z
0  POINT (1.00000 2.00000) 1.00 2.00 3.00
1  POINT (4.00000 5.00000) 4.00 5.00 6.00
2  POINT (1.00000 2.00000) 1.00 2.00 3.00
3  POINT (4.00000 5.00000) 4.00 5.00 6.00
```

However, a dedicated function to extract the X, Y and Z coordinates from a GeoDataFrame containing LineStrings with a Z component is also implemented in GemGIS -> `extract_xyz_linestrings(...)`.

[19]:
```
gdf_xyz = gg.vector.extract_xyz_linestrings(gdf=gdf, reset_index=True)
gdf_xyz
```

[19]:
```
                  geometry           points    X    Y    Z
0  POINT (1.00000 2.00000)  (1.0, 2.0, 3.0) 1.00 2.00 3.00
1  POINT (4.00000 5.00000)  (4.0, 5.0, 6.0) 4.00 5.00 6.00
2  POINT (1.00000 2.00000)  (1.0, 2.0, 3.0) 1.00 2.00 3.00
3  POINT (4.00000 5.00000)  (4.0, 5.0, 6.0) 4.00 5.00 6.00
```

The coordinates can also be extracted using the common function `extract_xyz(...)`.

[20]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf)
gdf_xyz
```

[20]:
```
                  geometry    X    Y    Z
0  POINT (1.00000 2.00000) 1.00 2.00 3.00
1  POINT (4.00000 5.00000) 4.00 5.00 6.00
2  POINT (1.00000 2.00000) 1.00 2.00 3.00
3  POINT (4.00000 5.00000) 4.00 5.00 6.00
```

6.40.4 Polygon Data

[21]:
```
import gemgis as gg
from shapely.geometry import Polygon

polygon = Polygon([[0, 0, 0], [1, 0, 0], [1, 1, 0], [0, 1, 0], [0, 0, 0]])
polygon
```

[21]:

[22]:
```
polygon.wkt
```

[22]:
```
'POLYGON Z ((0 0 0, 1 0 0, 1 1 0, 0 1 0, 0 0 0))'
```

Despite the Z component, the line is still a regular Shapely Polygon.

[23]:
```
type(polygon)
```

[23]:
```
shapely.geometry.polygon.Polygon
```

But the attribute that an object has a z component is now set to `True`.

[24]:
```
polygon.has_z
```

[24]:
```
True
```

Accessing the coordinates

The coordinates can be accessed via attributes.

[25]:
```
list(polygon.exterior.coords)
```

[25]:
```
[(0.0, 0.0, 0.0),
 (1.0, 0.0, 0.0),
 (1.0, 1.0, 0.0),
 (0.0, 1.0, 0.0),
 (0.0, 0.0, 0.0)]
```

Creating a GeoDataFrame and extracting coordinates

[26]:
```
import geopandas as gpd

gdf = gpd.GeoDataFrame(geometry=[polygon, polygon])
gdf
```

[26]:
```
                                            geometry
0  POLYGON Z ((0.00000 0.00000 0.00000, 1.00000 0...
1  POLYGON Z ((0.00000 0.00000 0.00000, 1.00000 0...
```

6.40.5 Extracting Coordinates

The coordinates can be extracted easily with `extract_xyz(...)`. As the polygons are exploded into LineStrings during the process of extracting the coordinates, the functionality to extract coordinates from Polygons is already present.

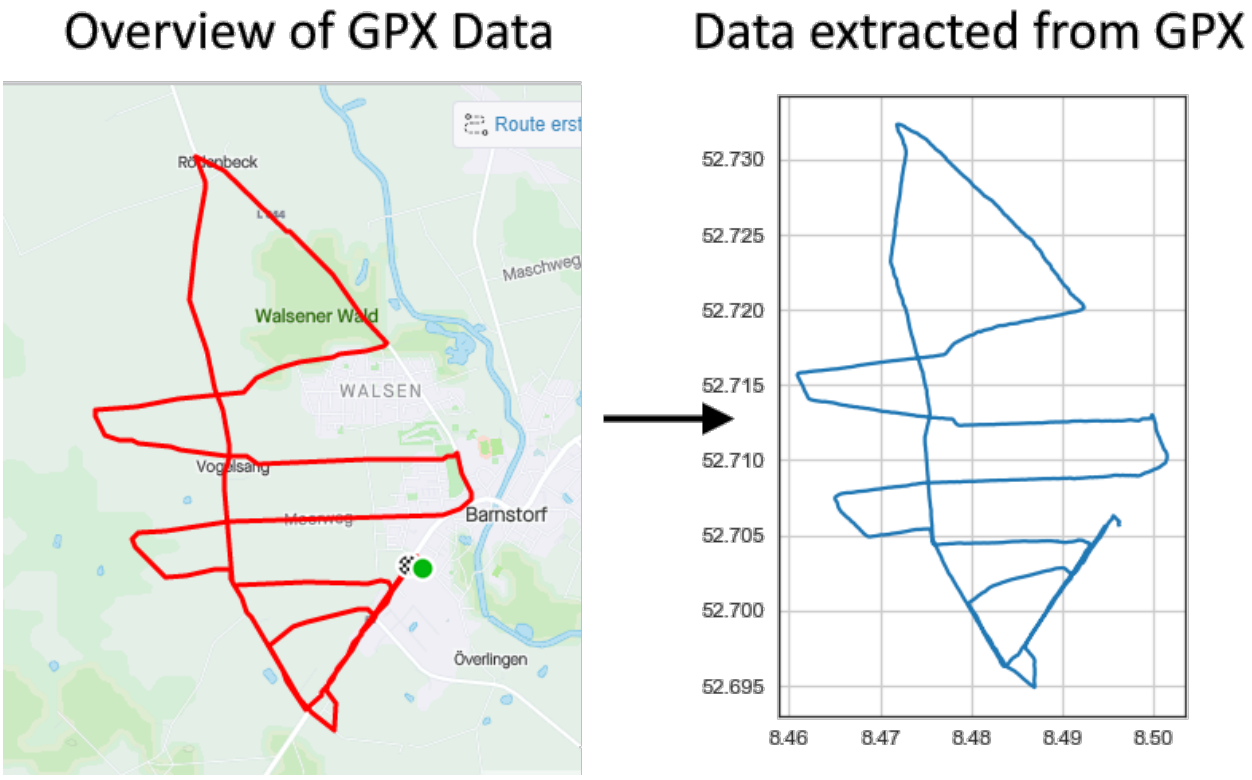
[27]:
```
gdf_xyz = gg.vector.extract_xyz(gdf=gdf)
gdf_xyz
```

[27]:

	geometry	X	Y	Z
0	POINT (0.00000 0.00000)	0.00	0.00	0.00
1	POINT (1.00000 0.00000)	1.00	0.00	0.00
2	POINT (1.00000 1.00000)	1.00	1.00	0.00
3	POINT (0.00000 1.00000)	0.00	1.00	0.00
4	POINT (0.00000 0.00000)	0.00	0.00	0.00
5	POINT (0.00000 0.00000)	0.00	0.00	0.00
6	POINT (1.00000 0.00000)	1.00	0.00	0.00
7	POINT (1.00000 1.00000)	1.00	1.00	0.00
8	POINT (0.00000 1.00000)	0.00	1.00	0.00
9	POINT (0.00000 0.00000)	0.00	0.00	0.00

6.41 40 Working with GPX Data in GemGIS

GPX, or GPS Exchange Format, is an XML schema designed as a common GPS data format for software applications. It can be used to describe waypoints, tracks, and routes. The format is open and can be used without the need to pay license fees. Location data (and optionally elevation, time, and other information) is stored in tags and can be interchanged between GPS devices and software. Common software applications for the data include viewing tracks projected onto various map sources, annotating maps, and geotagging photographs based on the time they were taken.



Source: https://en.wikipedia.org/wiki/GPS_Exchange_Format

6.41.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/40_working_with_gpx_data_in_gemgis/'
```

```
C:\Users\ale93371\Anaconda3\envs\gemgis\lib\site-packages\gemgis\gemgis.py:27:␣
↪UserWarning: Shapely 2.0 is installed, but because PyGEOS is also installed, GeoPandas␣
↪will still use PyGEOS by default for now. To force to use and test Shapely 2.0, you␣
↪have to set the environment variable USE_PYGEOS=0. You can do this before starting the␣
↪Python process, or in your code before importing geopandas:

import os
os.environ['USE_PYGEOS'] = '0'
import geopandas

In a future release, GeoPandas will switch to using Shapely by default. If you are using␣
↪PyGEOS directly (calling PyGEOS functions on geometries from GeoPandas), this will␣
↪then stop working and you are encouraged to migrate from PyGEOS to Shapely 2.0 (https:/
↪/shapely.readthedocs.io/en/latest/migration_pygeos.html).
```

(continues on next page)

(continued from previous page)

```
  import geopandas as gpd
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="40_working_with_gpx_data_in_
↪gemgis.zip", dirpath=file_path)
```

6.41.2 Load Data

Data from a running practice in northern Germany is used for demonstration purposes.

[3]:
```
import geopandas as gpd

gpx = gg.vector.load_gpx(path=file_path+'Run.gpx', layer='tracks')
gpx
```

[3]:
```
<open Collection 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\data\
↪40_working_with_gpx_data_in_gemgis\Run.gpx:tracks', mode 'r' at 0x2127ee838b0>
```

6.41.3 Inspecting the data

The driver used to open the data was `GPX`

[4]:
```
gpx.driver
```

[4]:
```
'GPX'
```

The CRS of the data is `EPGS:4326`.

[5]:
```
gpx.crs
```

[5]:
```
{'init': 'epsg:4326'}
```

[6]:
```
gpx.crs_wkt
```

[6]:
```
'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,AUTHORITY["EPSG
↪","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
↪UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS[
↪"Longitude",EAST],AUTHORITY["EPSG","4326"]]'
```

The extent of the data is defining the bounds of the gpx.

[7]:
```
gpx.bounds
```

[7]:
```
(8.460906, 52.694879, 8.501507, 52.732331)
```

Since the track has different start and end points, the track is not closed.

[8]:
```
gpx.closed
```

[8]:
```
False
```

Accessing the meta data.

[9]:
```
gpx.meta
```

[9]:
```
{'driver': 'GPX',
 'schema': {'properties': OrderedDict([('name', 'str'),
              ('cmt', 'str'),
              ('desc', 'str'),
              ('src', 'str'),
              ('link1_href', 'str'),
              ('link1_text', 'str'),
              ('link1_type', 'str'),
              ('link2_href', 'str'),
              ('link2_text', 'str'),
              ('link2_type', 'str'),
              ('number', 'int'),
              ('type', 'str')]),
  'geometry': 'MultiLineString'},
 'crs': {'init': 'epsg:4326'},
 'crs_wkt': 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY[
↪"EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS[
↪"Latitude",NORTH],AXIS["Longitude",EAST],AUTHORITY["EPSG","4326"]]'}
```

Name of the Track.

[10]:
```
gpx.name
```

[10]:
```
'tracks'
```

[11]:
```
gpx.profile
```

[11]:
```
{'driver': 'GPX',
 'schema': {'properties': OrderedDict([('name', 'str'),
              ('cmt', 'str'),
              ('desc', 'str'),
              ('src', 'str'),
              ('link1_href', 'str'),
              ('link1_text', 'str'),
              ('link1_type', 'str'),
              ('link2_href', 'str'),
              ('link2_text', 'str'),
              ('link2_type', 'str'),
              ('number', 'int'),
              ('type', 'str')]),
  'geometry': 'MultiLineString'},
 'crs': {'init': 'epsg:4326'},
 'crs_wkt': 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0,AUTHORITY[
↪"EPSG","8901"]],UNIT["degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS[
↪"Latitude",NORTH],AXIS["Longitude",EAST],AUTHORITY["EPSG","4326"]]'}
```

6.41.4 Loading GPX as dict

The GPX can also be loaded as dict for further processing of the contents of the GPX file using `load_gpx_as_dict(...)`. This dict contains the properties, the geometry including the coordinates of the data, the ID and the type of the data

[12]:
```
gpx_dict = gg.vector.load_gpx_as_dict(path=file_path+'Run.gpx', layer='tracks')
gpx_dict['geometry']['coordinates'][0][:5]
```

[12]:
```
[(8.496285, 52.705566),
 (8.49627, 52.705593),
 (8.496234, 52.705629),
 (8.496205, 52.705664),
 (8.496181, 52.705705)]
```

[13]:
```
gpx_dict.keys()
```

[13]:
```
dict_keys(['type', 'id', 'properties', 'geometry'])
```

[14]:
```
gpx_dict['type'], gpx_dict['id'], gpx_dict['properties']
```

[14]:
```
('Feature',
 '0',
 OrderedDict([('name', 'First half marathon distance of the year'),
              ('cmt', None),
              ('desc', None),
              ('src', None),
              ('link1_href', None),
              ('link1_text', None),
              ('link1_type', None),
              ('link2_href', None),
              ('link2_text', None),
              ('link2_type', None),
              ('number', None),
              ('type', '9')]))
```

[15]:
```
gpx_dict['geometry']['type']
```

[15]:
```
'MultiLineString'
```

6.41.5 Creating Shapely Base Geometry from GPX

In order to work with GPX data, a Shapely BaseGeometry can be created using `load_gpx_as_geometry(...)`.

[16]:
```
shape = gg.vector.load_gpx_as_geometry(path=file_path+'Run.gpx', layer='tracks')
shape
```

[16]:

[17]:
```
shape.wkt[:100]
```

[17]:
```
'MULTILINESTRING ((8.496285 52.705566, 8.49627 52.705593, 8.496234 52.705629, 8.496205␣
↪52.705664, 8.4'
```

6.41.6 Creating GeoData from Geometry

A GeoDataFrame containing the created geometry can easily be created. Notice that the CRS attribute of the GPX collection was provided.

[18]:
```
import geopandas as gpd

gdf = gpd.GeoDataFrame(geometry=[shape], crs=gpx.crs)
gdf
```

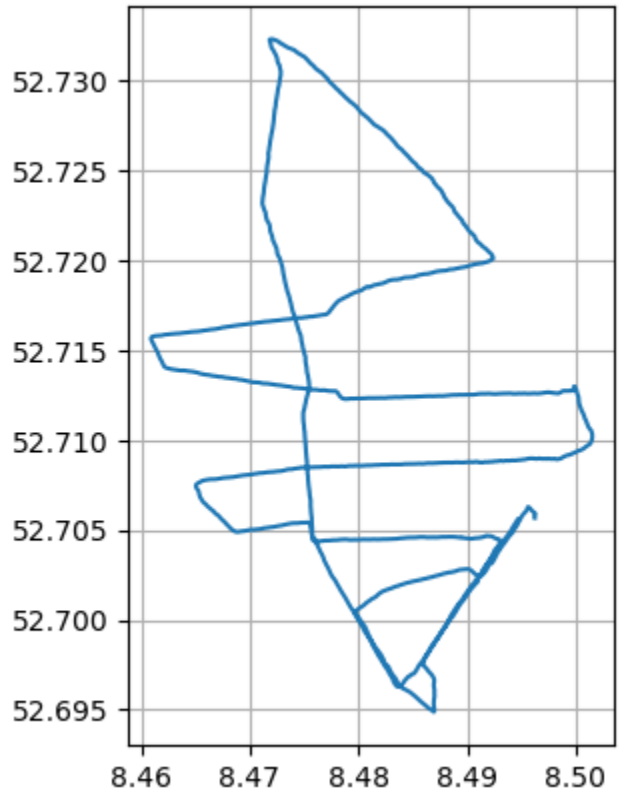
[18]:
```
                                             geometry
0  MULTILINESTRING ((8.49629 52.70557, 8.49627 52...
```

[19]:
```
gdf.crs
```

[19]:
```
<Geographic 2D CRS: +init=epsg:4326 +type=crs>
Name: WGS 84
Axis Info [ellipsoidal]:
- lon[east]: Longitude (degree)
- lat[north]: Latitude (degree)
Area of Use:
- name: World.
- bounds: (-180.0, -90.0, 180.0, 90.0)
Datum: World Geodetic System 1984 ensemble
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
```

And the data can be plotted.

[20]:
```
import matplotlib.pyplot as plt

gdf.plot()
plt.grid()
```



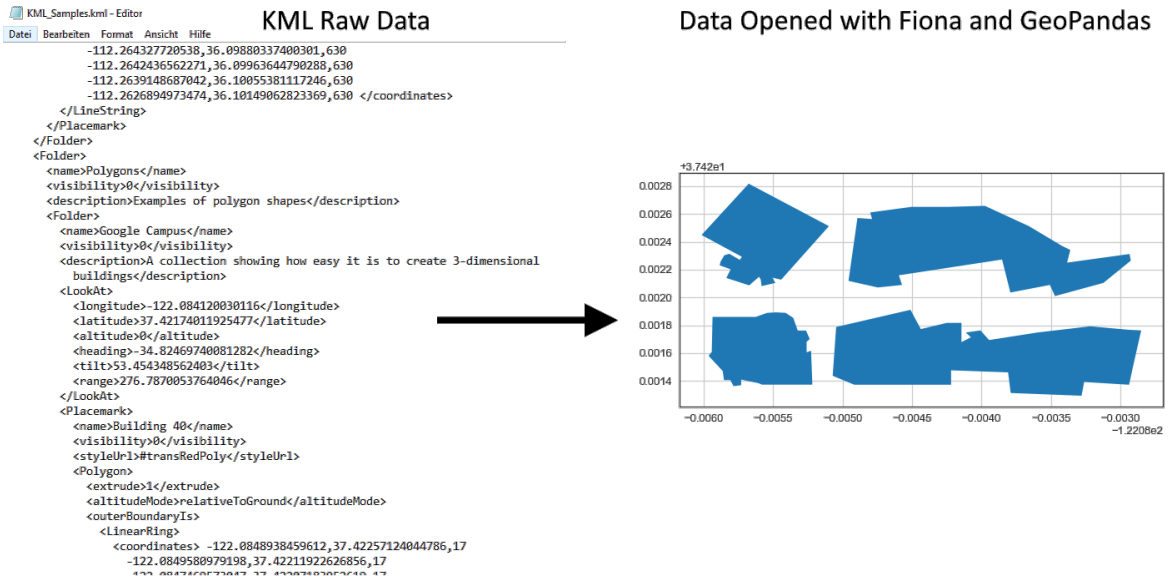
The length of the track is approximately 21139 m and therefore slightly longer than a half marathon distance (21097 m).

[21]:
```
gdf.to_crs(4647).loc[0].geometry.length
```

[21]:
```
21139.875974842187
```

6.42 41 Working with KML data

KML is a file format used to display geographic data in an Earth browser such as Google Earth. KML uses a tag-based structure with nested elements and attributes and is based on the XML standard.



Source: https://developers.google.com/kml/documentation/kml_tut

6.42.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/41_working_with_kml_data/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="41_working_with_kml_data.zip",␣
↪dirpath=file_path)
```

```
Downloading file '41_working_with_kml_data.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F41_working_with_kml_data.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\tutorial\data\41_working_with_kml_data'.
```

6.42.2 Loading the data

The data used for demonstration purposes here was downloaded from https://developers.google.com/kml/documentation/KML_Samples.kml. The different layers the KML consists of can be listed using fionas `listlayers(...)` function. When using GeoPandas, the KML driver has to be activated using `gpd.io.file.fiona.drvsupport.supported_drivers['KML'] = 'rw'`. When loading a layer with GeoPandas, the driver needs to be provided as well as the layer name. If no layer is provided, only the first element will be loaded.

[8]:
```
import geopandas as gpd
import fiona

gpd.io.file.fiona.drvsupport.supported_drivers['KML'] = 'rw'

layer_list = fiona.listlayers(file_path + 'KML_Samples.kml')
layer_list
```

[8]:
```
['Placemarks',
 'Highlighted Icon',
 'Paths',
 'Google Campus',
 'Extruded Polygon',
 'Absolute and Relative']
```

Loading Placemarks

[16]:
```
gdf_placemarks = gpd.read_file(filename=file_path+'KML_Samples.kml', driver='KML' ,␣
↪layer='Placemarks')
gdf_placemarks
```

[16]:
```
                Name                                          Description  \
0    Simple placemark  Attached to the ground. Intelligently places i...
1  Floating placemark        Floats a defined distance above the ground.
2  Extruded placemark    Tethered to the ground by a customizable "tail"

                                geometry
0   POINT Z (-122.08220 37.42229 0.00000)
1  POINT Z (-122.08407 37.42200 50.00000)
2  POINT Z (-122.08577 37.42157 50.00000)
```

Loading Icon

[10]:
```
gdf_highlight = gpd.read_file(filename=file_path+'KML_Samples.kml', driver='KML' , layer=
↪'Highlighted Icon')
gdf_highlight
```

[10]:
```
                 Name Description                                geometry
0  Roll over this icon             POINT Z (-122.08565 37.42243 0.00000)
```

Loading Paths

[11]:
```
gdf_paths = gpd.read_file(filename=file_path+'KML_Samples.kml', driver='KML' , layer=
↪'Paths')
gdf_paths
```

[11]:
```
                Name                                          Description  \
0        Tessellated  If the <tessellate> tag has a value of 1, the ...
1      Untessellated  If the <tessellate> tag has a value of 0, the ...
2           Absolute                              Transparent purple line
3  Absolute Extruded        Transparent green wall with yellow outlines
4           Relative  Black line (10 pixels wide), height tracks ter...
5  Relative Extruded  Opaque blue walls with red outline, height tra...

                                        geometry
0  LINESTRING Z (-112.08142 36.10678 0.00000, -11...
1  LINESTRING Z (-112.08062 36.10673 0.00000, -11...
2  LINESTRING Z (-112.26565 36.09448 2357.00000, ...
3  LINESTRING Z (-112.25508 36.07955 2357.00000, ...
4  LINESTRING Z (-112.25328 36.09887 645.00000, -...
5  LINESTRING Z (-112.26566 36.09445 630.00000, -...
```

Loading Google Campus

[12]:
```
gdf_campus = gpd.read_file(filename=file_path+'KML_Samples.kml', driver='KML' , layer=
↪'Google Campus')
gdf_campus
```

[12]:
```
          Name Description                                           geometry
0  Building 40              POLYGON Z ((-122.08489 37.42257 17.00000, -122...
1  Building 41              POLYGON Z ((-122.08574 37.42227 17.00000, -122...
2  Building 42              POLYGON Z ((-122.08579 37.42136 25.00000, -122...
3  Building 43              POLYGON Z ((-122.08444 37.42177 19.00000, -122...
```

Loading Polygon

[13]:
```
gdf_polygon = gpd.read_file(filename=file_path+'KML_Samples.kml', driver='KML' , layer=
↪'Extruded Polygon')
gdf_polygon
```

[13]:
```
           Name Description                                            geometry
0  The Pentagon              POLYGON Z ((-77.05788 38.87253 100.00000, -77...
```

Loading Absolute and Relative

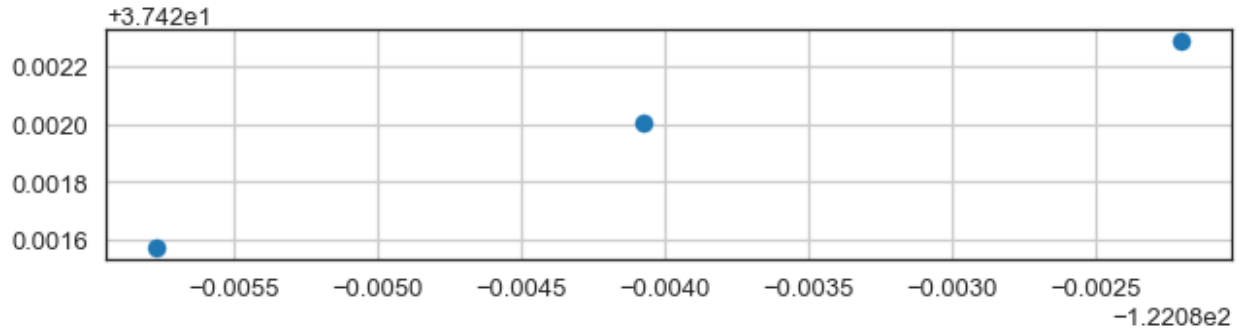
[14]:
```
gdf_absolute = gpd.read_file(filename=file_path+'KML_Samples.kml', driver='KML' , layer=
↪'Absolute and Relative')
gdf_absolute
```

[14]:
```
                Name Description  \
0           Absolute
1  Absolute Extruded
2           Relative
3  Relative Extruded

                                             geometry
0  POLYGON Z ((-112.33725 36.14889 1784.00000, -1...
1  POLYGON Z ((-112.33966 36.14638 1784.00000, -1...
2  POLYGON Z ((-112.33495 36.14989 100.00000, -11...
3  POLYGON Z ((-112.33488 36.15140 100.00000, -11...
```

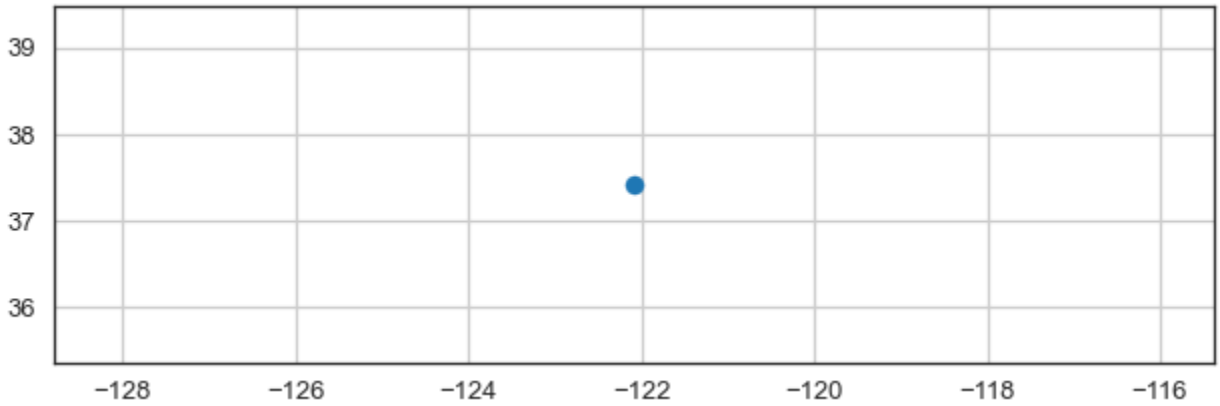
6.42.3 Plotting the Data

The data was just simply plotted to visualize the content of the KML file.

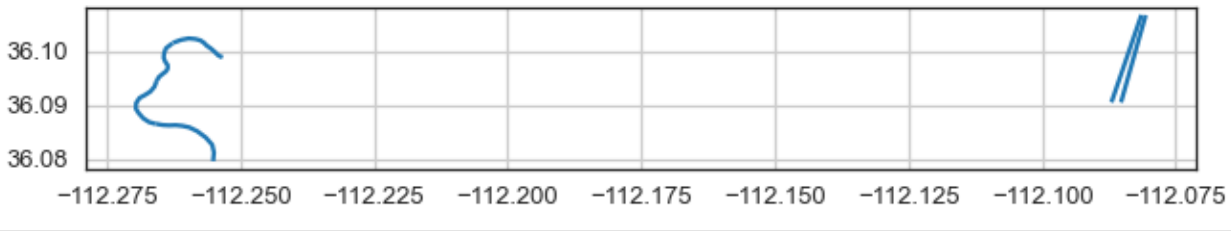
[35]:
```
import matplotlib.pyplot as plt

gdf_placemarks.plot(aspect='equal')
plt.grid()
```



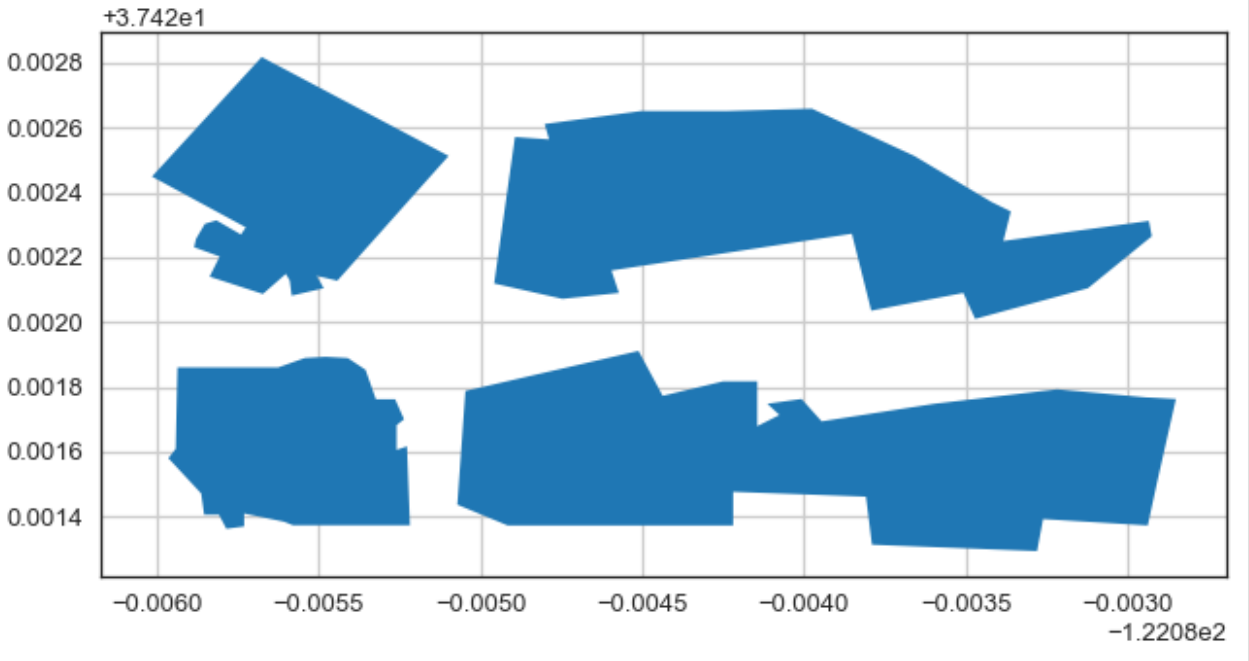
[34]:
```
gdf_highlight.plot(aspect='equal')
plt.grid()
```



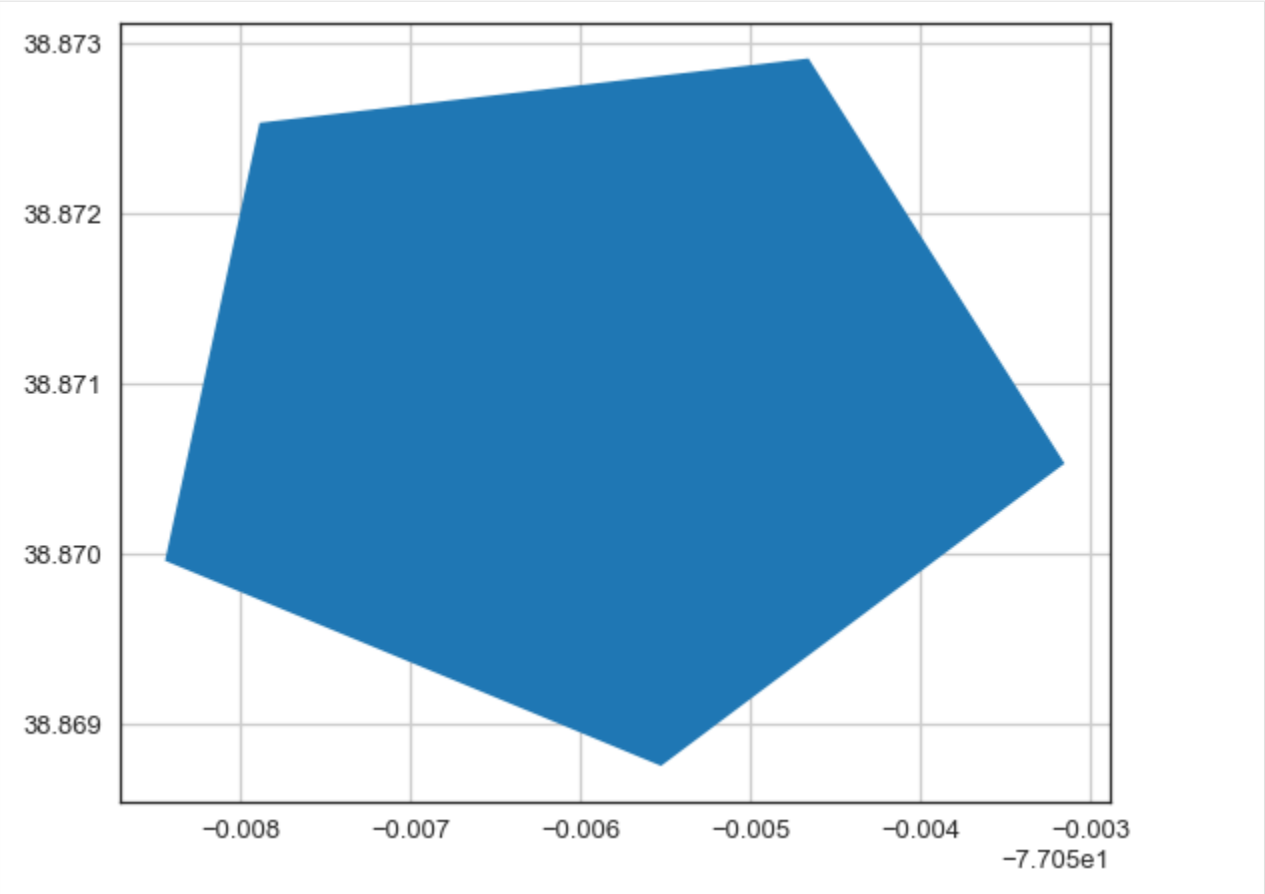
[33]:
```
gdf_paths.plot(aspect='equal')
plt.grid()
```



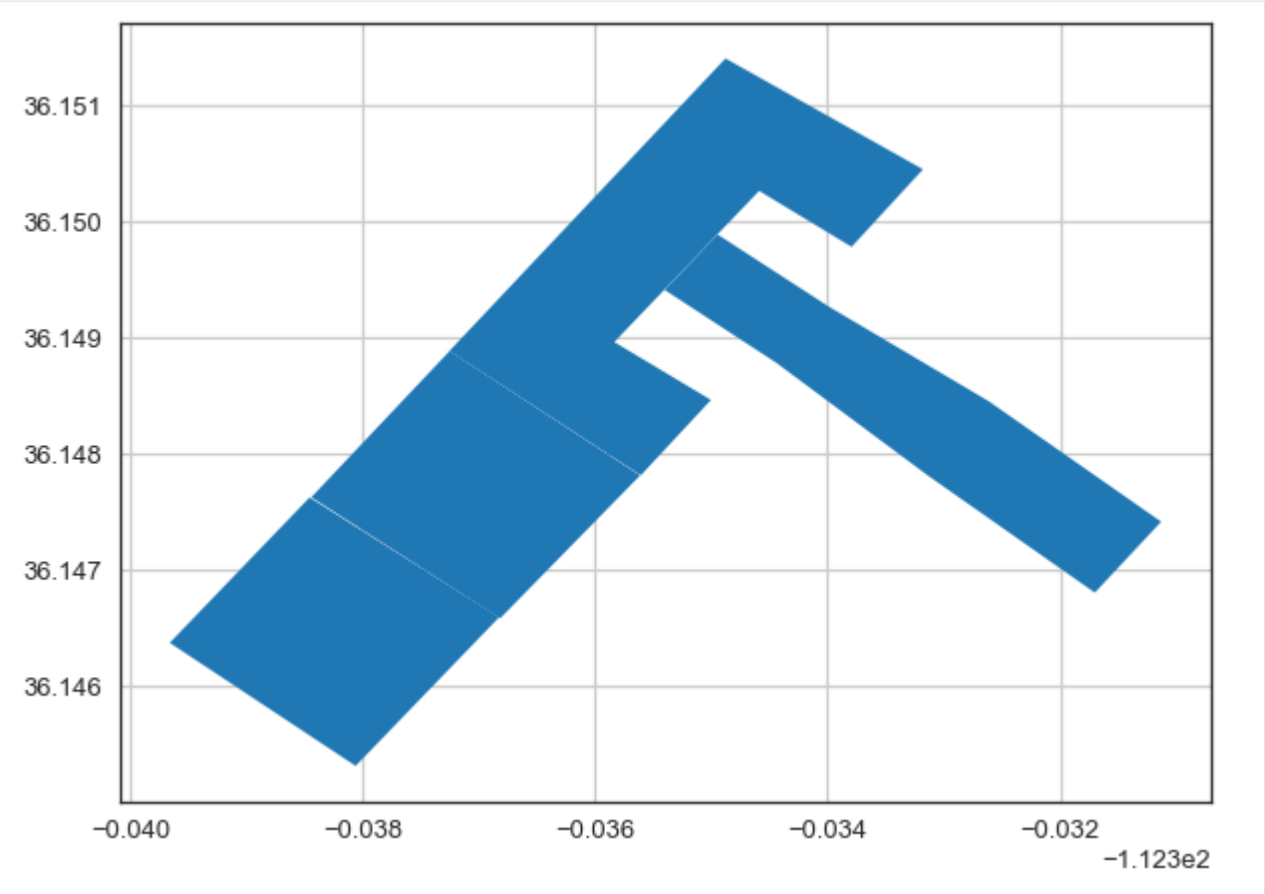
[32]:
```
gdf_campus.plot(aspect='equal')
plt.grid()
```



[30]:
```
gdf_polygon.plot(aspect='equal')
plt.grid()
```

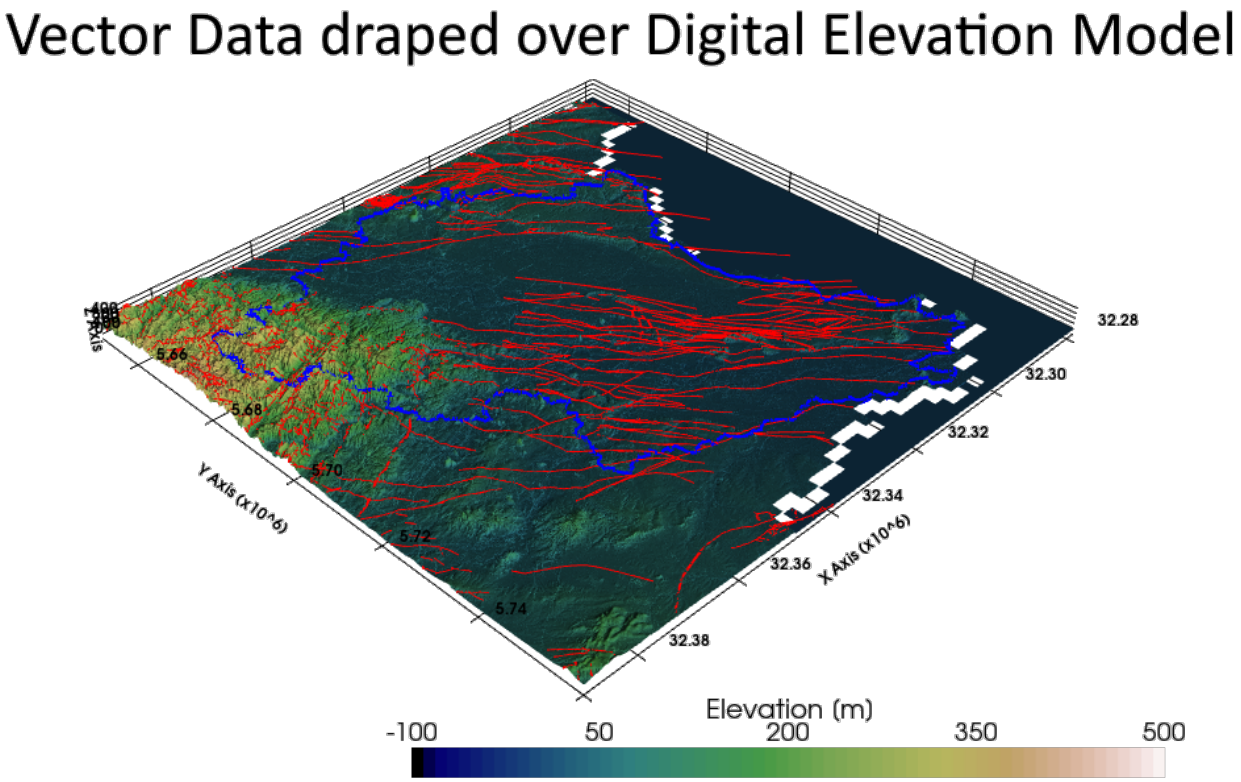


[27]:
```
gdf_absolute.plot(aspect='equal')
plt.grid()
```



6.43 42 Draping LineStrings over Digital Elevation Model in PyVista

When displaying LineString data in combination with a Digital Elevation Model, it is desired to map LineStrings representing elements such as roads on top of the Digital Elevation Model instead of above or below. Here it is shown how to do that with GemGIS.



6.43.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/42_draping_linestrings_over_dem_in_pyvista/'
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="42_draping_linestrings_over_dem_
↪in_pyvista.zip", dirpath=file_path)
```

6.43.2 Loading Mesh and Vector Data

The loaded digital elevation model will be used under Datenlizenz Deutschland – Zero – Version 2.0. It was obtained from the WCS Service https://www.wcs.nrw.de/geobasis/wcs_nw_dgm.

[3]:
```
import pyvista as pv
import rasterio

raster = rasterio.open(file_path + 'DEM50_EPSG_4647_clipped.tif')
```

(continues on next page)

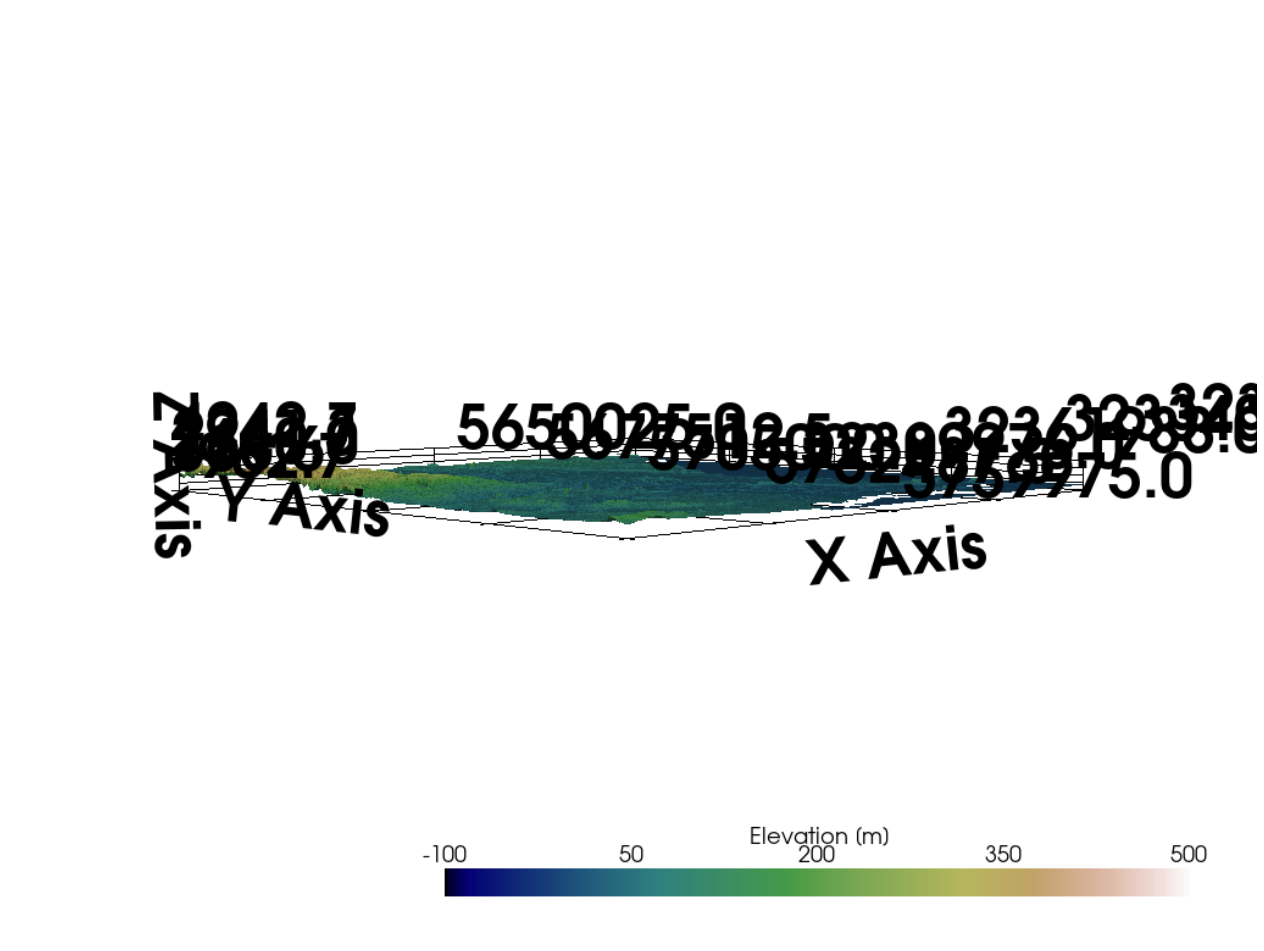
(continued from previous page)

```
topo = pv.read(file_path + 'topo.vtk')
topo
```

[3]:
```
StructuredGrid (0x1c816bec6a0)
  N Cells:      4923561
  N Points:     4928000
  X Bounds:     3.228e+07, 3.239e+07
  Y Bounds:     5.650e+06, 5.760e+06
  Z Bounds:     -9.627e+01, 4.243e+02
  Dimensions:   2200, 2240, 1
  N Arrays:     1
```

[4]:
```
import pyvista as pv

sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

# Adding DEM
p.add_mesh(topo, scalars='Elevation [m]', cmap='gist_earth', scalar_bar_args=sargs,␣
↪clim=[-100, 500])

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\pyvista\jupyter\
↪notebook.py:58: UserWarning: Failed to use notebook backend:

No module named 'trame'

Falling back to a static output.
  warnings.warn(
```



6.43.3 Creating 3D Line of County Boundary

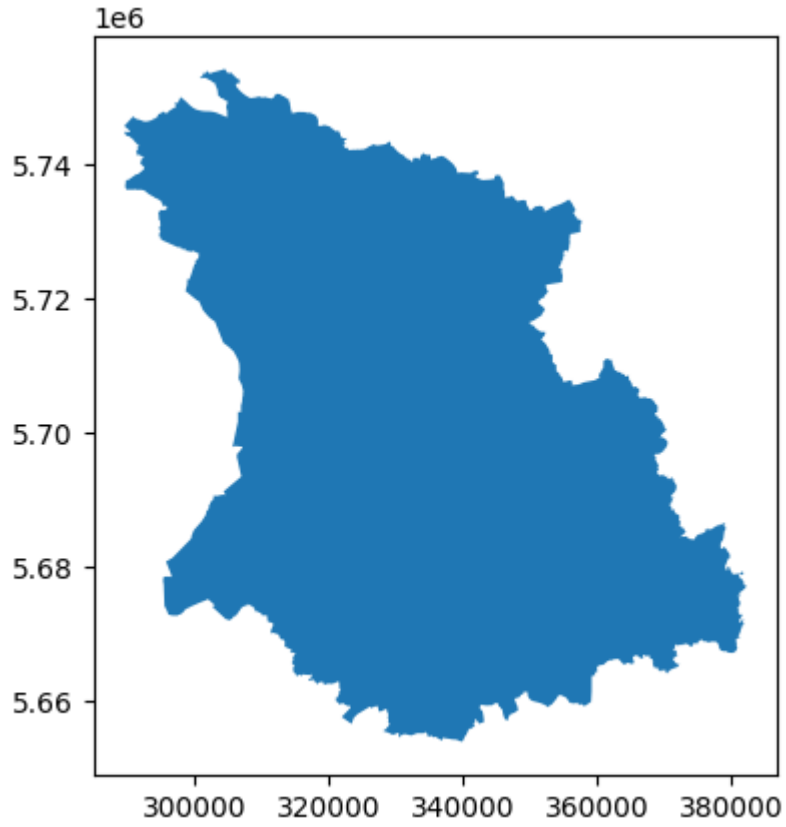
The boundary data was obtained from https://www.opengeodata.nrw.de/produkte/geobasis/vkg/dvg/dvg1/ and will be used under the same license as the Digital Elevation Model.

[5]:
```
import geopandas as gpd

boundary = gpd.read_file(file_path + 'dvg1rbz_nw.shp')
boundary = boundary[boundary['GN'] == 'Düsseldorf']
boundary.head()
```

[5]:
```
  ART          GN        KN       STAND
0   R  Düsseldorf  05100000  2020-03-04  \

                                              geometry
0  POLYGON ((295896.673 5747849.577, 295897.626 5...
```

[6]:
```
boundary.plot()
```

[6]:
```
<Axes: >
```



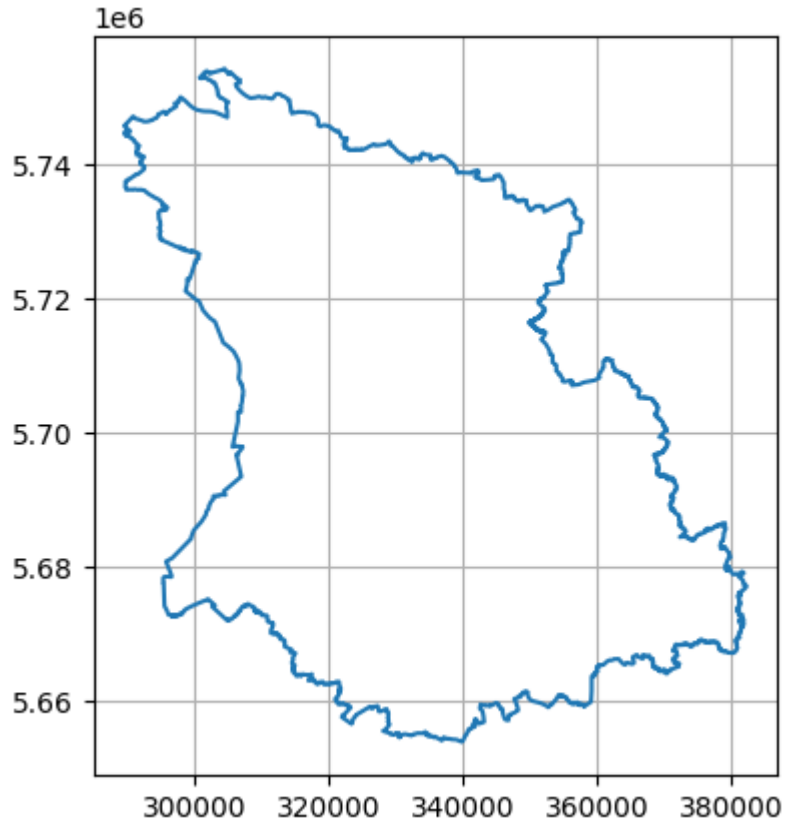
Exploding Polygon to LineString

The first is to explode the polygon to a LineString using `explode_polygons(..)`.

[7]:
```
boundary_line = gg.vector.explode_polygons(gdf=boundary)
boundary_line
```

[7]:
```
  ART          GN        KN       STAND
0   R  Düsseldorf  05100000  2020-03-04  \

                                              geometry
0  LINESTRING (295896.673 5747849.577, 295897.626...
```

[8]:
```
import matplotlib.pyplot as plt

boundary_line.plot()
plt.grid()
```



Extracting vertices from line

The next step is to extract the vertices from this one line using `extract_xyz(..)`.

[9]:
```
boundary_points = gg.vector.extract_xyz(gdf=boundary_line, dem=raster, target_crs='EPSG:
↪4647')
boundary_points
```

[9]:
```
      ART          GN        KN       STAND                          geometry  \
0       R  Düsseldorf  05100000  2020-03-04  POINT (32295896.673 5747849.577)
1       R  Düsseldorf  05100000  2020-03-04  POINT (32295897.626 5747850.839)
2       R  Düsseldorf  05100000  2020-03-04  POINT (32295943.179 5747880.192)
3       R  Düsseldorf  05100000  2020-03-04  POINT (32295959.070 5747891.749)
4       R  Düsseldorf  05100000  2020-03-04  POINT (32295975.984 5747906.266)
...    ..         ...       ...         ...                               ...
27446   R  Düsseldorf  05100000  2020-03-04  POINT (32295802.571 5747693.901)
27447   R  Düsseldorf  05100000  2020-03-04  POINT (32295840.751 5747767.801)
27448   R  Düsseldorf  05100000  2020-03-04  POINT (32295857.959 5747796.720)
27449   R  Düsseldorf  05100000  2020-03-04  POINT (32295870.009 5747814.641)
27450   R  Düsseldorf  05100000  2020-03-04  POINT (32295896.673 5747849.577)

                X          Y     Z
0     32295896.67 5747849.58 11.49
1     32295897.63 5747850.84 11.56
```

(continues on next page)

(continued from previous page)

```
2     32295943.18 5747880.19 11.44
3     32295959.07 5747891.75 11.70
4     32295975.98 5747906.27 11.55
...           ...        ...   ...
27446 32295802.57 5747693.90 11.58
27447 32295840.75 5747767.80 11.44
27448 32295857.96 5747796.72 11.65
27449 32295870.01 5747814.64 11.49
27450 32295896.67 5747849.58 11.49

[27451 rows x 8 columns]
```

Creating LineString with Z component

A LineString with a Z component can be created using `create_linestring_from_xyz_points(..)`.

[10]:
```
boundary_line_z = gg.vector.create_linestring_from_xyz_points(points=boundary_points)
boundary_line_z
```

[10]:

Creating GeoDataFrame from LineString

[11]:
```
gdf_boundary = gpd.GeoDataFrame(geometry=[boundary_line_z])
gdf_boundary
```

[11]:
```
                                            geometry
0  LINESTRING Z (32295896.673 5747849.577 11.490,...
```

Creating 3D Lines

The last step is to create the 3D lines and return the data as PyVista PolyData.

[12]:
```
boundary_pv = gg.visualization.create_lines_3d_polydata(gdf=gdf_boundary)
boundary_pv['Label'] = ['Regierungsbezirk Dusseldorf']
boundary_pv
```

[12]:
```
PolyData (0x1c825a43e80)
  N Cells:    1
  N Points:   27451
  N Strips:   0
  X Bounds:   3.229e+07, 3.238e+07
  Y Bounds:   5.654e+06, 5.754e+06
  Z Bounds:   -4.745e+01, 3.433e+02
  N Arrays:   1
```

6.43.4 Creating 3D lines of Faults

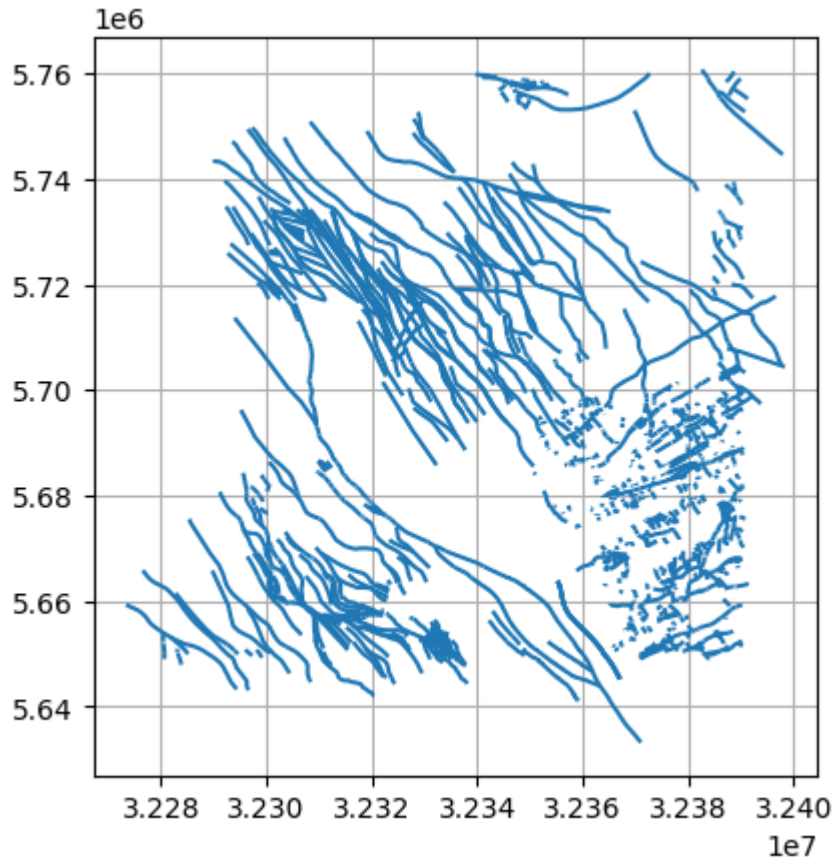
The same workflow can be applied to a dataset with multiple LineStrings. In this case the faults that were mapped in the area.

[13]:
```
bbox = (32250000, 5650000, 32390000, 5760000)
```

[14]:
```
faults = gpd.read_file(file_path + 'gg_nrw_geotekst_l.shp', bbox=bbox)
faults.head()
```

[14]:
```
   OBJECTID_1  OBJECTID_2                          Layer_Name
0       52.00          52      Störung040_Quadrather Sprung_SW  \
1       99.00          99  Störung067_Peringshofer Sprung 1_NE
2      100.00         100        Störung068_Sandberg Sprung_NE
3      101.00         101       Störung069_Bedburger Sprung_NE
4      102.00         102     Störung070_Bedburger Sprung 1_NE

                         Quelle              ST_NAME  OBJECTID  Id
0  Erft Scholle RWE Projekt 2015    Quadrather Sprung         0   0  \
1  Erft Scholle RWE Projekt 2015  Peringshofer Sprung         0   0
2  Erft Scholle RWE Projekt 2015      Sandberg Sprung         0   0
3  Erft Scholle RWE Projekt 2015     Bedburger Sprung         0   0
4  Erft Scholle RWE Projekt 2015     Bedburger Sprung         0   0

   ssymbol_QB  ST_NR  ST_ART_ID  ...  DIP_3D               STOE_UEB_3
0         NaN   0.00          0  ...   62.50      regionale Bedeutung  \
1         NaN   0.00          0  ...   62.50         lokale Bedeutung
2         NaN   0.00          0  ...   62.50  überregionale Bedeutung
3         NaN   0.00          0  ...   62.50         lokale Bedeutung
4         NaN   0.00          0  ...   62.50         lokale Bedeutung

   STOE_TOP_H                      STOE_BASIS  ST_ART_3D  Shape_Le_2
0         DGM  Tertiaer_b/Praeperm_t/Karbon_t        NaN     9317.89  \
1         DGM  Tertiaer_b/Praeperm_t/Karbon_t        NaN     3304.34
2         DGM  Tertiaer_b/Praeperm_t/Karbon_t        NaN     3242.54
3         DGM  Tertiaer_b/Praeperm_t/Karbon_t        NaN     6404.51
4         DGM  Tertiaer_b/Praeperm_t/Karbon_t        NaN     1656.99

   Versatz_TB  Doku_Versa Shape_Le_3
0        0.00         NaN    9317.89  \
1        0.00         NaN    3304.34
2        0.00         NaN    3242.54
3        0.00         NaN    6404.51
4        0.00         NaN    1656.99

                                          geometry
0  LINESTRING (32338164.037 5644376.205, 32338092...
1  LINESTRING (32332455.392 5650412.839, 32332407...
2  LINESTRING (32331619.679 5651276.996, 32331513...
3  LINESTRING (32333471.001 5648319.956, 32333423...
4  LINESTRING (32332714.438 5649401.541, 32332618...

[5 rows x 47 columns]
```

[15]:
```
import matplotlib.pyplot as plt

faults.plot()
plt.grid()
```



Extracting vertices from line

The vertices will be extracted from the lines again. Here, it is important to NOT reset the index of the resulting GeoDataFrame!

[16]:
```
faults = faults[faults.is_valid]
faults = faults[~faults.is_empty]
faults = faults.explode().reset_index(drop=True)
faults_xyz = gg.vector.extract_xyz(gdf=faults, dem=raster, reset_index=False)
faults_xyz.head()
```

[16]:
```
     OBJECTID_1  OBJECTID_2                       Layer_Name
0 0       52.00          52  Störung040_Quadrather Sprung_SW  \
  0       52.00          52  Störung040_Quadrather Sprung_SW
  0       52.00          52  Störung040_Quadrather Sprung_SW
  0       52.00          52  Störung040_Quadrather Sprung_SW
  0       52.00          52  Störung040_Quadrather Sprung_SW

                          Quelle            ST_NAME  OBJECTID  Id
```

(continues on next page)

(continued from previous page)

```
0 0  Erft Scholle RWE Projekt 2015  Quadrather Sprung         0   0  \
  0  Erft Scholle RWE Projekt 2015  Quadrather Sprung         0   0
  0  Erft Scholle RWE Projekt 2015  Quadrather Sprung         0   0
  0  Erft Scholle RWE Projekt 2015  Quadrather Sprung         0   0
  0  Erft Scholle RWE Projekt 2015  Quadrather Sprung         0   0

     ssymbol_QB  ST_NR  ST_ART_ID  ...                       STOE_BASIS
0 0         NaN   0.00          0  ...  Tertiaer_b/Praeperm_t/Karbon_t  \
  0         NaN   0.00          0  ...  Tertiaer_b/Praeperm_t/Karbon_t
  0         NaN   0.00          0  ...  Tertiaer_b/Praeperm_t/Karbon_t
  0         NaN   0.00          0  ...  Tertiaer_b/Praeperm_t/Karbon_t
  0         NaN   0.00          0  ...  Tertiaer_b/Praeperm_t/Karbon_t

     ST_ART_3D  Shape_Le_2  Versatz_TB  Doku_Versa  Shape_Le_3
0 0        NaN     9317.89        0.00         NaN     9317.89  \
  0        NaN     9317.89        0.00         NaN     9317.89
  0        NaN     9317.89        0.00         NaN     9317.89
  0        NaN     9317.89        0.00         NaN     9317.89
  0        NaN     9317.89        0.00         NaN     9317.89

                            geometry           X          Y       Z
0 0  POINT (32338164.037 5644376.205) 32338164.04 5644376.20 9999.00
  0  POINT (32338092.156 5644454.183) 32338092.16 5644454.18 9999.00
  0  POINT (32338020.275 5644532.162) 32338020.28 5644532.16 9999.00
  0  POINT (32337976.168 5644579.490) 32337976.17 5644579.49 9999.00
  0  POINT (32337847.430 5644717.324) 32337847.43 5644717.32 9999.00

[5 rows x 50 columns]
```

Creating LineStrings with Z component

As before, LineStrings with Z components will be created. This time, the function `create_linestrings_from_xyz_points(..)` is used.

[17]:
```
faults_lines_z = gg.vector.create_linestrings_from_xyz_points(gdf=faults_xyz, groupby=
↪'OBJECTID_2', return_gdf=True)
faults_lines_z
```

[17]:
```
      OBJECTID_1  OBJECTID_2                          Layer_Name
0          52.00          52     Störung040_Quadrather Sprung_SW  \
1          99.00          99  Störung067_Peringshofer Sprung 1_NE
2         100.00         100       Störung068_Sandberg Sprung_NE
3         101.00         101      Störung069_Bedburger Sprung_NE
4         102.00         102    Störung070_Bedburger Sprung 1_NE
...          ...         ...                                  ...
1510    13323.00       13324                                  NaN
1511    13324.00       13325                                  NaN
1512    13325.00       13326                                  NaN
1513    13331.00       13332                                  NaN
1514    13333.00       13334                                  NaN
```

(continues on next page)

(continued from previous page)

```
                                      Quelle                  ST_NAME
0                 Erft Scholle RWE Projekt 2015        Quadrather Sprung  \
1                 Erft Scholle RWE Projekt 2015      Peringshofer Sprung
2                 Erft Scholle RWE Projekt 2015          Sandberg Sprung
3                 Erft Scholle RWE Projekt 2015         Bedburger Sprung
4                 Erft Scholle RWE Projekt 2015         Bedburger Sprung
...                                         ...                     ...
1510                           3DLandesmodell         Hervester Sprung
1511                           3DLandesmodell           Tertius Sprung
1512                           3DLandesmodell          Rhader Stoerung
1513                                  isgk100  Bergische Überschiebung
1514  gk25_4707_Mettmann_Preußenkarte_Schriiver                     NaN

      OBJECTID  Id  ssymbol_QB  ST_NR  ST_ART_ID  ...  DIP_3D
0            0   0         NaN   0.00          0  ...   62.50  \
1            0   0         NaN   0.00          0  ...   62.50
2            0   0         NaN   0.00          0  ...   62.50
3            0   0         NaN   0.00          0  ...   62.50
4            0   0         NaN   0.00          0  ...   62.50
...        ...  ..         ...    ...        ...  ...     ...
1510         0   0         NaN   0.00          0  ...   79.00
1511         0   0         NaN   0.00          0  ...   80.00
1512         0   0         NaN   0.00          0  ...   83.00
1513         0   0         NaN   0.00          0  ...   80.00
1514         0   0         NaN   0.00          0  ...    0.00

                  STOE_UEB_3    STOE_TOP_H                      STOE_BASIS
0         regionale Bedeutung           DGM  Tertiaer_b/Praeperm_t/Karbon_t  \
1            lokale Bedeutung           DGM  Tertiaer_b/Praeperm_t/Karbon_t
2     überregionale Bedeutung           DGM  Tertiaer_b/Praeperm_t/Karbon_t
3            lokale Bedeutung           DGM  Tertiaer_b/Praeperm_t/Karbon_t
4            lokale Bedeutung           DGM  Tertiaer_b/Praeperm_t/Karbon_t
...                       ...           ...                             ...
1510  überregionale Bedeutung  Oberkreide_b             Praeperm_t/Karbon_t
1511  überregionale Bedeutung  Oberkreide_b             Praeperm_t/Karbon_t
1512  überregionale Bedeutung  Oberkreide_b             Praeperm_t/Karbon_t
1513         lokale Bedeutung           DGM                             NaN
1514      regionale Bedeutung           NaN                             NaN

        ST_ART_3D  Shape_Le_2  Versatz_TB  Doku_Versa Shape_Le_3
0             NaN     9317.89        0.00         NaN    9317.89  \
1             NaN     3304.34        0.00         NaN    3304.34
2             NaN     3242.54        0.00         NaN    3242.54
3             NaN     6404.51        0.00         NaN    6404.51
4             NaN     1656.99        0.00         NaN    1656.99
...           ...         ...         ...         ...        ...
1510          NaN    32201.82        0.00         NaN   32201.82
1511          NaN    45355.26        0.00         NaN   45355.26
1512          NaN    50431.89        0.00         NaN   50431.89
1513  Abschiebung      627.73        0.00         NaN     627.73
1514          NaN     7815.04        0.00         NaN    7815.04
```

(continues on next page)

(continued from previous page)

```
                                                 geometry
0     LINESTRING Z (32332790.835 5650048.917 83.820,...
1     LINESTRING Z (32332455.392 5650412.839 69.640,...
2     LINESTRING Z (32331619.679 5651276.996 54.820,...
3     LINESTRING Z (32331896.183 5650024.802 69.040,...
4     LINESTRING Z (32331991.560 5650081.463 69.630,...
...                                                 ...
1510  LINESTRING Z (32372522.961 5716737.961 46.730,...
1511  LINESTRING Z (32386330.289 5705646.290 88.140,...
1512  LINESTRING Z (32365258.965 5733746.961 52.790,...
1513  LINESTRING Z (32372730.326 5650769.467 181.760...
1514  LINESTRING Z (32357075.547 5675077.272 82.920,...

[1515 rows x 47 columns]
```

Creating 3D Lines

The last step is to create the 3D lines and return the data as PyVista PolyData.

[19]:
```
faults_pv = gg.visualization.create_lines_3d_polydata(gdf=faults_lines_z)
faults_pv
```

[19]:
```
PolyData (0x1c81d0b2fe0)
  N Cells:    1515
  N Points:   36539
  N Strips:   0
  X Bounds:   3.228e+07, 3.239e+07
  Y Bounds:   5.650e+06, 5.760e+06
  Z Bounds:   -8.993e+01, 4.156e+02
  N Arrays:   0
```

6.43.5 Plotting the data on top of the DEM

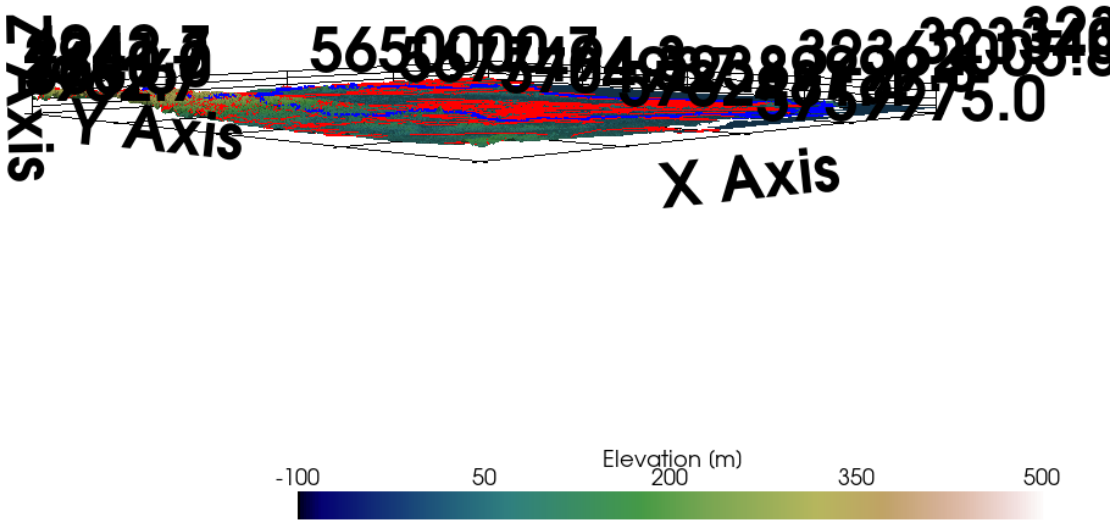
[20]:
```
import pyvista as pv

sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

# Adding DEM
p.add_mesh(topo, scalars='Elevation [m]', cmap='gist_earth', scalar_bar_args=sargs,␣
↪clim=[-100, 500])

# Adding boundary
p.add_mesh(boundary_pv, color='blue', line_width=5)

# Adding faults
p.add_mesh(faults_pv,color='red', line_width=2)

p.set_background('white')
```

(continues on next page)

(continued from previous page)

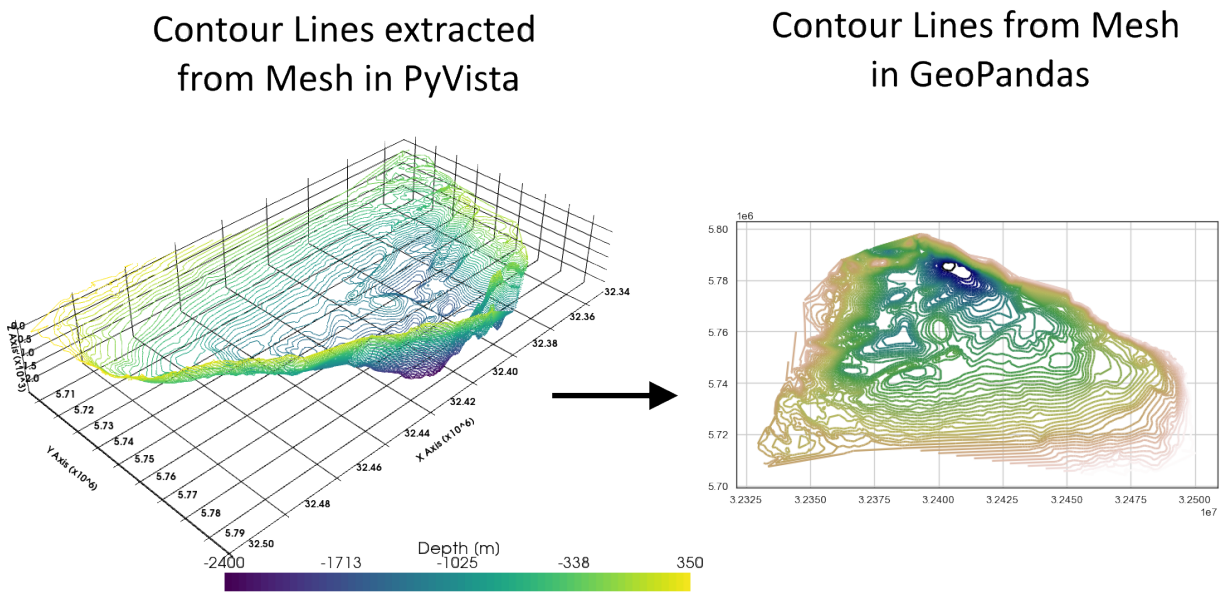
```
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\pygeomechanical\lib\site-packages\pyvista\jupyter\
↪notebook.py:58: UserWarning: Failed to use notebook backend:

No module named 'trame'

Falling back to a static output.
  warnings.warn(
```



6.44 43 Creating LineStrings from PyVista Contour Lines

PyVista has the capability to create contour lines from a mesh using the built-in `contours` function. It may be desirable to visualize these contour lines again in 2D within a GIS environment. The functionality to convert PyVista contour lines into Shapely LineStrings and GeoDataFrames was hence implemented in GemGIS.



6.44.1 Set File Paths

If you downloaded the latest `GemGIS` from the Github repository, append the path so that the package can be imported successfully. In addition, the file path to the files within the `gemgis_data` folder is set. You can download the data here if you have not done so.

[1]:
```
import gemgis as gg

file_path ='data/43_create_linestrings_from_pyvista_contours/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="43_create_linestrings_from_
↪pyvista_contours.zip", dirpath=file_path)
```

```
Downloading file '43_create_linestrings_from_pyvista_contours.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F43_create_linestrings_from_pyvista_
↪contours.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\
↪data\43_create_linestrings_from_pyvista_contours'.
```

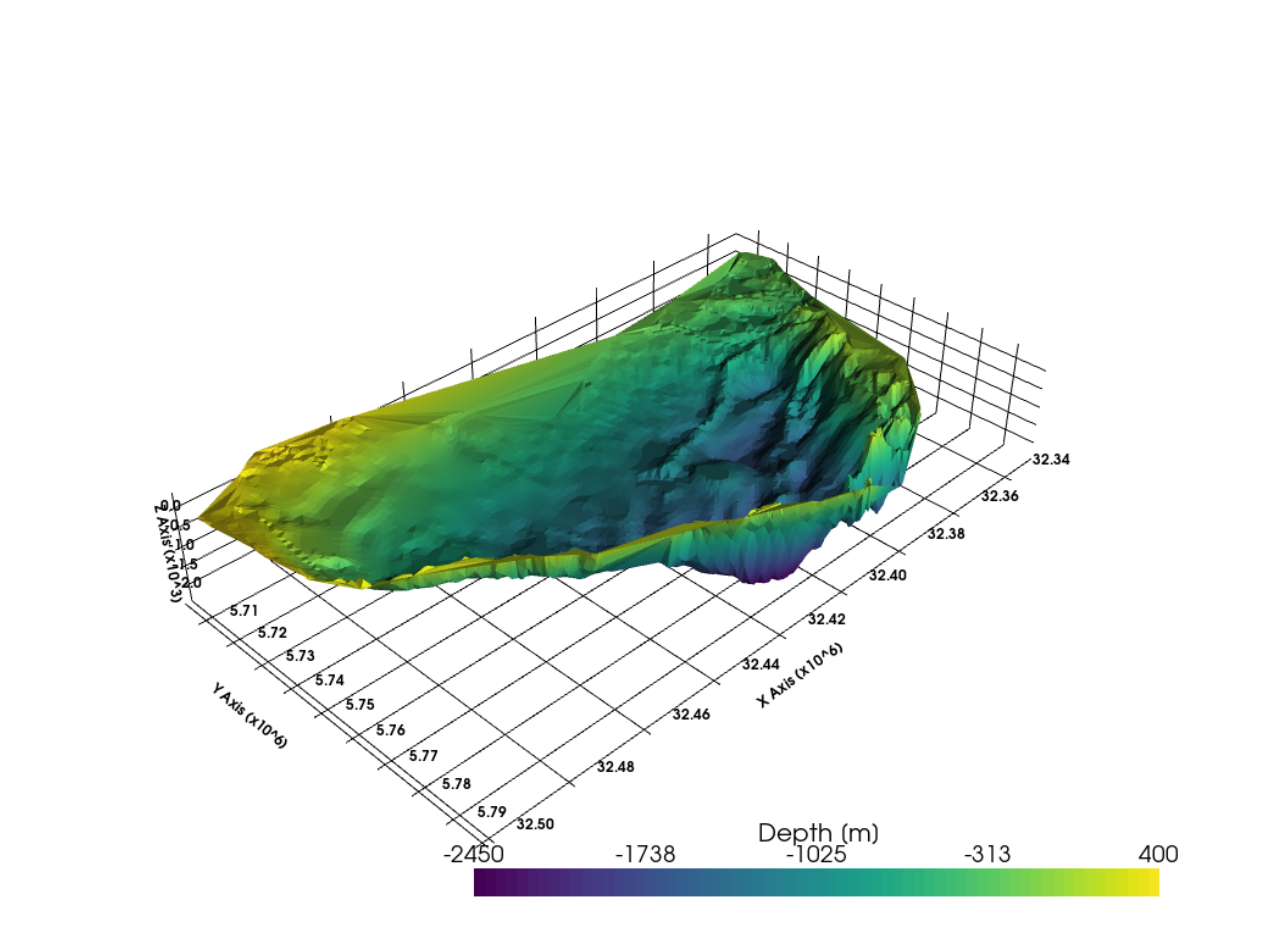
6.44.2 Loading Mesh

[2]:
```
import pyvista as pv

mesh = pv.read(file_path + 'surf.vtk')
mesh
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
PolyData (0x2195cf28700)
  N Cells:  45536
  N Points: 23009
  X Bounds: 3.233e+07, 3.250e+07
  Y Bounds: 5.702e+06, 5.798e+06
  Z Bounds: -2.450e+03, 4.000e+02
  N Arrays: 1
```

[3]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh, scalars='Depth [m]', scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```



6.44.3 Calculating Contours

The contours can be calculated with the built-in `contour(..)` function in PyVista. Isosurfaces are defined every 50 meters.

[4]:
```
import numpy as np
contours = mesh.contour(isosurfaces=np.arange(-2450, 400,50))
contours
```

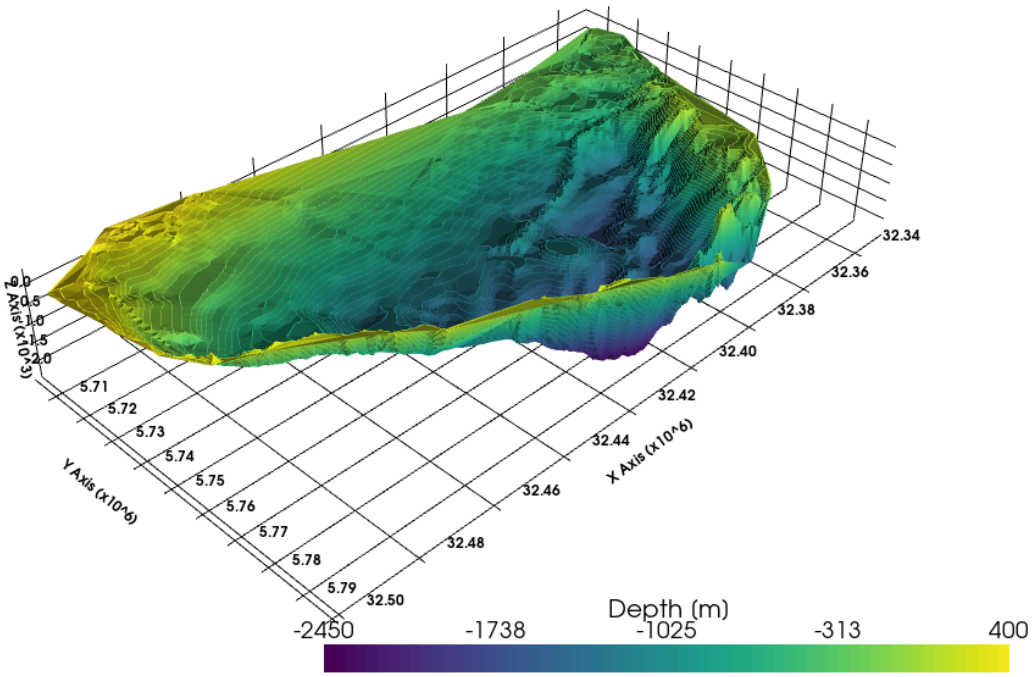
[4]:
```
PolyData (0x219622267c0)
  N Cells:  36337
  N Points: 36178
  X Bounds: 3.233e+07, 3.250e+07
  Y Bounds: 5.704e+06, 5.798e+06
  Z Bounds: -2.400e+03, 3.500e+02
  N Arrays: 1
```

[5]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
```

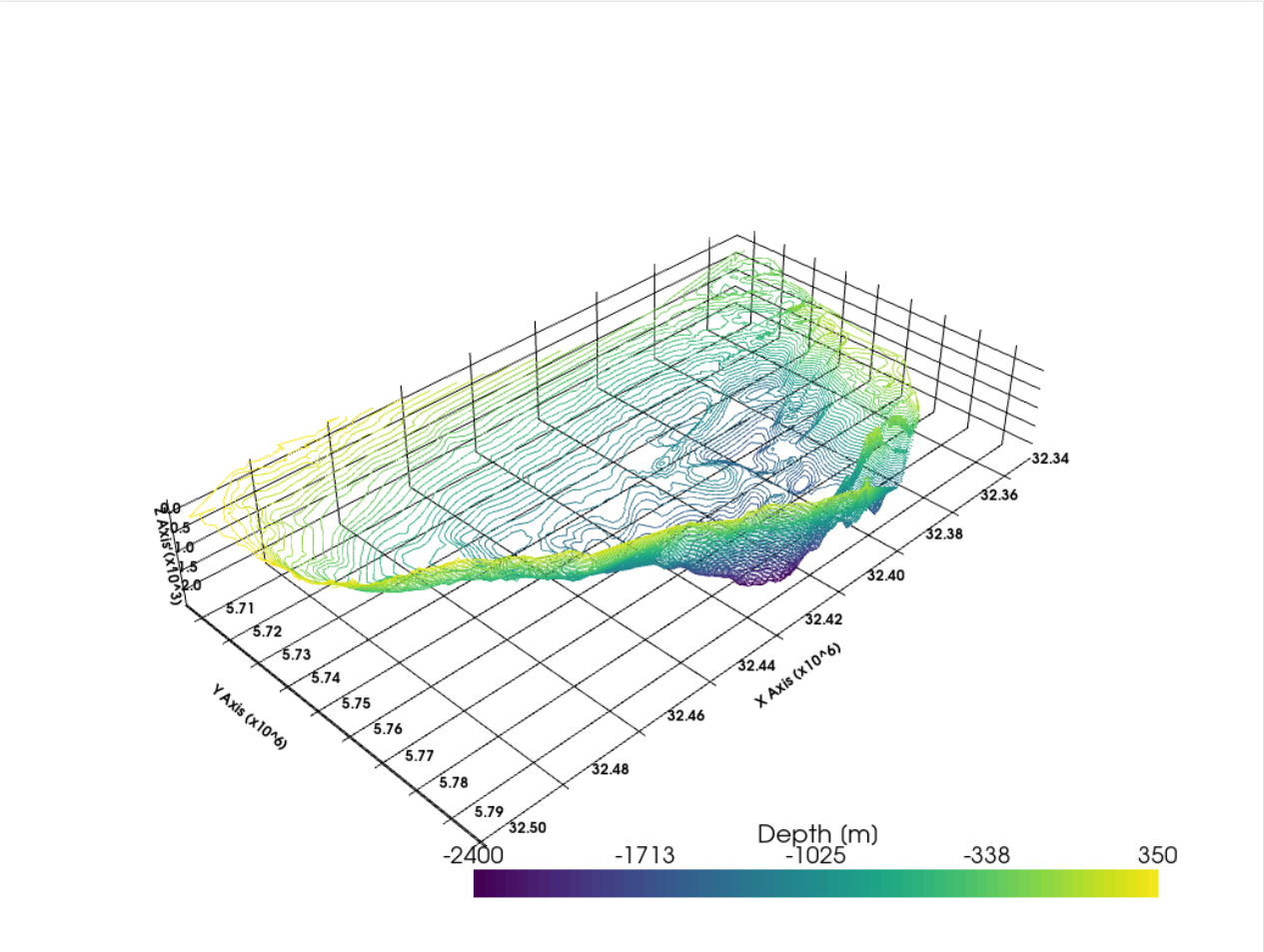
(continues on next page)

(continued from previous page)

```
p.add_mesh(mesh, scalars='Depth [m]', scalar_bar_args=sargs)
p.add_mesh(contours, scalars='Depth [m]', scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```



[6]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(contours, scalars='Depth [m]', scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```



6.44.4 Extracting LineStrings

The LineStrings can be extracted using `create_linestrings_from_contours(..)`. If the results are returned as GeoDataFrame, a CRS can be set for the gdf.

[7]:
```
gdf_linestrings = gg.vector.create_linestrings_from_contours(contours=contours, return_
↪gdf=True, crs='EPSG:4647')
gdf_linestrings
```

[7]:

	geometry	Z
0	LINESTRING Z (32409587.930 5780538.824 -2350.0...	-2350.00
1	LINESTRING Z (32407304.336 5777048.086 -2050.0...	-2050.00
2	LINESTRING Z (32408748.977 5778005.047 -2200.0...	-2200.00
3	LINESTRING Z (32403693.547 5786613.994 -2400.0...	-2400.00
4	LINESTRING Z (32404738.664 5782672.480 -2350.0...	-2350.00
...	...	...
36332	LINESTRING Z (32472712.875 5705828.297 350.000...	350.00
36333	LINESTRING Z (32476371.684 5706520.105 350.000...	350.00
36334	LINESTRING Z (32476542.145 5706398.895 350.000...	350.00
36335	LINESTRING Z (32476538.465 5706556.277 350.000...	350.00
36336	LINESTRING Z (32476540.793 5706593.047 350.000...	350.00

(continues on next page)

(continued from previous page)

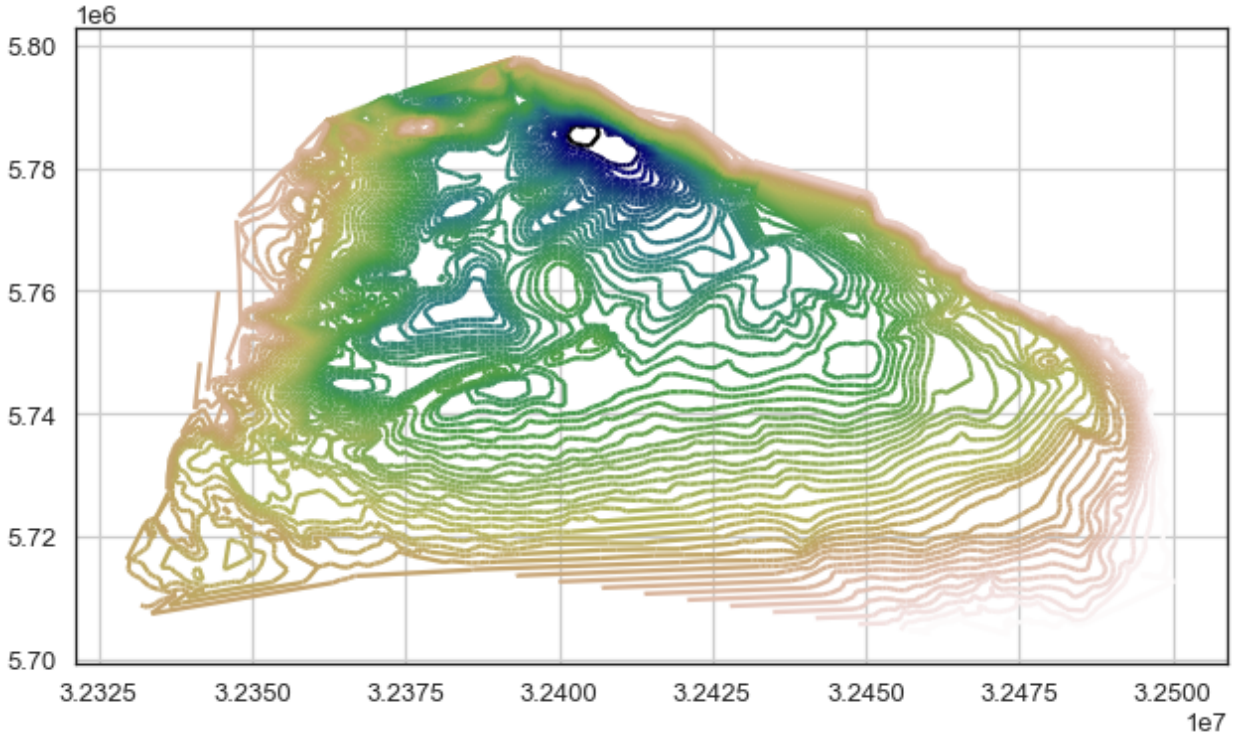
```
[36337 rows x 2 columns]
```

[8]:
```
gdf_linestrings.crs
```

[8]:
```
<Projected CRS: EPSG:4647>
Name: ETRS89 / UTM zone 32N (zE-N)
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Germany - 6°E to 12°E
- bounds: (6.0, 47.27, 12.0, 55.47)
Coordinate Operation:
- name: UTM zone 32N with prefix
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

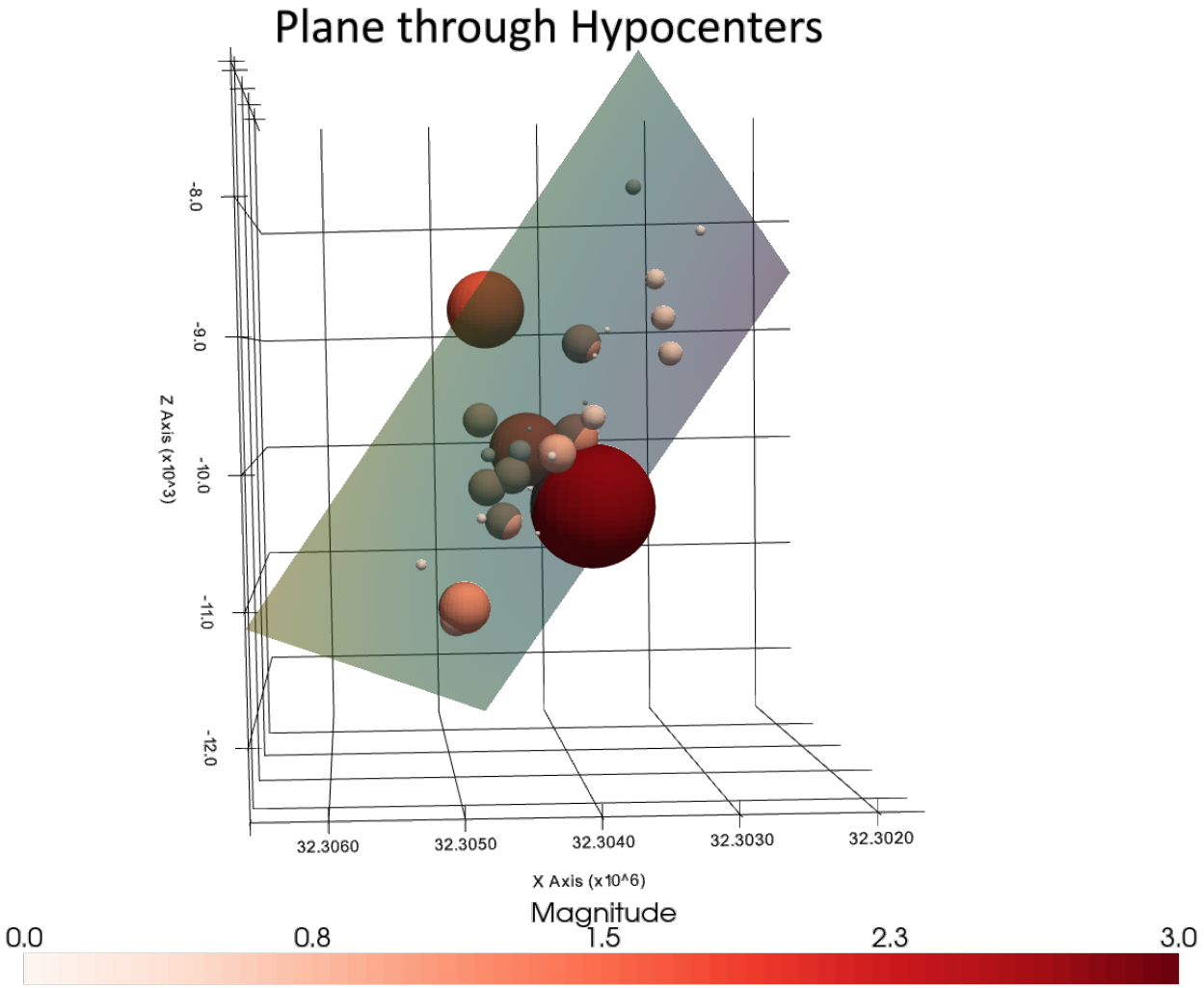
6.44.5 Plotting the LineStrings

[9]:
```
import matplotlib.pyplot as plt

gdf_linestrings.plot(column = 'Z', cmap = 'gist_earth')
plt.grid()
```



6.45 44 Fitting a plane through earthquake hypocenters

A plane can be put through a set of hypocenters of earthquakes to see if they would coincide with a single fault plane.



6.45.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/44_fitting_plane_through_earthquake_hypocenters/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="44_fitting_plane_through_
↪earthquake_hypocenters.zip", dirpath=file_path)
```

```
Downloading file '44_fitting_plane_through_earthquake_hypocenters.zip' from 'https://
↪rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F44_fitting_plane_through_
↪earthquake_hypocenters.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_
↪started\tutorial\data\44_fitting_plane_through_earthquake_hypocenters'.
```

6.45.2 Loading Earthquake Data

The data from earthquakes in the beginning of January 2021 is used.

[2]:
```
import geopandas as gpd

gdf = gpd.read_file(filename= file_path+'earthquake_data.shp')
gdf['Z'] = gdf['Tiefe [km]']*(-1000)
gdf
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:

	Datum	Uhrzeit (U	Breite [°	Länge [°	Tiefe [km]	Magnitude	\
0	08.01.2021	02:56:50.400	50.69	6.23	10.00	1.60	
1	06.01.2021	23:59:43.400	50.70	6.22	9.00	0.10	
2	06.01.2021	18:43:18.900	50.69	6.23	8.80	1.60	
3	06.01.2021	12:03:38.100	50.70	6.22	9.20	0.10	
4	06.01.2021	11:18:11.500	50.69	6.22	9.10	0.80	
5	06.01.2021	04:14:19.000	50.69	6.24	11.30	1.10	
6	05.01.2021	04:26:56.800	50.70	6.24	9.70	0.70	
7	04.01.2021	05:57:23.900	50.69	6.23	10.00	0.50	
8	03.01.2021	20:49:33.700	50.70	6.24	10.50	0.20	
9	03.01.2021	15:01:48.500	50.70	6.22	7.90	0.30	

(continues on next page)

(continued from previous page)

```
10  03.01.2021  06:25:50.100      50.70      6.23       10.00       0.20
11  03.01.2021  06:04:51.700      50.69      6.23       10.20       0.80
12  03.01.2021  05:41:54.100      50.69      6.23       10.00       0.80
13  03.01.2021  03:03:59.000      50.69      6.23       10.30       0.10
14  03.01.2021  02:41:14.200      50.69      6.23       10.30       0.80
15  03.01.2021  00:47:14.600      50.69      6.23       10.00       0.30
16  03.01.2021  00:13:27.800      50.70      6.24       10.80       0.20
17  02.01.2021  14:57:37.700      50.69      6.23       10.60       0.80
18  02.01.2021  14:46:34.300      50.69      6.21        8.20       0.20
19  02.01.2021  14:33:58.500      50.69      6.24       11.40       0.70
20  02.01.2021  14:05:37.600      50.70      6.21        8.60       0.40
21  02.01.2021  13:39:30.500      50.69      6.21        8.90       0.50
22  02.01.2021  10:19:41.400      50.70      6.22        9.70       0.50
23  02.01.2021  07:31:02.200      50.69      6.21        9.20       0.50
24  02.01.2021  06:36:29.300      50.69      6.22       10.50       2.70
25  01.01.2021  20:34:53.800      50.69      6.23        9.80       0.10
26  01.01.2021  14:26:51.000      50.69      6.22        9.90       1.00
27  01.01.2021  14:26:12.400      50.69      6.23       10.70       0.10
28  01.01.2021  14:20:20.500      50.69      6.22        9.60       0.10

         Epizentrum                          geometry         Z
0   S MULARTSHUETTE  POINT (32304265.590 5618571.115) -10000.00
1     MULARTSHUETTE  POINT (32303759.041 5620148.713)  -9000.00
2   S MULARTSHUETTE  POINT (32304643.654 5619224.937)  -8800.00
3     MULARTSHUETTE  POINT (32303817.122 5619812.545)  -9200.00
4     MULARTSHUETTE  POINT (32303871.041 5619365.205)  -9100.00
5   S MULARTSHUETTE  POINT (32304851.354 5619105.849) -11300.00
6   S MULARTSHUETTE  POINT (32304730.889 5619666.987)  -9700.00
7   S MULARTSHUETTE  POINT (32304348.701 5618901.987) -10000.00
8     MULARTSHUETTE  POINT (32304730.889 5619666.987) -10500.00
9     MULARTSHUETTE  POINT (32303547.227 5620156.674)  -7900.00
10    MULARTSHUETTE  POINT (32304170.169 5619799.301) -10000.00
11  S MULARTSHUETTE  POINT (32304419.322 5618899.344) -10200.00
12  S MULARTSHUETTE  POINT (32304082.887 5619357.259) -10000.00
13  S MULARTSHUETTE  POINT (32304286.406 5619126.976) -10300.00
14  S MULARTSHUETTE  POINT (32304631.187 5618891.420) -10300.00
15  S MULARTSHUETTE  POINT (32304647.810 5619336.109) -10000.00
16  E MULARTSHUETTE  POINT (32305250.032 5620315.575) -10800.00
17  S MULARTSHUETTE  POINT (32304485.785 5618785.530) -10600.00
18    MULARTSHUETTE  POINT (32302878.235 5619291.232)  -8200.00
19  S MULARTSHUETTE  POINT (32304921.972 5619103.213) -11400.00
20    MULARTSHUETTE  POINT (32303318.673 5619719.955)  -8600.00
21  S MULARTSHUETTE  POINT (32303152.326 5619058.241)  -8900.00
22    MULARTSHUETTE  POINT (32303808.774 5619590.201)  -9700.00
23    MULARTSHUETTE  POINT (32303077.518 5618949.729)  -9200.00
24  S MULARTSHUETTE  POINT (32303700.583 5618592.303) -10500.00
25  S MULARTSHUETTE  POINT (32304286.406 5619126.976)  -9800.00
26  S MULARTSHUETTE  POINT (32303841.835 5618587.001)  -9900.00
27  S MULARTSHUETTE  POINT (32304207.458 5618907.277) -10700.00
28  S MULARTSHUETTE  POINT (32303779.556 5618811.996)  -9600.00
```

6.45.3 Create Meshes

[3]:
```
spheres = gg.visualization.create_meshes_hypocenters(gdf=gdf)
spheres.save(file_path + 'spheres.vtm')
spheres
```

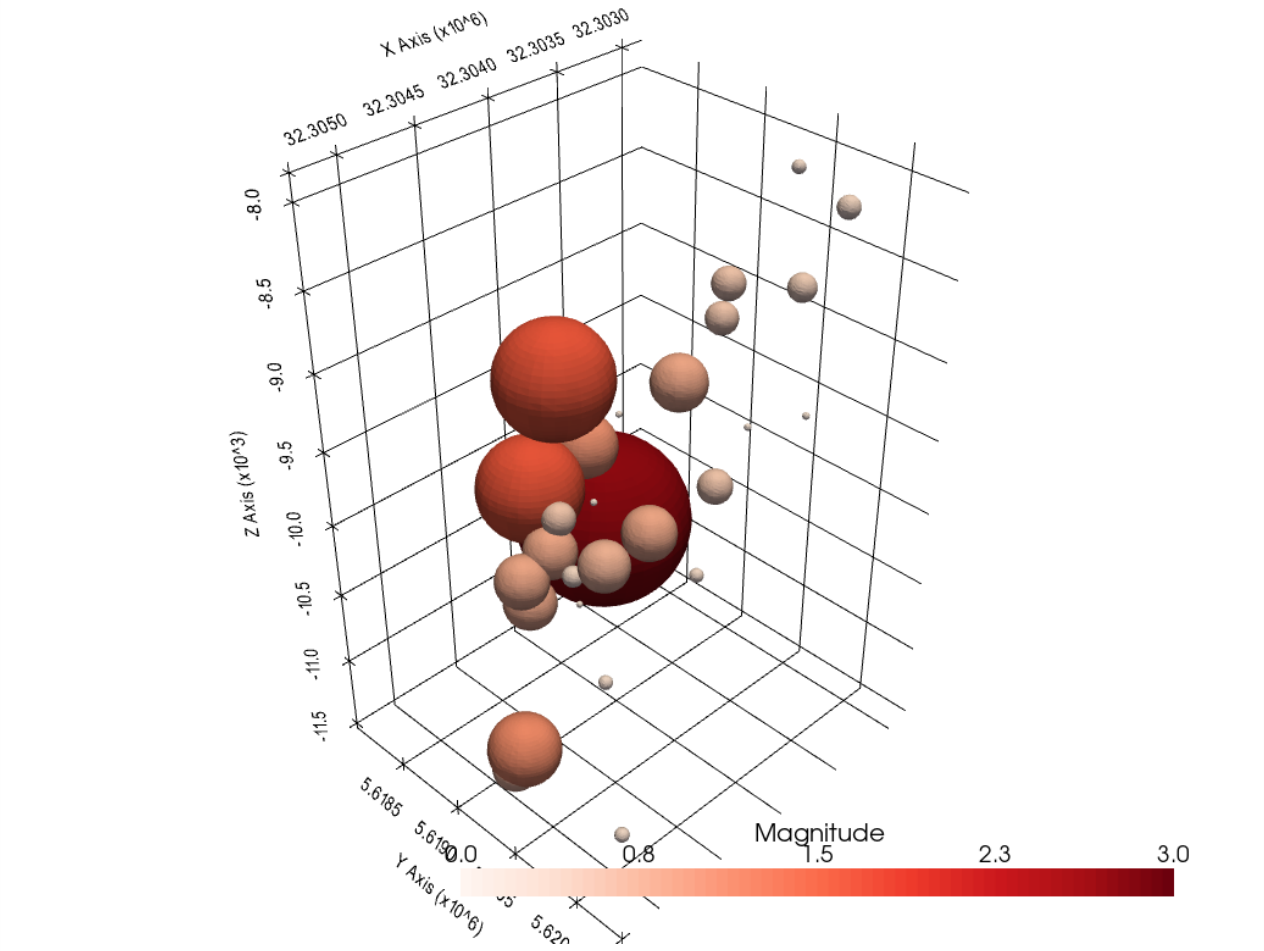
[3]:
```
MultiBlock (0x22a4e08ce20)
  N Blocks: 29
  X Bounds: 32302838.000, 32305290.000
  Y Bounds: 5618056.000, 5620355.500
  Z Bounds: -11540.000, -7840.000
```

[4]:
```
type(spheres)
```

[4]:
```
pyvista.core.composite.MultiBlock
```

6.45.4 Plot Data

[5]:
```
import pyvista as pv

sargs = dict(fmt="%.1f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(spheres,scalars='Magnitude', cmap='Reds', clim=[0,3], scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.show()
```



[6]:
```
centers = [spheres.GetBlock(block).center for block in range(spheres.
↪GetNumberOfBlocks())]
centers[:10]
```

[6]:
```
[[32304266.0, 5618571.25, -10000.0],
 [32303760.0, 5620148.75, -9000.0],
 [32304644.0, 5619224.75, -8800.0],
 [32303818.0, 5619812.5, -9200.0],
 [32303871.0, 5619365.25, -9100.0],
 [32304852.0, 5619106.0, -11300.0],
 [32304731.0, 5619667.0, -9700.0],
 [32304348.0, 5618902.0, -10000.0],
 [32304730.0, 5619667.0, -10500.0],
 [32303548.0, 5620156.75, -7900.0]]
```

[7]:
```
import numpy as np

array = np.array(centers)
array[:10]
```

[7]:
```
array([[ 3.23042660e+07,  5.61857125e+06, -1.00000000e+04],
       [ 3.23037600e+07,  5.62014875e+06, -9.00000000e+03],
       [ 3.23046440e+07,  5.61922475e+06, -8.80000000e+03],
```

(continues on next page)

(continued from previous page)

```
       [ 3.23038180e+07,  5.61981250e+06, -9.20000000e+03],
       [ 3.23038710e+07,  5.61936525e+06, -9.10000000e+03],
       [ 3.23048520e+07,  5.61910600e+06, -1.13000000e+04],
       [ 3.23047310e+07,  5.61966700e+06, -9.70000000e+03],
       [ 3.23043480e+07,  5.61890200e+06, -1.00000000e+04],
       [ 3.23047300e+07,  5.61966700e+06, -1.05000000e+04],
       [ 3.23035480e+07,  5.62015675e+06, -7.90000000e+03]])
```

[8]:
```
C = np.cov(array, rowvar=False)
eig, eiv = np.linalg.eigh(C)
normal = eiv[:, 0]
normal
```

[8]:
```
array([-0.70904181,  0.49482888, -0.50239834])
```

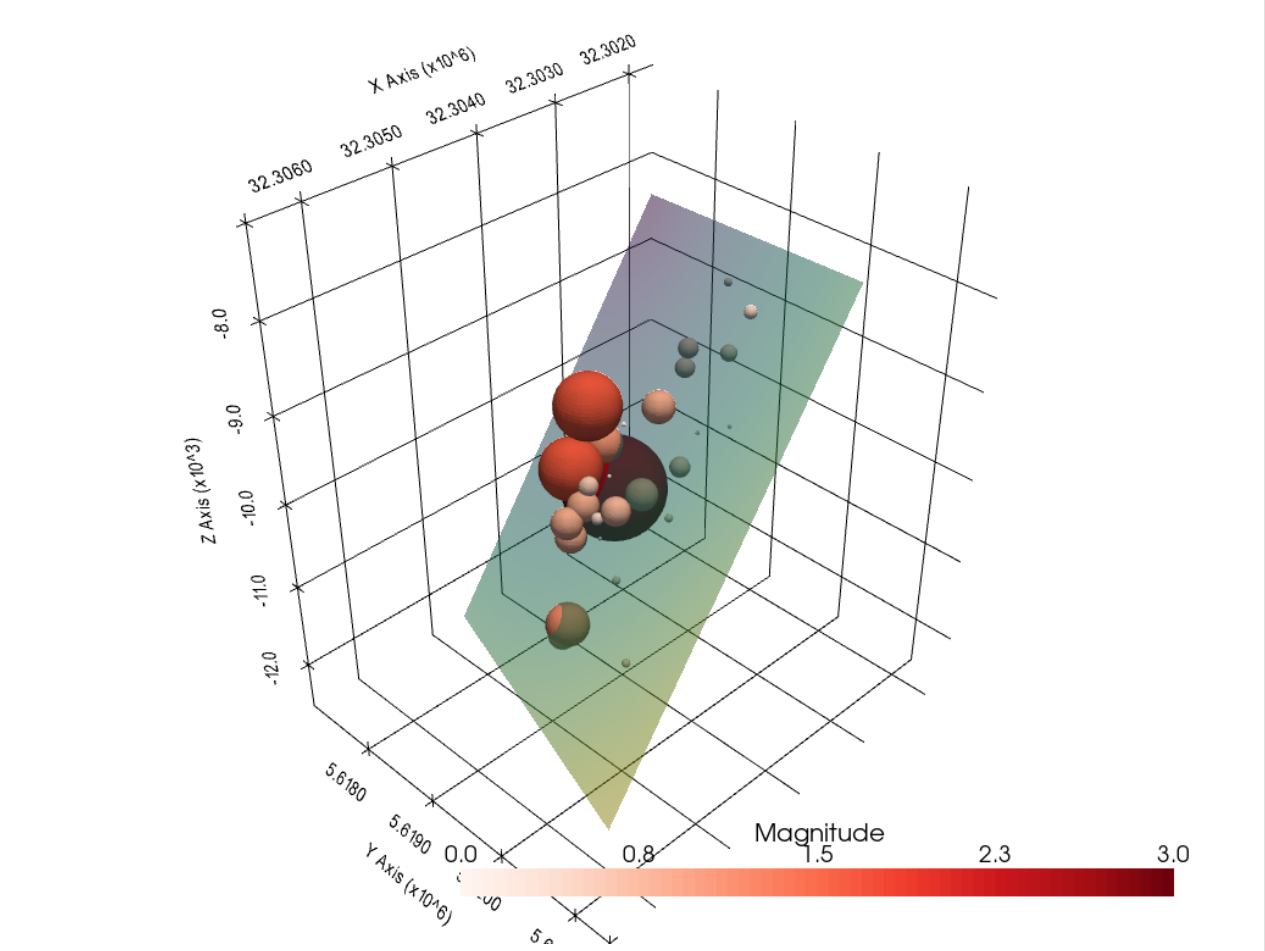
[9]:
```
center = [array[:, 0].mean(),array[:, 1].mean(),array[:, 2].mean()]
center
```

[9]:
```
[32304121.20689655, 5619271.405172414, -9800.0]
```

[10]:
```
plane = pv.Plane(center=center, direction=normal, i_size=5000, j_size=5000)
plane
```

[10]:
```
PolyData (0x22a5429da00)
  N Cells:  100
  N Points: 121
  X Bounds: 3.230e+07, 3.231e+07
  Y Bounds: 5.617e+06, 5.621e+06
  Z Bounds: -1.256e+04, -7.045e+03
  N Arrays: 2
```

[11]:
```
import pyvista as pv

sargs = dict(fmt="%.1f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(spheres,scalars='Magnitude', cmap='Reds', clim=[0,3], scalar_bar_args=sargs)
p.add_mesh(plane, opacity=0.5, show_scalar_bar=False)

p.set_background('white')
p.show_grid(color='black')
p.show()
```



The functionality shown above is also implemented in the function `plane_through_hypocenters(..)`.

[12]:
```
plane = gg.visualization.plane_through_hypocenters(spheres=spheres)
plane
```

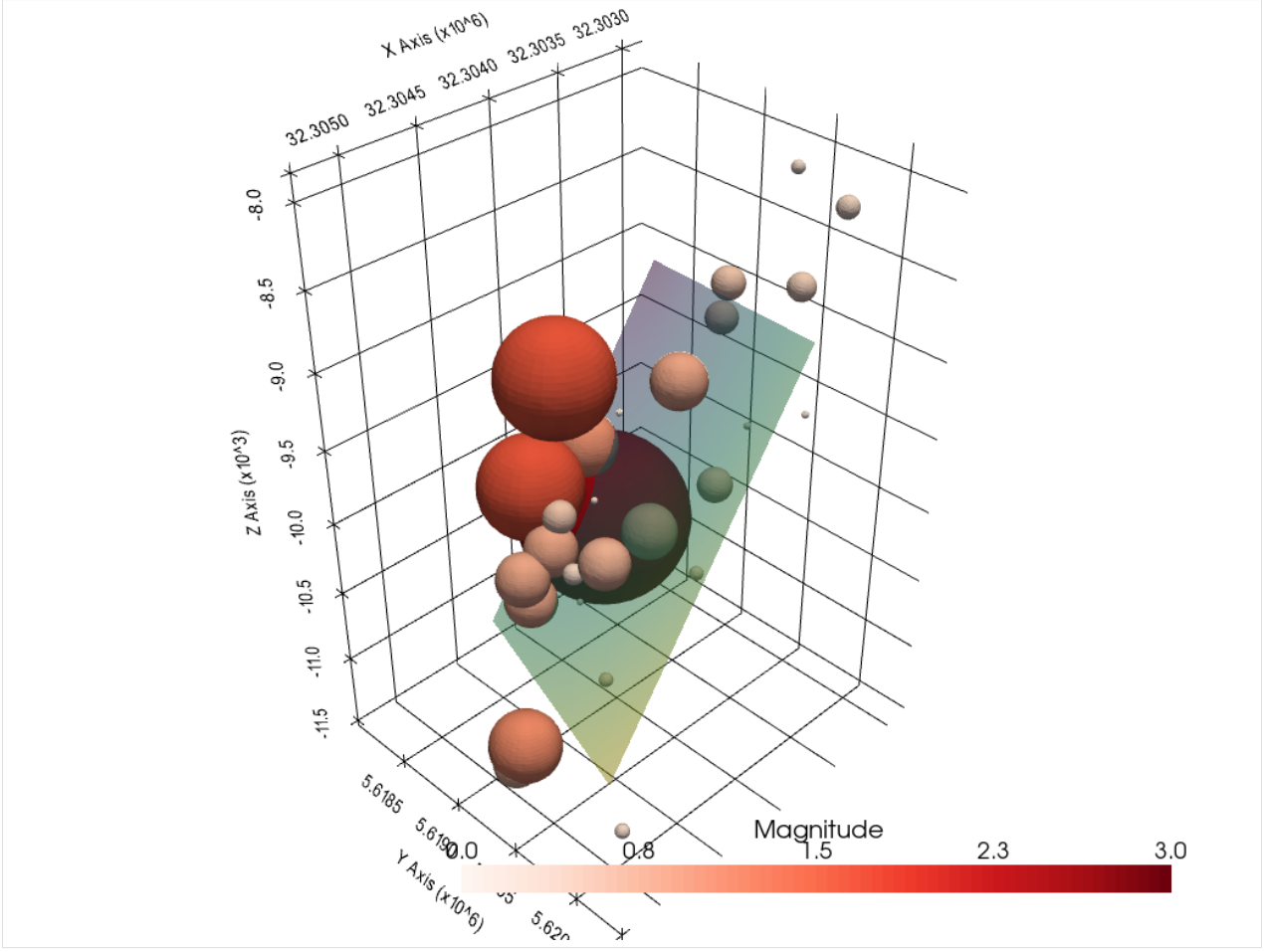
[12]:
```
PolyData (0x22a4f16cee0)
  N Cells:  100
  N Points: 121
  X Bounds: 3.230e+07, 3.231e+07
  Y Bounds: 5.618e+06, 5.620e+06
  Z Bounds: -1.113e+04, -8.471e+03
  N Arrays: 2
```

[13]:
```
import pyvista as pv

sargs = dict(fmt="%.1f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(spheres,scalars='Magnitude', cmap='Reds', clim=[0,3], scalar_bar_args=sargs)
p.add_mesh(plane, opacity=0.5, show_scalar_bar=False)

p.set_background('white')
```

(continues on next page)

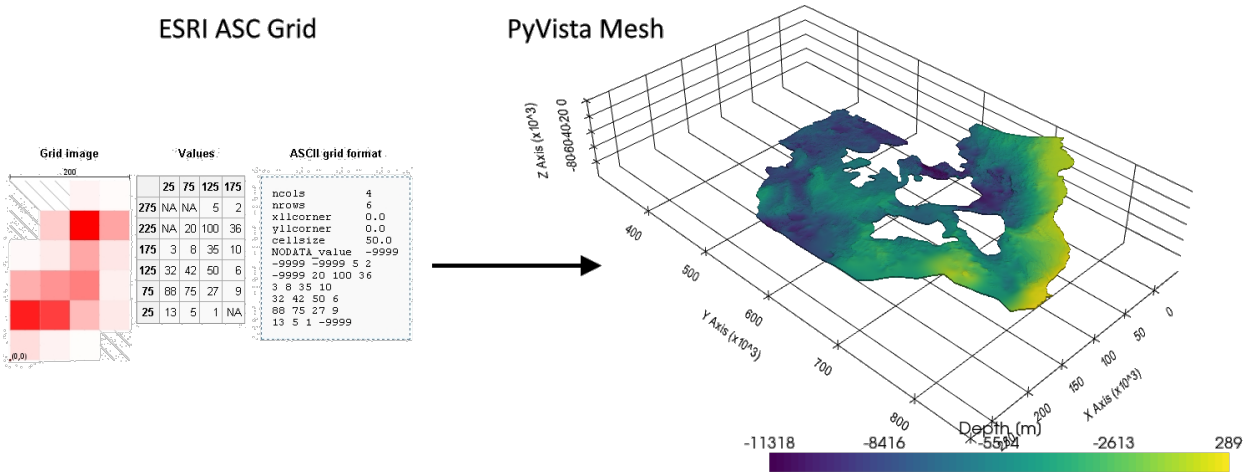
(continued from previous page)

```
p.show_grid(color='black')
p.show()
```



6.46 45 Opening ESRI ASC Grids and ZMAP Grids

GemGIS can also read ArcGIS ASC files and ZMAP Grids. The data examples shown below were obtained from https://www.nlog.nl/en/scan-2d-seismic-interpretation-and-depth-conversion-dinantian.



ESRI Grids Reference: https://desktop.arcgis.com/en/arcmap/10.3/manage-data/raster-and-images/esri-grid-format.htm

6.46.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/45_opening_asc_and_zmap_grids/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="45_opening_asc_and_zmap_grids.
↪zip", dirpath=file_path)
```

```
Downloading file '45_opening_asc_and_zmap_grids.zip' from 'https://rwth-aachen.sciebo.de/
↪s/AfXRsZywYDbUF34/download?path=%2F45_opening_asc_and_zmap_grids.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\tutorial\data\45_opening_asc_and_zmap_
↪grids'.
```

6.46.2 Loading the ASC Grid

When loading the ASC Grid, it will be returned as dict containing the array data, the extent, resolution and nodata_val. A CRS is not provided for the grid. The retrieved data can now be used and saved as tif using `save_as_tif(..)` or it can be visualized in PyVista as shown below.

[2]:
```
data = gg.raster.read_asc(path= file_path + 'top_dinant_final_tvd.asc')
data
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
{'Data': array([[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]),
 'Extent': [-42250.0, 279000.0, 306000.0, 867000.0],
 'Resolution': 250.0,
 'Nodata_val': nan}
```

[3]:
```
data['Data']
```

[3]:
```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]])
```

The extent is defined as xmin, xmax, ymin, ymax.

[4]:
```
data['Extent']
```

[4]:
```
[-42250.0, 279000.0, 306000.0, 867000.0]
```

[5]:
```
data['Resolution']
```

[5]:
```
250.0
```

[6]:
```
data['Nodata_val']
```

[6]:
```
nan
```

[7]:
```
import numpy as np
data['Data'][data['Data'] == data['Nodata_val']] = 500
data['Data']
```

[7]:
```
array([[nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       ...,
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan],
       [nan, nan, nan, ..., nan, nan, nan]])
```

[8]:
```
data['Nodata_val'] = 500
```

Visualization in PyVista

The data can be visualized using a Structured Grid. Therefore, the function `create_structured_grid_from_asc(...)` can be used.

[9]:
```
grid = gg.visualization.create_structured_grid_from_asc(data=data)
grid
```

[9]:
```
StructuredGrid (0x246685ec640)
  N Cells:  2880012
  N Points: 2883540
  X Bounds: -4.225e+04, 2.788e+05
  Y Bounds: 3.060e+05, 8.668e+05
  Z Bounds: -1.132e+04, 2.887e+02
  Dimensions:       2244, 1285, 1
  N Arrays: 1
```

[10]:
```
grid.save(file_path + 'top_dinant_final_tvd.vtk')
```

Creating Contour Lines

import numpy as np contours = grid.contour(isosurfaces=np.arange(-11000, 0, 250)) contours

contours.save(file$_path + 'top_dinant_final_tvd_contours.vtk')$

[11]:
```
import pyvista as pv
contours = pv.read(file_path+'top_dinant_final_tvd_contours.vtk')
```

contours$_shape = gg.vector.create_linestrings_from_contours(contours = contours, crs = 'EPSG : 28992')contours_shape.head()$

contours$_shape.to_file(file_path + 'top_dinant_final_tvd_contours.shp')$

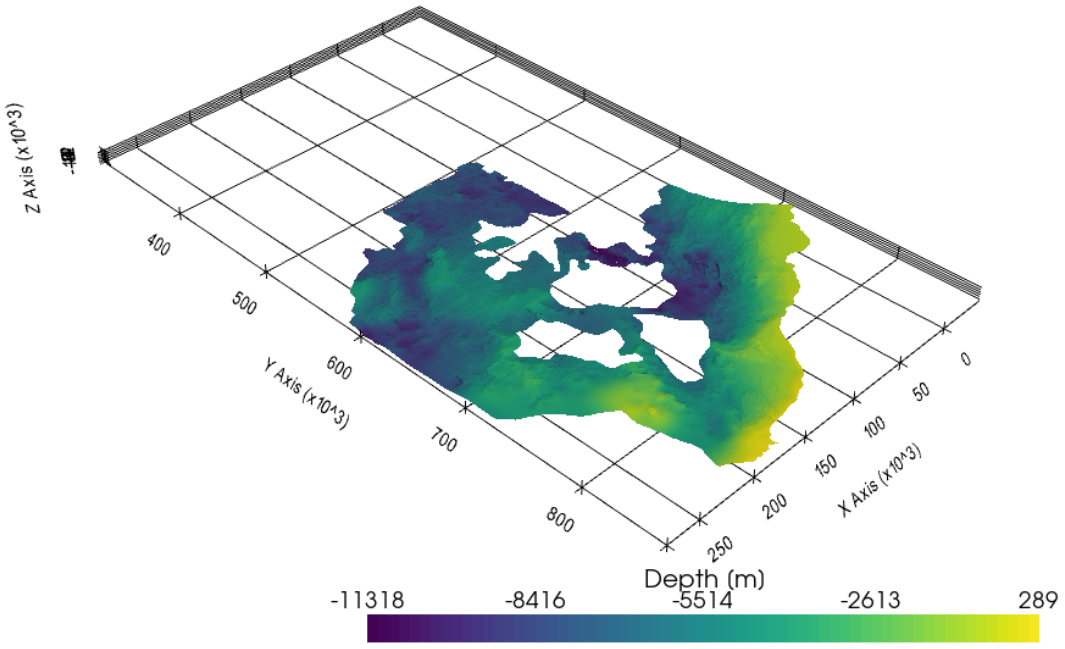
import geopandas as gpd contours$_shape = gpd.read_file(filename = file_path + 'top_dinant_final_tvd_contours.shp')contours_shape.head()$

[12]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
```

(continues on next page)

(continued from previous page)

```
p.add_mesh(grid, scalars='Depth [m]', nan_opacity=0, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.46.3 Loading the ZMAP Grid

The same dataset as before is also provided as ZMAP Grid.

[13]:
```
data = gg.raster.read_zmap(file_path + 'top_dinant_final_tvd.dat')
data
```

[13]:
```
{'Data': array([[nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        ...,
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan],
        [nan, nan, nan, ..., nan, nan, nan]]),
```

(continues on next page)

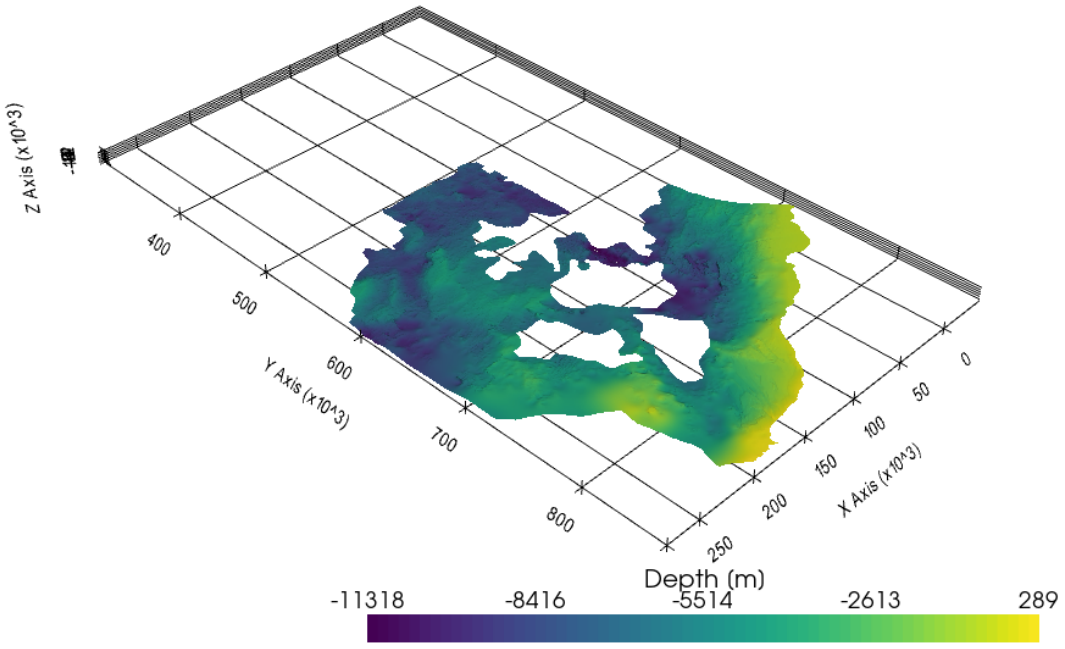
(continued from previous page)

```
 'Extent': [-42250.0, 278750.0, 306000.0, 866750.0],
 'Resolution': [250.0, 250.0],
 'Nodata_val': 1e+30,
 'Dimensions': (2244, 1285),
 'CRS': 'Amersfoort * EPSG-Nld / RD New [28992,1672]',
 'Creation_date': '21/10/2019',
 'Creation_time': '16',
 'File_name': 'TOP_DINANTIAN_TVD_final'}
```

[14]:
```
grid = gg.visualization.create_structured_grid_from_zmap(data=data)
grid
```

[14]:
```
StructuredGrid (0x2466aef7700)
  N Cells:  2880012
  N Points: 2883540
  X Bounds: -4.225e+04, 2.788e+05
  Y Bounds: 3.060e+05, 8.668e+05
  Z Bounds: -1.132e+04, 2.887e+02
  Dimensions:       2244, 1285, 1
  N Arrays: 1
```

[15]:
```
import pyvista as pv
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(grid, scalars='Depth [m]', nan_opacity=0, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



[ ]:

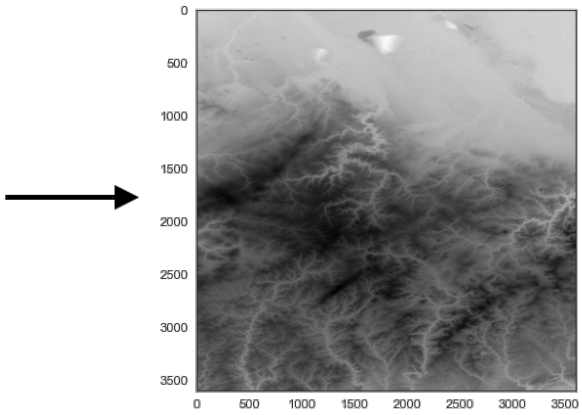
6.47 46 Working with HGT Files in GemGIS

A file with the HGT file extension is a Shuttle Radar Topography Mission (SRTM) Data file. HGT files contain digital elevation models, which are 3D pictures of a surface, usually a planet, obtained during the Shuttle Radar Topography Mission (SRTM) by NASA and the National Geospatial-Intelligence Agency (NGA).

This data can easily be opened using rasterio.

HGT File from the Shuttle Radar Topograpy Mission

HGT File opened with rasterio



Source: https://www.lifewire.com/hgt-file-2621580#:~:text=A%20file%20with%20the%20HGT,%2DIntelligence%20Agency%20(NGA).

6.47.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/46_working_with_hgt_files_in_gemgis/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="46_working_with_hgt_files_in_
↪gemgis.zip", dirpath=file_path)
```

```
Downloading file '46_working_with_hgt_files_in_gemgis.zip' from 'https://rwth-aachen.
↪sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F46_working_with_hgt_files_in_gemgis.zip'␣
↪to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\data\46_working_
↪with_hgt_files_in_gemgis'.
```

6.47.2 Load HGT Data

The HGT data can easily be loaded using Rasterio.

[2]:
```
import rasterio

raster = rasterio.open(file_path + 'N50E006.hgt')
raster
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

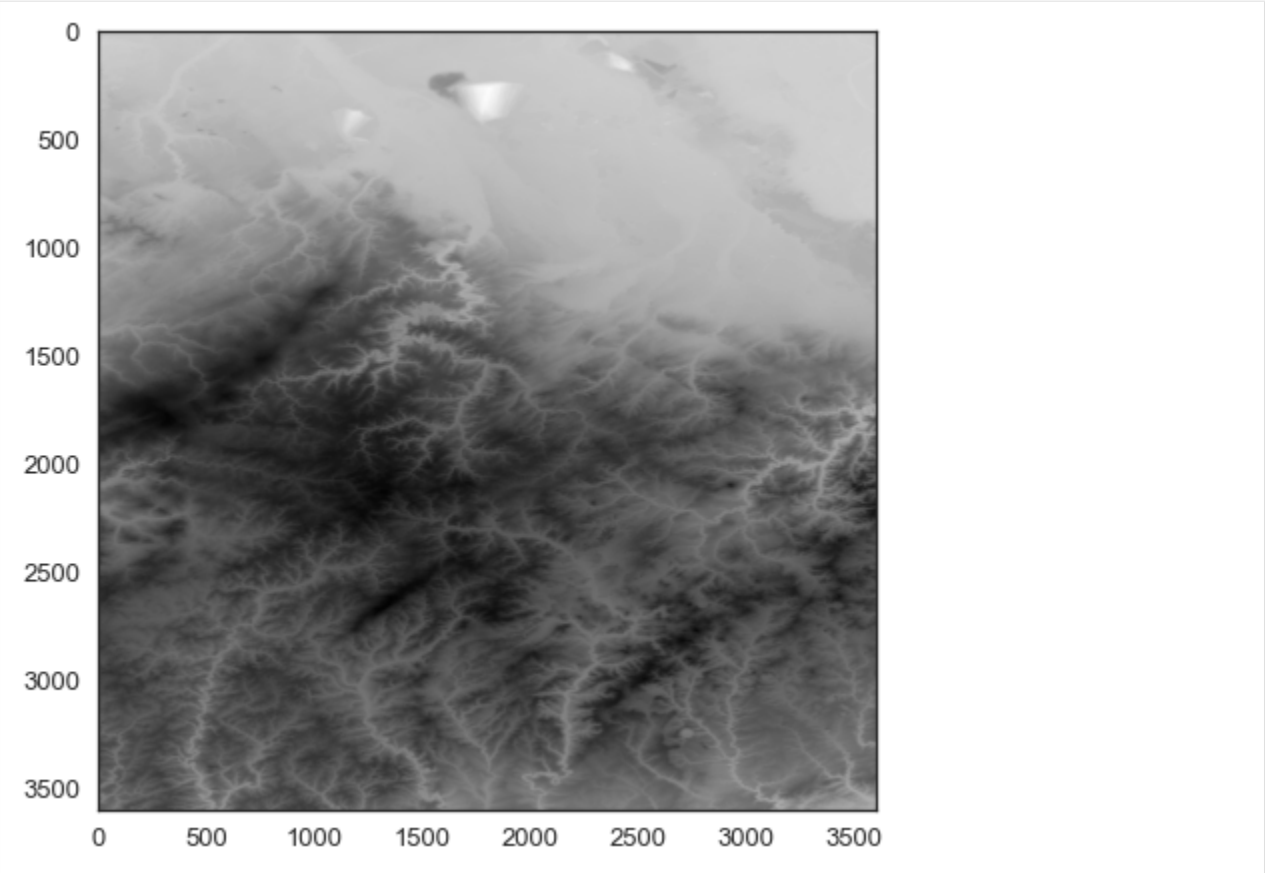
[2]:
```
<open DatasetReader name='../../../../gemgis_data/data/46_working_with_hgt_files_in_
↪gemgis/N50E006.hgt' mode='r'>
```

6.47.3 Plotting the data

The data can be plotted like rasters from tif files.

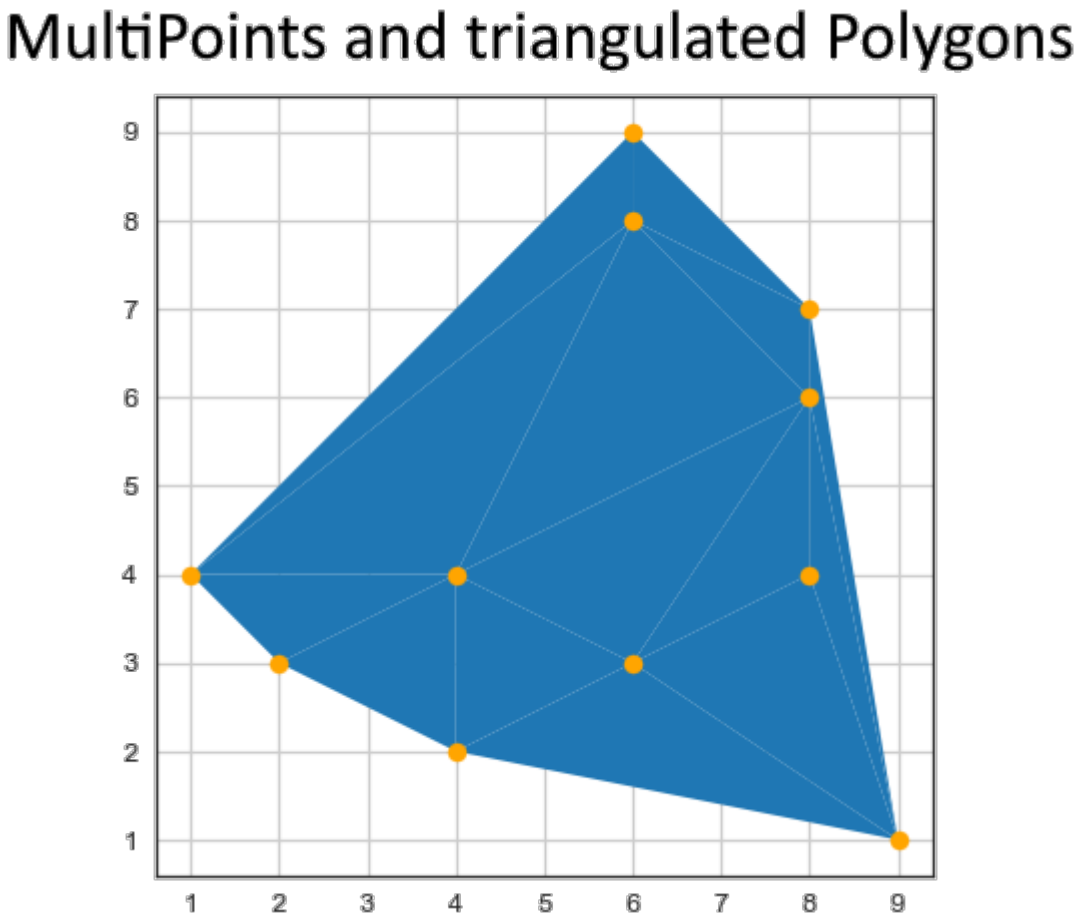
[3]:
```
import matplotlib.pyplot as plt

plt.imshow(raster.read(1))
```

[3]:
```
<matplotlib.image.AxesImage at 0x2927f990130>
```



6.48 47 Delaunay Triangulation of Shapely Multipoints

Similar to the Delaunay triangulation for GeoDataFrames to return a PolyData dataset, the triangulation can also be performed on Shapely MultiPoints.



6.48.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/47_delaunay_triangulation_of_shapely_multipoints/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
```

(continues on next page)

(continued from previous page)

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

6.48.2 Create Shapely Multipoint

A multipoint object is created from two NumPy arrays consisting of random coordinate pairs to demonstrate the delaunay triangulation of Shapely.

[2]:
```
import gemgis as gg
from shapely.geometry import MultiPoint
import numpy as np

x = np.random.randint(0,10,11)
y = np.random.randint(0,10,11)

points = np.array([x,y])

multipoints = MultiPoint(points.T)
multipoints
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:

6.48.3 Performing the Triangulation

The function `triangulate(..)` of the shapely.ops module is used to perform the triangulation. The result is a list of polygons consisting of three vertices.

[3]:
```
from shapely.ops import triangulate

triangles = triangulate(multipoints)
triangles
```

[3]:
```
[<shapely.geometry.polygon.Polygon at 0x28036efb280>,
 <shapely.geometry.polygon.Polygon at 0x28036efb190>,
 <shapely.geometry.polygon.Polygon at 0x28036efb0d0>,
 <shapely.geometry.polygon.Polygon at 0x28036efb520>,
 <shapely.geometry.polygon.Polygon at 0x28036efb580>,
 <shapely.geometry.polygon.Polygon at 0x28036efb610>,
 <shapely.geometry.polygon.Polygon at 0x28036efb640>,
 <shapely.geometry.polygon.Polygon at 0x28036efb6a0>,
 <shapely.geometry.polygon.Polygon at 0x28036efb5e0>,
```

(continues on next page)

(continued from previous page)

```
 <shapely.geometry.polygon.Polygon at 0x28036efb550>,
 <shapely.geometry.polygon.Polygon at 0x28036efb670>,
 <shapely.geometry.polygon.Polygon at 0x28036efb3d0>,
 <shapely.geometry.polygon.Polygon at 0x28036efb850>,
 <shapely.geometry.polygon.Polygon at 0x28036efb760>]
```

6.48.4 Creating GeoDataFrames

A GeoDataFrame is created from the Polygons and MultiPoints for plotting.

[4]:
```
import geopandas as gpd

gdf = gpd.GeoDataFrame(geometry=triangles)
gdf
```

[4]:
```
                                             geometry
0   POLYGON ((1.00000 4.00000, 2.00000 3.00000, 4...
1   POLYGON ((1.00000 4.00000, 4.00000 4.00000, 6...
2   POLYGON ((1.00000 4.00000, 6.00000 8.00000, 6...
3   POLYGON ((6.00000 9.00000, 6.00000 8.00000, 8...
4   POLYGON ((8.00000 7.00000, 6.00000 8.00000, 8...
5   POLYGON ((8.00000 7.00000, 8.00000 6.00000, 9...
6   POLYGON ((9.00000 1.00000, 8.00000 6.00000, 8...
7   POLYGON ((9.00000 1.00000, 8.00000 4.00000, 6...
8   POLYGON ((9.00000 1.00000, 6.00000 3.00000, 4...
9   POLYGON ((4.00000 2.00000, 6.00000 3.00000, 4...
10  POLYGON ((4.00000 2.00000, 4.00000 4.00000, 2...
11  POLYGON ((4.00000 4.00000, 6.00000 3.00000, 8...
12  POLYGON ((4.00000 4.00000, 8.00000 6.00000, 6...
13  POLYGON ((8.00000 6.00000, 6.00000 3.00000, 8...
```

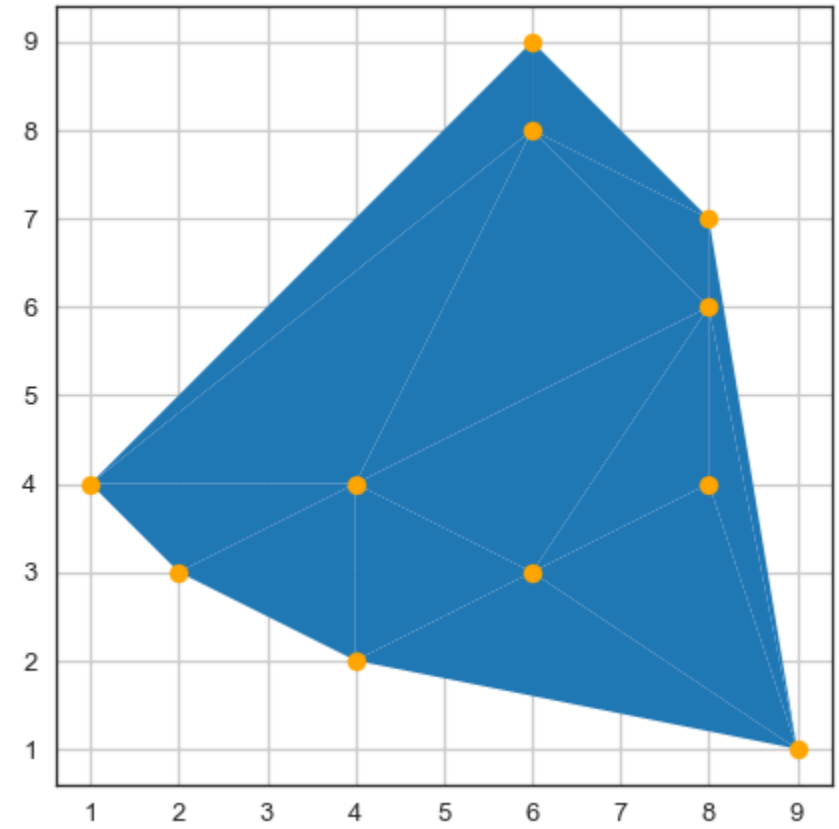
[5]:
```
gdf_points = gpd.GeoDataFrame(geometry=[multipoints])
gdf_points
```

[5]:
```
                                            geometry
0  MULTIPOINT (2.00000 3.00000, 8.00000 7.00000, ...
```

6.48.5 Plotting the Results

The result of the triangulation can be plotted.

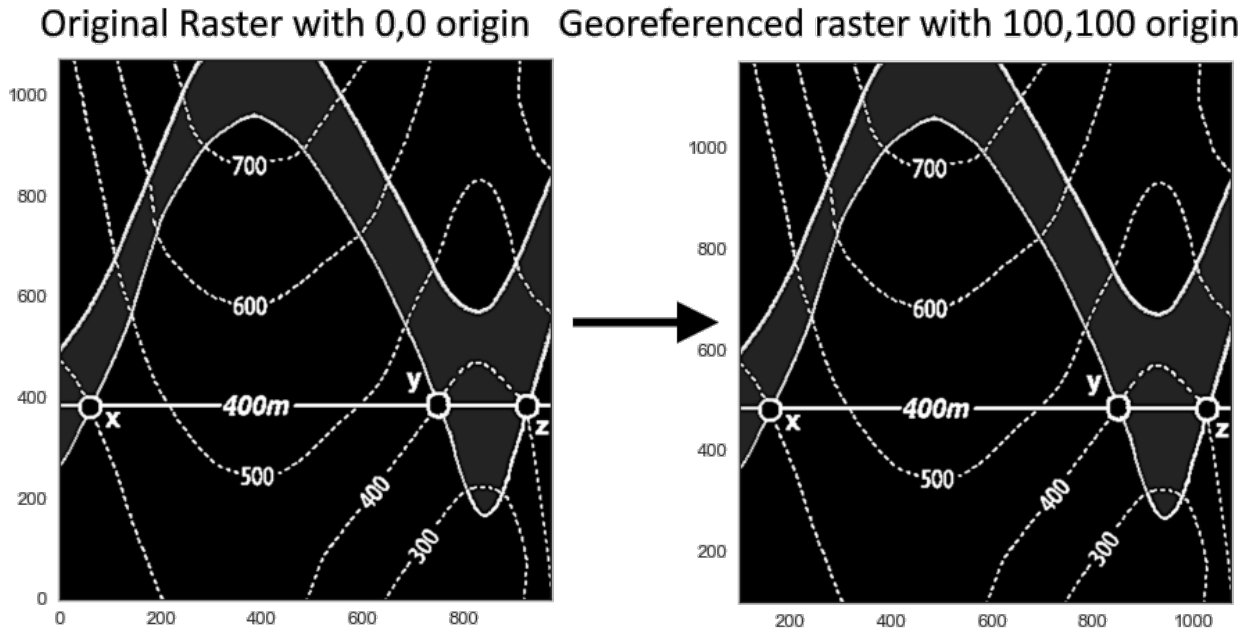
[6]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(1,1)

gdf.plot(ax=ax, aspect='equal')
gdf_points.plot(ax=ax, aspect='equal', color='orange')
plt.grid()
```



[ ]:

6.49 48 Georeferencing Rasters using Rasterio in GemGIS

Rasters with no spatial reference can be georeferenced so that they are located properly in space. This can be done using rasterio.



Source: Powell, D. (1995): Interpretation geologischer Strukturen durch Karten - Eine praktische Anleitung mit Aufgaben und Lösungen, page 15, figure 10 A, Springer Verlag Berlin, Heidelberg, New York, ISBN: 978-3-540-58607-4.

6.49.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/48_georeferencing_rasters_using_rasterio/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="48_georeferencing_rasters_using_
↪rasterio.zip", dirpath=file_path)
```

```
Downloading file '48_georeferencing_rasters_using_rasterio.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F48_georeferencing_rasters_using_
↪rasterio.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\
↪data\48_georeferencing_rasters_using_rasterio'.
```

(continues on next page)

(continued from previous page)

6.49.2 Loading Data

In order to demonstrate the capabilities of rasterio in georeferencing rasters, we are using the the raster used in the first tutorials.

[2]:
```
import rasterio
import matplotlib.pyplot as plt

raster = rasterio.open(file_path + 'task1.tif')
raster
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

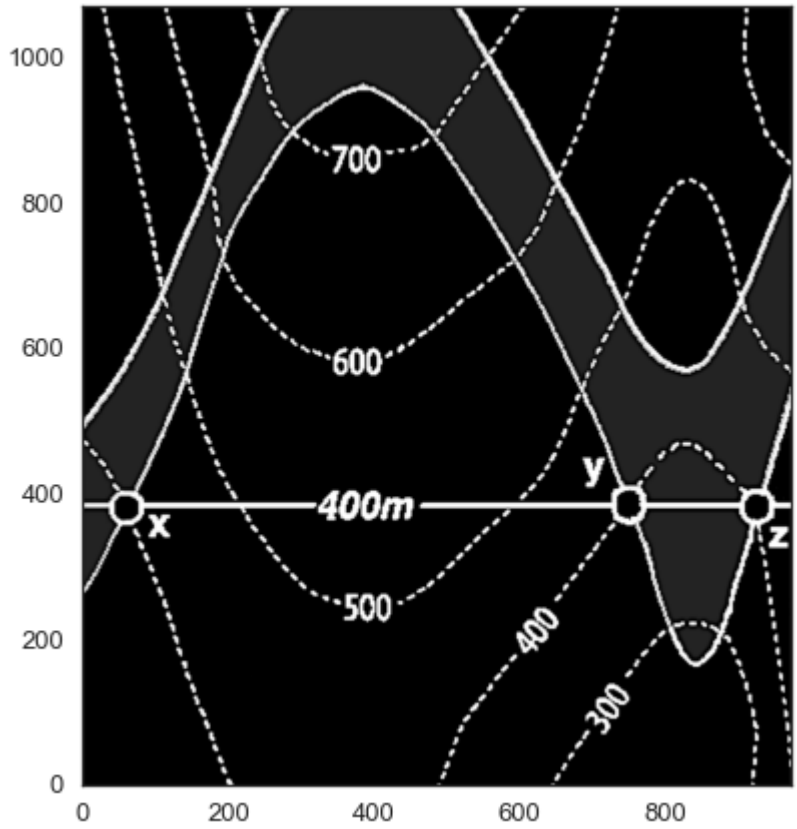
[2]:
```
<open DatasetReader name='../../../../gemgis_data/data/48_georeferencing_rasters_using_
↪rasterio/task1.tif' mode='r'>
```

[3]:
```
raster.read(1)
```

[3]:
```
array([[217, 229, 227, ..., 227, 233, 204],
       [245, 254, 254, ..., 254, 254, 246],
       [244, 254, 254, ..., 254, 254, 240],
       ...,
       [244, 254, 254, ..., 254, 254, 230],
       [245, 254, 254, ..., 254, 254, 246],
       [244, 254, 254, ..., 254, 254, 246]], dtype=uint8)
```

The origin of the raster is located at 0,0.

[4]:
```
from rasterio.plot import show

show(raster)
```



[4]:
```
<AxesSubplot:>
```

6.49.3 Defining GroundControlPoints

Ground control points are used by Rasterio to map a row and column of an image to an x and y (and z) value. We now set the origin to 100,100 and add additional 100 meters to the margins of the raster.

[5]:
```
raster.read(1).shape
```

[5]:
```
(1069, 972)
```

[6]:
```
point1 = rasterio.control.GroundControlPoint(row=0, col=0, x=100, y=1169)
point1
```

[6]:
```
GroundControlPoint(row=0, col=0, x=100, y=1169, id='8281849d-0eb4-464b-b99e-536d3d4edd56
↪')
```

[7]:
```
point2 = rasterio.control.GroundControlPoint(row=0, col=972, x=1072, y=1169)
point2
```

[7]:
```
GroundControlPoint(row=0, col=972, x=1072, y=1169, id='6ec1385c-2d33-4614-adc4-
↪07f024878aff')
```

[8]:
```
point3 = rasterio.control.GroundControlPoint(row=1069, col=0, x=100, y=100)
point3
```

[8]:
```
GroundControlPoint(row=1069, col=0, x=100, y=100, id='e3dba004-87c6-4f3b-b30e-
↪4e953d49400c')
```

[9]:
```
point4 = rasterio.control.GroundControlPoint(row=1069, col=972, x=1072, y=100)
point4
```

[9]:
```
GroundControlPoint(row=1069, col=972, x=1072, y=100, id='abd9e2e4-782c-421e-b794-
↪614ab58c74b0')
```

A list of ground control points is created

[10]:
```
points = [point1, point2, point3, point4]
points
```

[10]:
```
[GroundControlPoint(row=0, col=0, x=100, y=1169, id='8281849d-0eb4-464b-b99e-536d3d4edd56
↪'),
 GroundControlPoint(row=0, col=972, x=1072, y=1169, id='6ec1385c-2d33-4614-adc4-
↪07f024878aff'),
 GroundControlPoint(row=1069, col=0, x=100, y=100, id='e3dba004-87c6-4f3b-b30e-
↪4e953d49400c'),
 GroundControlPoint(row=1069, col=972, x=1072, y=100, id='abd9e2e4-782c-421e-b794-
↪614ab58c74b0')]
```

6.49.4 Creating Affine Transformation

An affine transformation is created. The data array can then be saved in the next step.

[11]:
```
transformation = rasterio.transform.from_gcps(points)
transformation
```

[11]:
```
Affine(1.0, 0.0, 100.0,
       0.0, -1.0000000000000002, 1169.0000000000005)
```

6.49.5 Saving the georeferenced raster

The raster is now saved as new tif file with the dimensions of the data array, the projection, the transformation and the actual data.

[12]:
```
with rasterio.open(
    file_path + 'task1_new.tif',
    'w',
    driver='GTiff',
    height=raster.read(1).shape[0],
    width=raster.read(1).shape[1],
    count=1,
    dtype=raster.read(1).dtype,
    crs='+proj=latlong',
    transform=transformation,
) as dst:
    dst.write(raster.read(1), 1)
```

6.49.6 Opening new raster and plotting

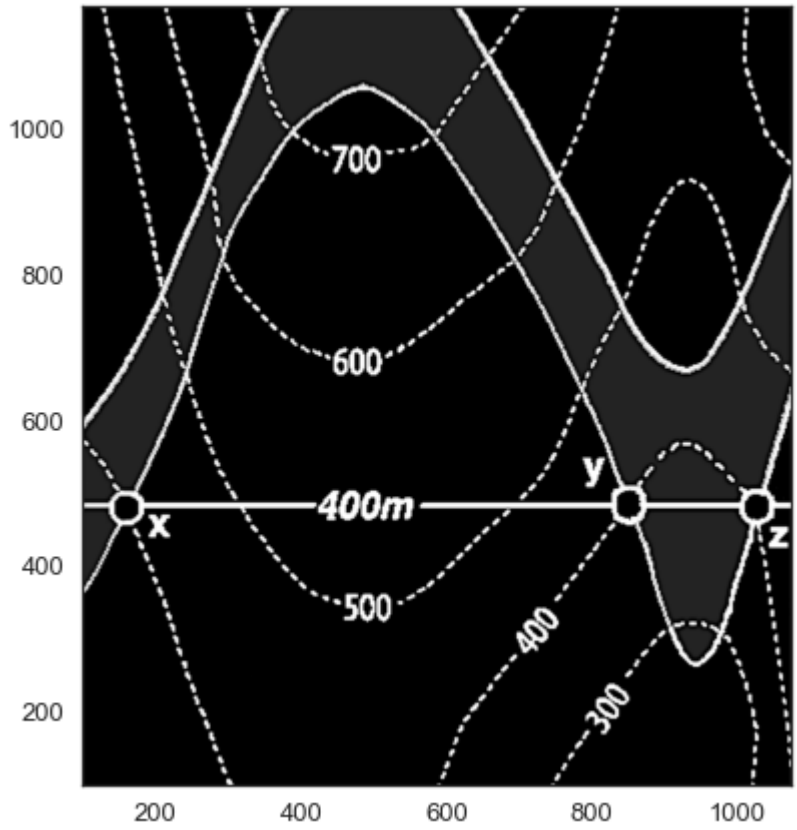
The new raster can now be opened as usual.

[13]:
```
raster_new = rasterio.open(file_path + 'task1_new.tif')
raster_new
```

[13]:
```
<open DatasetReader name='../../../../gemgis_data/data/48_georeferencing_rasters_using_
↪rasterio/task1_new.tif' mode='r'>
```

When plotting the raster, it can be seen that the origin is now correctly placed at 100,100 instead of 0,0 as for the original raster. The georeferencing was successful.

[14]:
```
from rasterio.plot import show

show(raster_new)
```

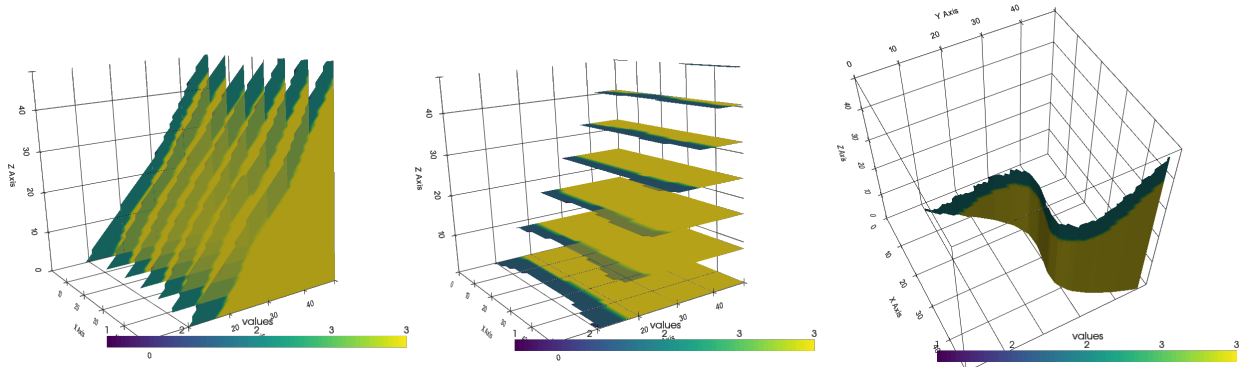


[14]:
```
<AxesSubplot:>
```

[ ]:
```

```

6.50 49 Slicing GemPy Lith Blocks in PyVista with GemGIS

The lith block of a GemPy model can be loaded as a volume into PyVista and be sliced to get geological cross sections. The different slicing options are introduced here.



6.50.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/49_slicing_gempy_lith_blocks_in_pyvista_with_gemgis/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="49_slicing_gempy_lith_blocks_in_
↪pyvista_with_gemgis.zip", dirpath=file_path)
```

```
Downloading file '49_slicing_gempy_lith_blocks_in_pyvista_with_gemgis.zip' from 'https://
↪rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F49_slicing_gempy_lith_blocks_
↪in_pyvista_with_gemgis.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_
↪started\tutorial\data\49_slicing_gempy_lith_blocks_in_pyvista_with_gemgis'.
```

6.50.2 Loading the lith block

The lith block of a computed GemPy model is loaded using NumPy and reshaped to the original dimensions (resolution) of the model.

[2]:
```
import pyvista as pv
import numpy as np

lith_block = np.load(file_path + 'lith_block.npy').reshape(50,50,50)
lith_block[0]
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [3., 3., 3., ..., 2., 2., 1.],
       [3., 3., 3., ..., 2., 2., 2.],
       [3., 3., 3., ..., 2., 2., 2.]])
```

All values above the topography representing air are assigned NaN values so that they can be hidden from the visualization.

[3]:
```
lith_block[lith_block == 1] = np.nan
lith_block[:0]
```

[3]:
```
array([], shape=(0, 50, 50), dtype=float64)
```

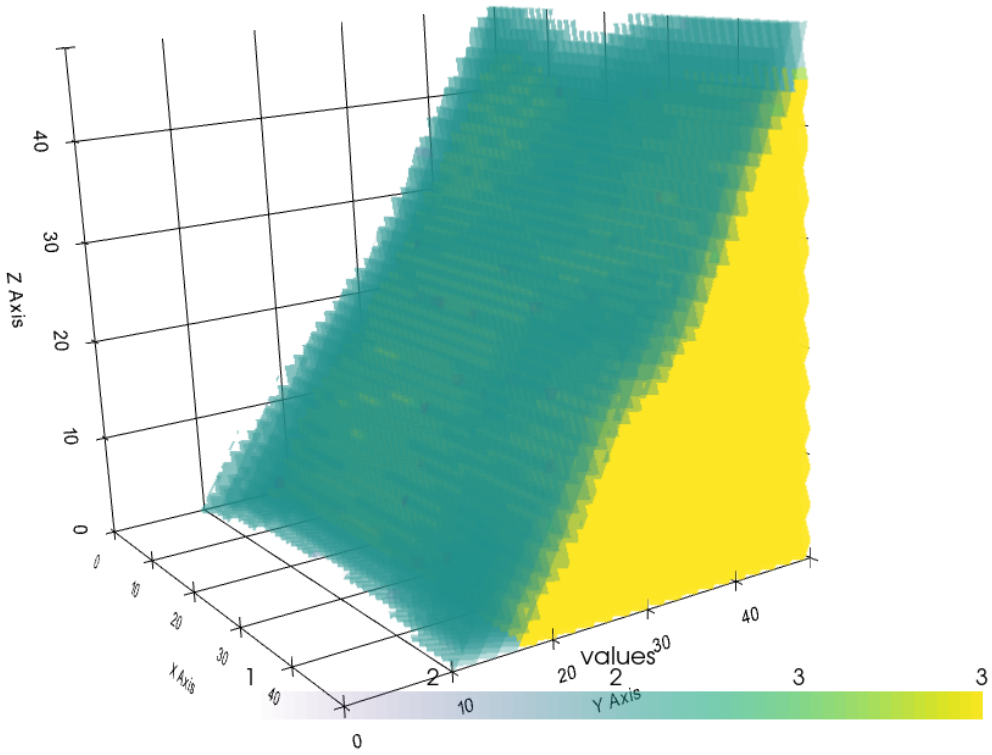
6.50.3 Wrapping the object

The array can now be wrapped using `pv.wrap(..)` to create a volume.

[4]:
```
volume = pv.wrap(lith_block)
volume
```

[4]:
```
UniformGrid (0x275e8d23ca0)
  N Cells:  117649
  N Points: 125000
  X Bounds: 0.000e+00, 4.900e+01
  Y Bounds: 0.000e+00, 4.900e+01
  Z Bounds: 0.000e+00, 4.900e+01
  Dimensions:       50, 50, 50
  Spacing:  1.000e+00, 1.000e+00, 1.000e+00
  N Arrays: 1
```

[5]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.850622003601 07), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]

p.add_volume(volume, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.50.4 Slicing the volume orthogonally

When slicing the volume, a multi block object is created where the different blocks represent the different slices.

[6]:
```
slices = volume.slice_orthogonal()
slices
```

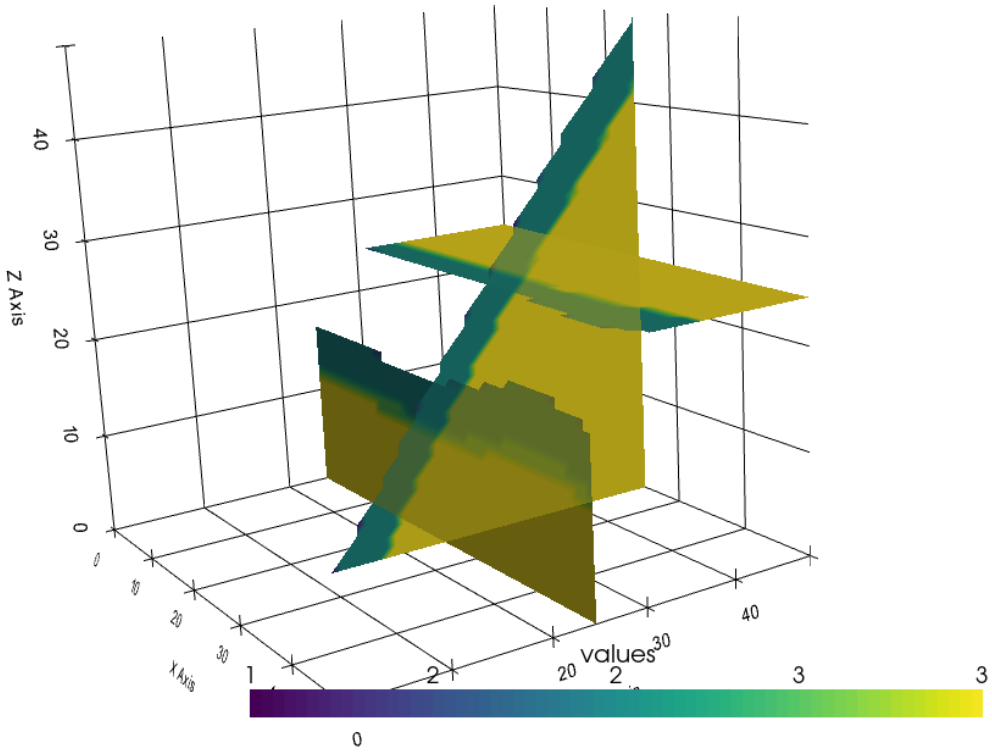
[6]:
```
MultiBlock (0x275ee300be0)
  N Blocks: 3
  X Bounds: 0.000, 49.000
```

(continues on next page)

(continued from previous page)

```
  Y Bounds: 0.000, 49.000
  Z Bounds: 0.000, 49.000
```

[7]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.85062200360107), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]

p.add_mesh(slices, scalar_bar_args=sargs, nan_opacity=0)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



[8]:
```
slices = volume.slice_orthogonal(x=5, y=20, z=5)
slices
```

[8]:
```
MultiBlock (0x275ee325040)
  N Blocks: 3
  X Bounds: 0.000, 49.000
  Y Bounds: 0.000, 49.000
```

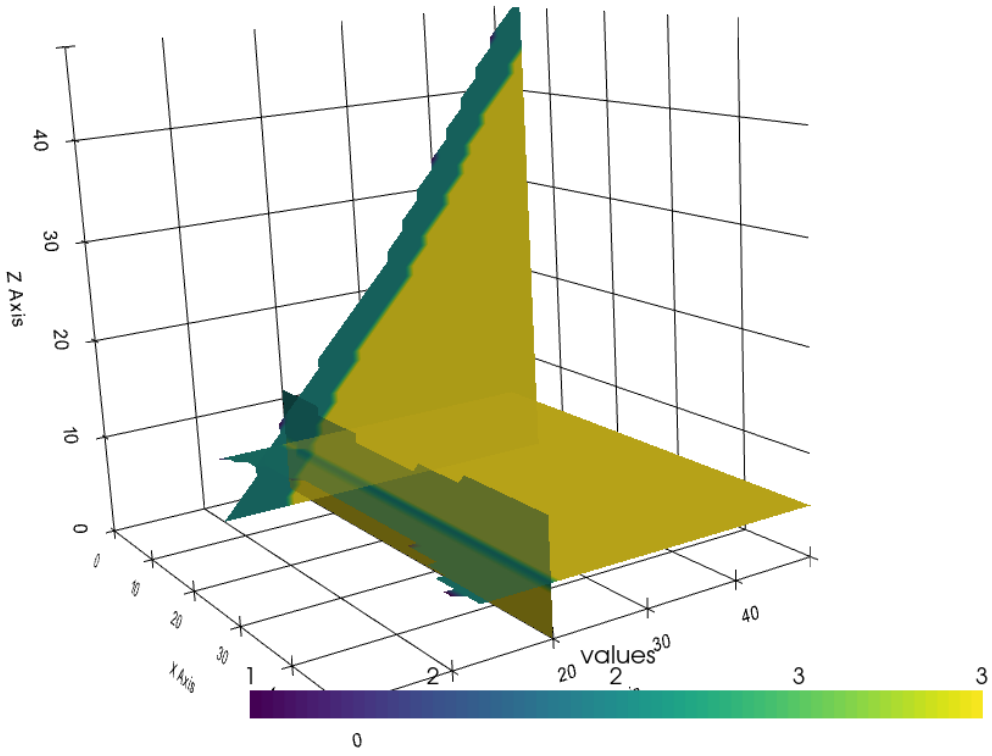
(continues on next page)

(continued from previous page)

```
  Z Bounds: 0.000, 49.000
```

The orthogonal slices can be easily translated throughout the volume.

[9]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.850622003601 07), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]

p.add_mesh(slices, scalar_bar_args=sargs, nan_opacity=0)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.50.5 Adding single slices

Arbitrary single slices can also be extracted. The origin defaults to the center of the mesh.

[10]:
```
single_slice1 = volume.slice(normal=[1, 1, 0])
single_slice1
```

[10]:
```
PolyData (0x275ee325ee0)
  N Cells:  2401
  N Points: 2500
  X Bounds: 0.000e+00, 4.900e+01
  Y Bounds: 0.000e+00, 4.900e+01
  Z Bounds: 0.000e+00, 4.900e+01
  N Arrays: 1
```

[11]:
```
single_slice2 = volume.slice(normal=[1, 0, 0])
single_slice2
```

[11]:
```
PolyData (0x275ee325fa0)
  N Cells:  2401
  N Points: 2500
  X Bounds: 2.450e+01, 2.450e+01
  Y Bounds: 0.000e+00, 4.900e+01
  Z Bounds: 0.000e+00, 4.900e+01
  N Arrays: 1
```

[12]:
```
single_slice3 = volume.slice(normal=[0, 1, 0])
single_slice3
```

[12]:
```
PolyData (0x275ea1722e0)
  N Cells:  2401
  N Points: 2500
  X Bounds: 0.000e+00, 4.900e+01
  Y Bounds: 2.450e+01, 2.450e+01
  Z Bounds: 0.000e+00, 4.900e+01
  N Arrays: 1
```

[13]:
```
single_slice4 = volume.slice(normal=[0, 0, 1])
single_slice4
```

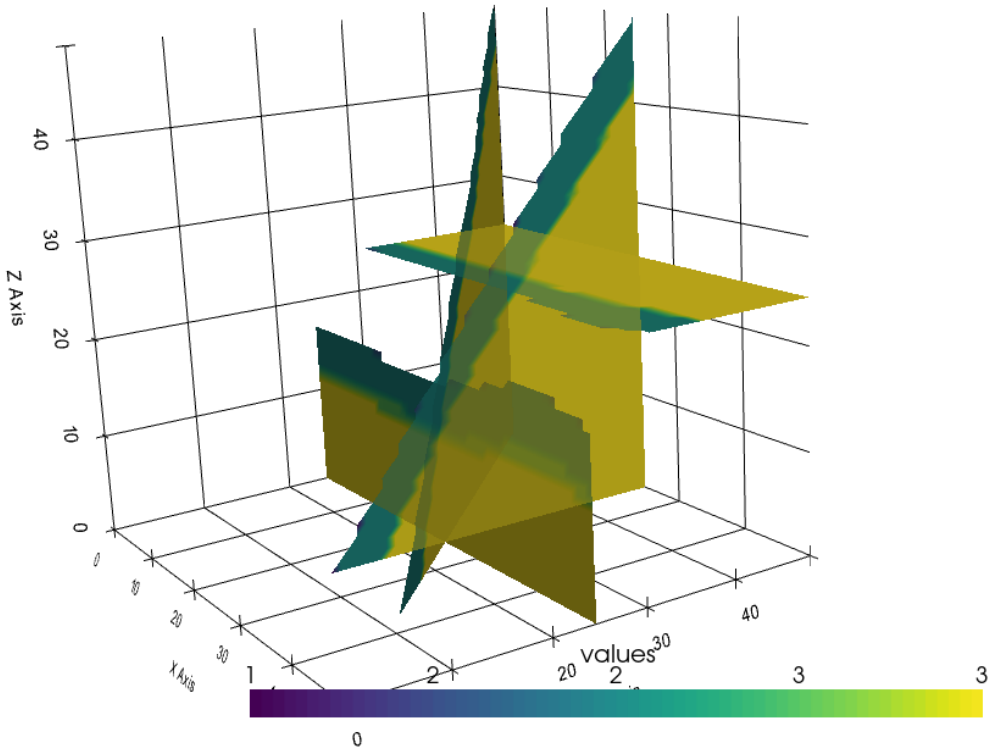
[13]:
```
PolyData (0x275ea172700)
  N Cells:  2401
  N Points: 2500
  X Bounds: 0.000e+00, 4.900e+01
  Y Bounds: 0.000e+00, 4.900e+01
  Z Bounds: 2.450e+01, 2.450e+01
  N Arrays: 1
```

[14]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.850622003601107), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]
```

(continues on next page)

(continued from previous page)

```
p.add_mesh(single_slice1, scalar_bar_args=sargs, nan_opacity=0)
p.add_mesh(single_slice2, scalar_bar_args=sargs, nan_opacity=0)
p.add_mesh(single_slice3, scalar_bar_args=sargs, nan_opacity=0)
p.add_mesh(single_slice4, scalar_bar_args=sargs, nan_opacity=0)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



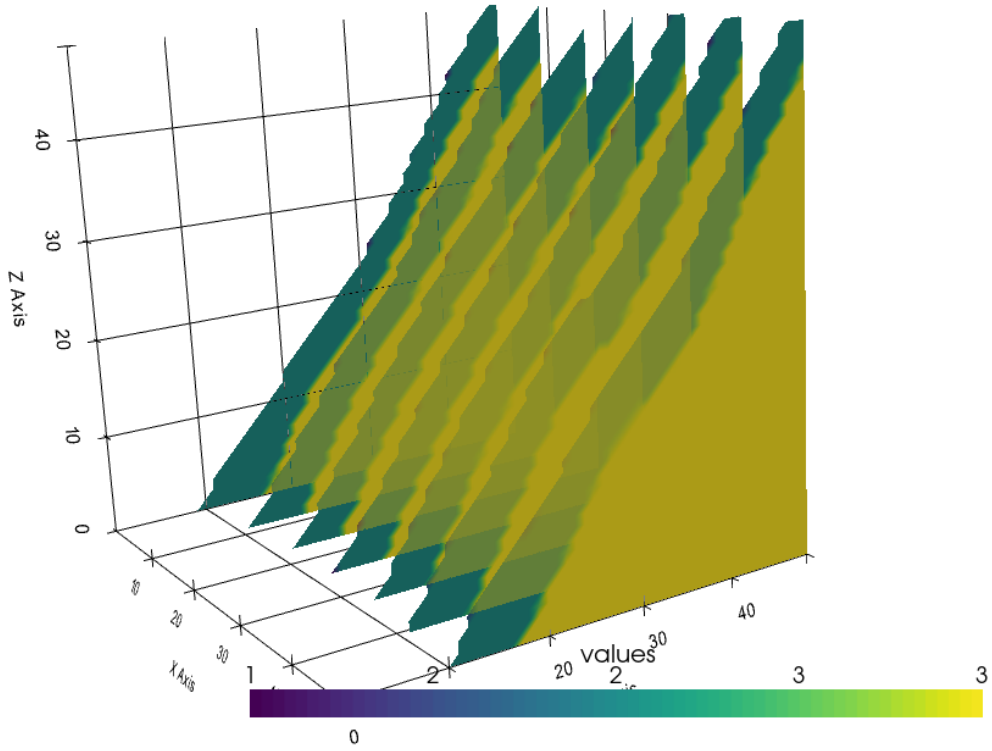
6.50.6 Slices uniformly accross an axial direction

Adding slicing planes uniformly across an axial direction can also be automated using `slice_along_axis(..)`.

[15]:
```
slices = volume.slice_along_axis(n=7, axis="x")
slices
```

[15]:
```
MultiBlock (0x275ea172460)
  N Blocks: 7
  X Bounds: 0.490, 48.510
  Y Bounds: 0.000, 49.000
  Z Bounds: 0.000, 49.000
```

[16]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.850622003601 07), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]

p.add_mesh(slices, scalar_bar_args=sargs, nan_opacity=0)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



[17]:
```
slices = volume.slice_along_axis(n=7, axis="z")
slices
```

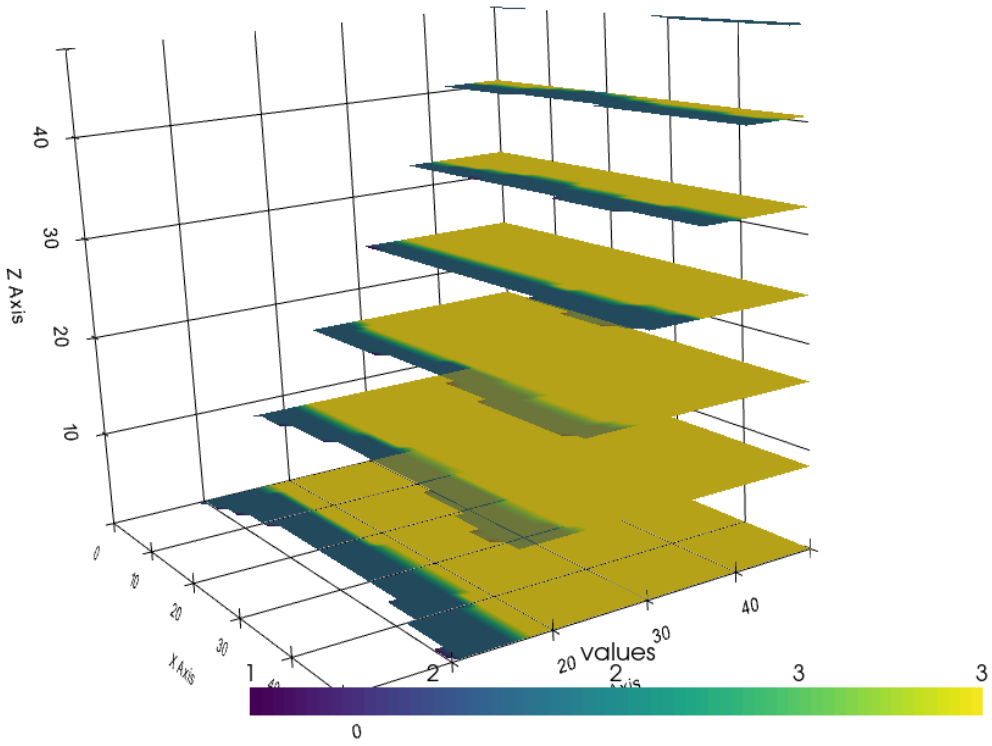
[17]:
```
MultiBlock (0x275ea18ae20)
  N Blocks: 7
  X Bounds: 0.000, 49.000
  Y Bounds: 0.000, 49.000
  Z Bounds: 0.490, 48.510
```

[18]:
```
sargs = dict(fmt="%.0f", color='black')
```

(continues on next page)

(continued from previous page)

```
p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.85062200360107), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]

p.add_mesh(slices, scalar_bar_args=sargs, nan_opacity=0)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.50.7 Cross section along a given LineString/Profile

A cross section can also be extracted along a predefined LineString, such as a Profile Line on the surface.

[19]:
```
from shapely.geometry import LineString

linestring = LineString([(10,20,0), (20,30,0), (40,30,0), (50,50,0)])
linestring
```

[19]:

[20]:
```
np.asarray(linestring)
```

[20]:
```
array([[10., 20.,  0.],
       [20., 30.,  0.],
       [40., 30.,  0.],
       [50., 50.,  0.]])
```

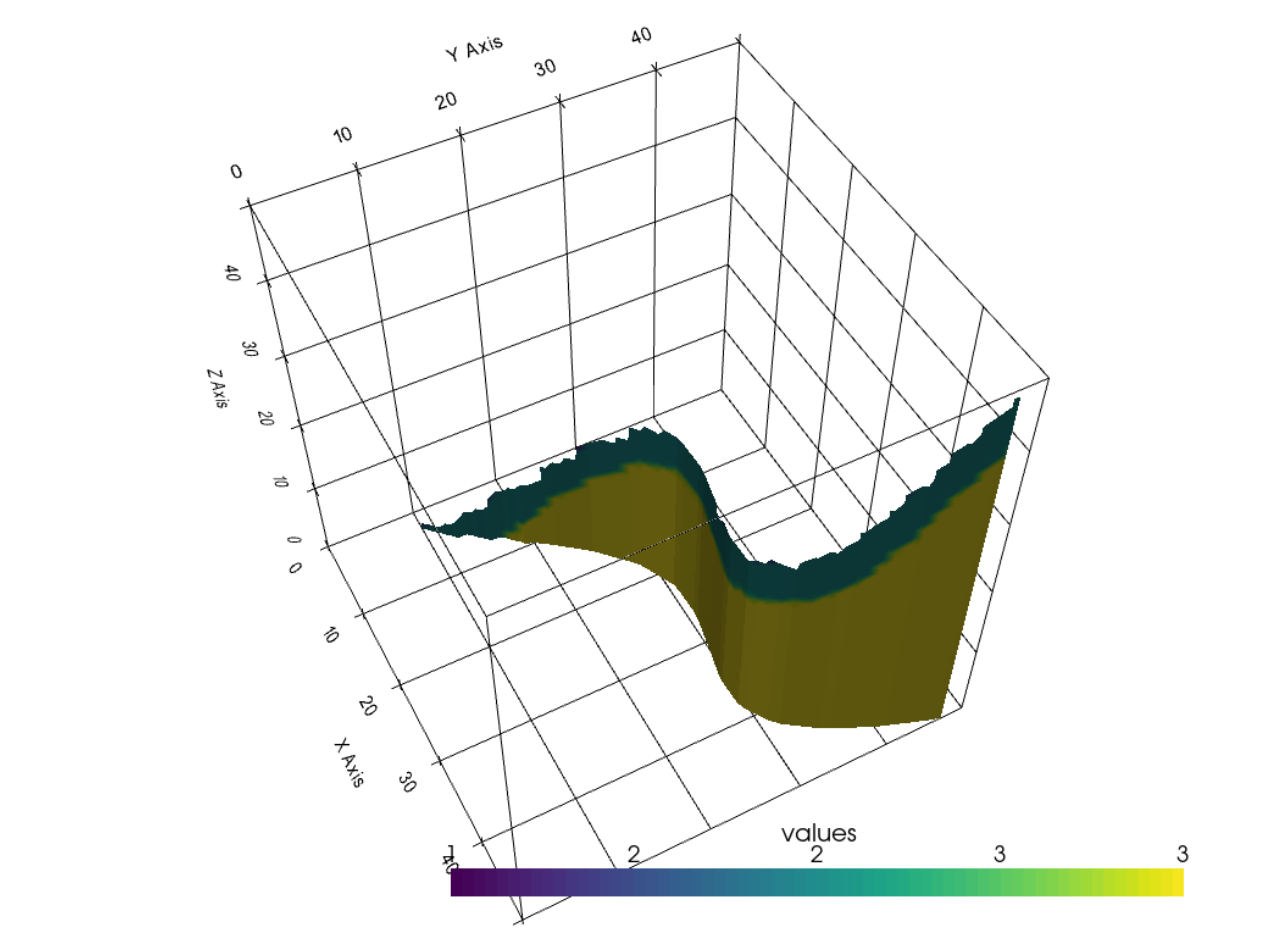
[21]:
```
spline = pv.Spline(np.asarray(linestring),15)
spline
```

[21]:
```
PolyData (0x275ea192ac0)
  N Cells:  1
  N Points: 15
  X Bounds: 1.000e+01, 5.000e+01
  Y Bounds: 2.000e+01, 5.000e+01
  Z Bounds: 0.000e+00, 0.000e+00
  N Arrays: 1
```

[22]:
```
slice = volume.slice_along_line(spline)
slice
```

[22]:
```
PolyData (0x275ee325d60)
  N Cells:  4459
  N Points: 4600
  X Bounds: 0.000e+00, 4.900e+01
  Y Bounds: 5.512e+00, 4.644e+01
  Z Bounds: 0.000e+00, 4.900e+01
  N Arrays: 1
```

[23]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(116.94895879808979, -23.54486636447751, 151.0971674252113), (24.5,␣
↪24.5, 24.5), (-0.6820986393942956, 0.36187292918718394, 0.6354442770675789)]
p.add_mesh(slice, scalar_bar_args=sargs, nan_opacity=0)

p.show_grid(color='black')
p.set_background(color='white')
p.add_mesh(volume.outline(), color="k")
p.show()
```



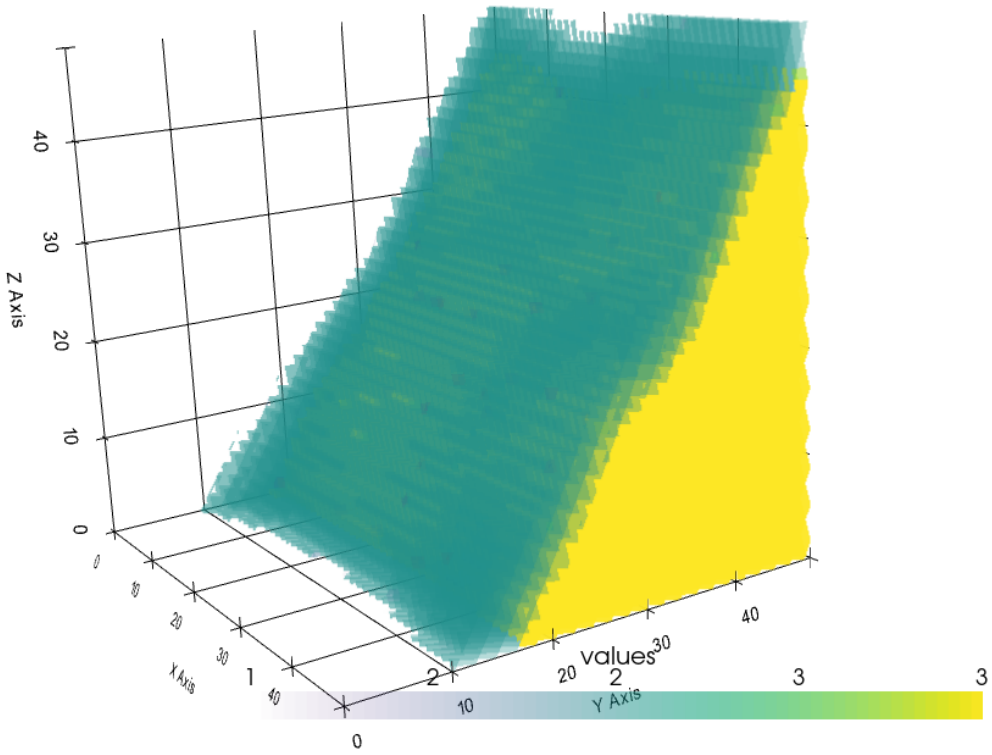
6.50.8 Translating the volume and changing cell size

As the lith block was opened as NumPy array, the spatial information of the location of the block and the size of each block was lost. By assigning new values to the `origin` and `spacing` attribute of the `volume`, the volume can be translated to its real world coordinates and the cell size can be adjusted to fit the model cell sizes. Do not interchange the number of cells with size of each cell which is calculated by the length of each model dimension divided by the number of cells. The loaded volume was located at 0,0,0 and each cell had the dimension of 1,1,1.

[24]:
```
volume
```

[24]:
```
UniformGrid (0x275e8d23ca0)
  N Cells:  117649
  N Points: 125000
  X Bounds: 0.000e+00, 4.900e+01
  Y Bounds: 0.000e+00, 4.900e+01
  Z Bounds: 0.000e+00, 4.900e+01
  Dimensions:       50, 50, 50
  Spacing:  1.000e+00, 1.000e+00, 1.000e+00
  N Arrays: 1
```

[25]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.850622003601 07), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]

p.add_volume(volume, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.50.9 Changing the origin

[26]:
```
volume.origin = [100,100,100]
volume.origin
```

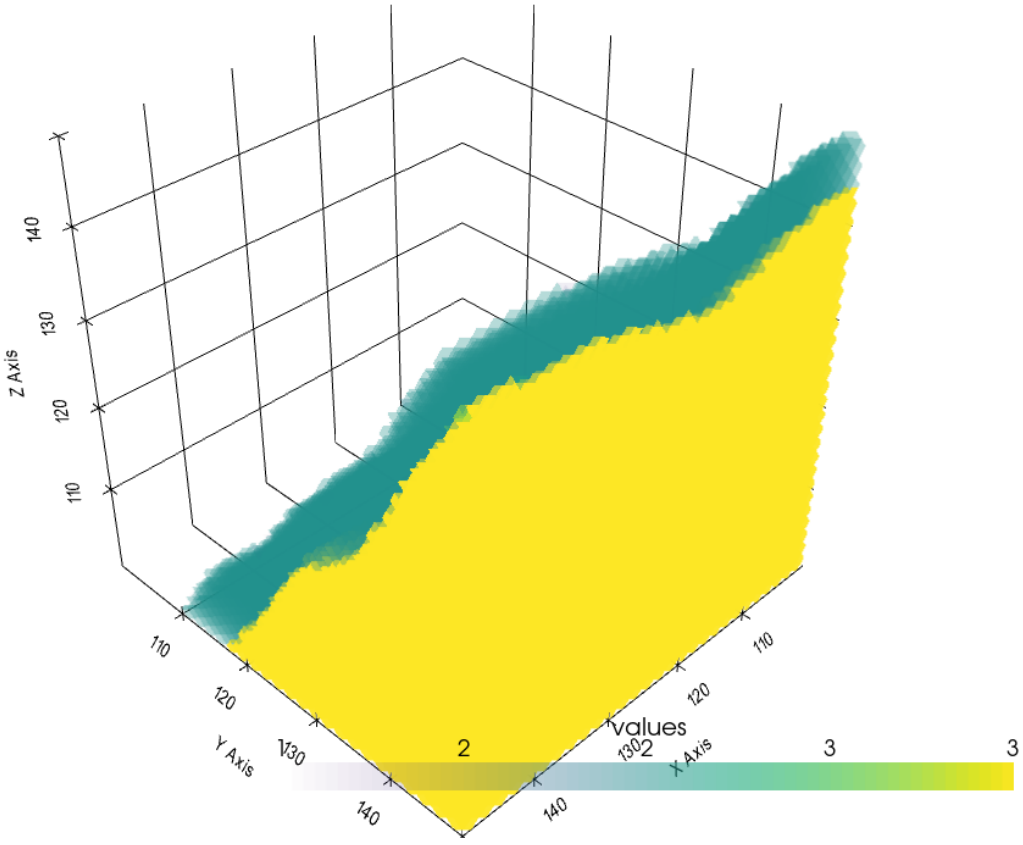
[26]:
```
[100.0, 100.0, 100.0]
```

[27]:
```
sargs = dict(fmt="%.0f", color='black')
```

(continues on next page)

(continued from previous page)

```
p = pv.Plotter(notebook=True)

p.add_volume(volume, scalar_bar_args=sargs)

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.50.10 Changing the cell size

[28]:
```
volume.spacing = [19, 21, 10]
volume.spacing
```

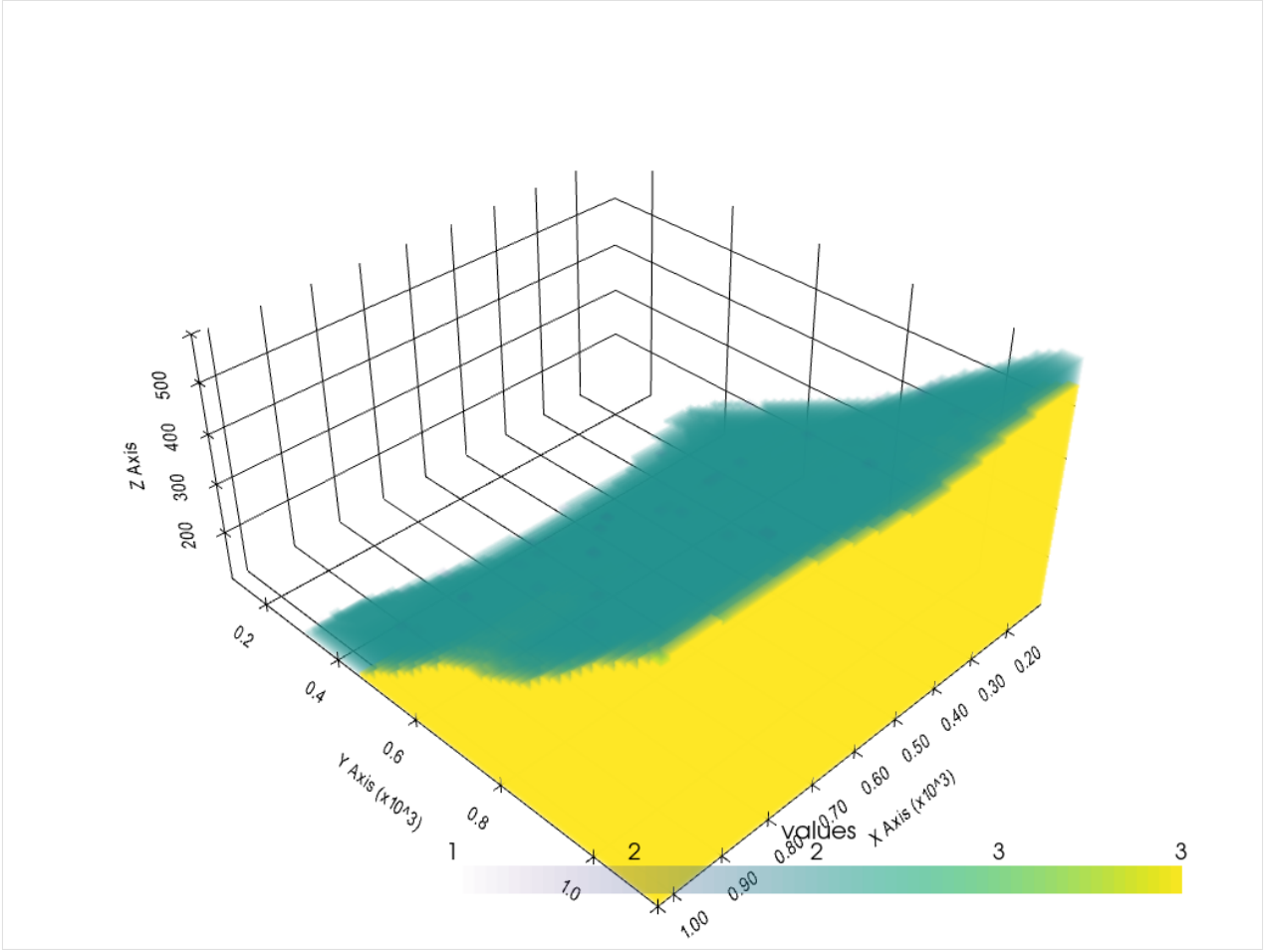
[28]:
```
[19.0, 21.0, 10.0]
```

[29]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_volume(volume, scalar_bar_args=sargs)
```

(continues on next page)

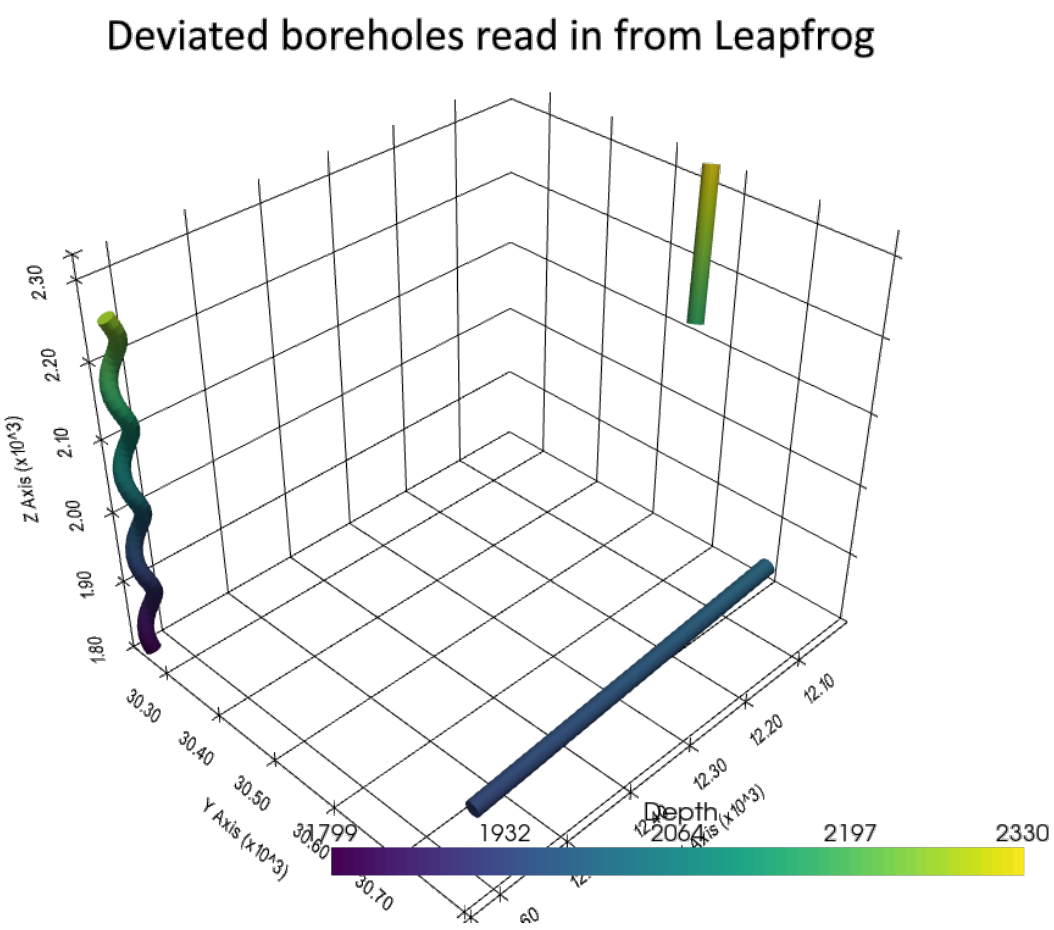
(continued from previous page)

```
p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



6.51 50 Parsing Leapfrog Wells

Leapfrog provides well data in the form of several CSV files. These include a collar file, a survey file, a litho file and an assay file. With GemGIS it is now possible to read in these wells and visualize them.



6.51.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/50_parsing_leapfrog_wells/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="50_parsing_leapfrog_wells.zip",␣
↪dirpath=file_path)
```

```
Downloading file '50_parsing_leapfrog_wells.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2F50_parsing_leapfrog_wells.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\tutorial\data\50_parsing_leapfrog_wells'.
```

6.51.2 Loading Data

The Leapfrog well data is read in as Pandas DataFrames.

[2]:
```
import pandas as pd

litho = pd.read_csv(file_path+'litho.csv')
litho.head()
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:

	holeid	from	to	Geology
0	UG_034	0.00	24.46	Conglomerate
1	UG_034	24.46	44.01	Arkoses
2	UG_034	44.01	122.29	Quartzitic sandstones
3	UG_034	122.29	140.70	Quartzites
4	UG_034	140.70	180.00	Lower Black shales

[3]:
```
collar = pd.read_csv(file_path+'collar.csv', delimiter=';')
collar
```

[3]:

	holeid	x	y	z	maxdepth	comment
0	UG_034	12172.83	30799.71	1977.71	180	underground DH, upwards
1	RWTHO_006	12628.76	30259.13	2259.59	450	inclined
2	SonicS_006	12012.68	30557.53	2325.53	500	lift

[4]:
```
survey = pd.read_csv(file_path+'survey.csv')
survey.head()
```

[4]:

	holeid	depth	dip	azimuth
0	UG_034	0	-65.00	20
1	UG_034	180	-65.00	20
2	RWTHO_006	0	55.00	308
3	RWTHO_006	450	55.00	308
4	SonicS_006	0	90.00	20

6.51.3 Plotting the first well

The first well to be plotted is well SonicS_006.

[5]:
```
survey006 = survey[survey['holeid']=='SonicS_006']
survey006 = survey006.reset_index().drop('index', axis=1)
survey006.head()
```

[5]:
```
       holeid  depth   dip  azimuth
0  SonicS_006      0 90.00       20
1  SonicS_006     10 89.50       20
2  SonicS_006     20 89.00       20
3  SonicS_006     30 88.50       20
4  SonicS_006     40 88.00       20
```

Getting the coordinates of the well at the surface.

[6]:
```
x0 = collar[['x', 'y', 'z']].loc[2].values
x0
```

[6]:
```
array([12012.68053 , 30557.53476 ,  2325.532416])
```

Creating the DataFrame from which the well paths are created.

[7]:
```
df_survey = gg.visualization.create_deviated_borehole_df(df_survey=survey006,␣
↪position=x0)
df_survey.head()
```

[7]:
```
       holeid depth   dip azimuth depth_bottom  \
0  SonicS_006     0 90.00      20        10.00
1  SonicS_006     0 90.00      20        10.00
2  SonicS_006    10 89.50      20        20.00
3  SonicS_006    20 89.00      20        30.00
4  SonicS_006    30 88.50      20        40.00

                                          vector segment_length        X  \
0  [[0.36482400173640905], [-0.18285080511417406]...             -10 12012.68
1  [[0.36482400173640905], [-0.18285080511417406]...             -10 12009.03
2  [[0.4078265278090107], [0.014439265532404447],...             -10 12004.95
3  [[0.3509788964265648], [0.2081941003896598], [...             -10 12001.44
4  [[0.2081993903819504], [0.3509757584484337], [...             -10 11999.36

         Y       Z                                          points
0 30557.53 2325.53     [12012.68053, 30557.534760000002, 2325.532416]
1 30559.36 2316.40  [12009.032289982635, 30559.363268051144, 2316...
2 30559.22 2307.27  [12004.954024704546, 30559.21887539582, 2307.2...
3 30557.14 2298.14  [12001.44423574028, 30557.136934391925, 2298.1...
4 30553.63 2289.01  [11999.362241836461, 30553.62717680744, 2289.0...
```

Creating lines from DataFrame.

[8]:
```
lines = gg.visualization.create_lines_from_points(df=df_survey)
lines
```

[8]:
```
PolyData (0x2050be299a0)
  N Cells:  52
```

(continues on next page)

(continued from previous page)

```
  N Points: 51
  X Bounds: 1.200e+04, 1.202e+04
  Y Bounds: 3.054e+04, 3.056e+04
  Z Bounds: 1.869e+03, 2.326e+03
  N Arrays: 0
```

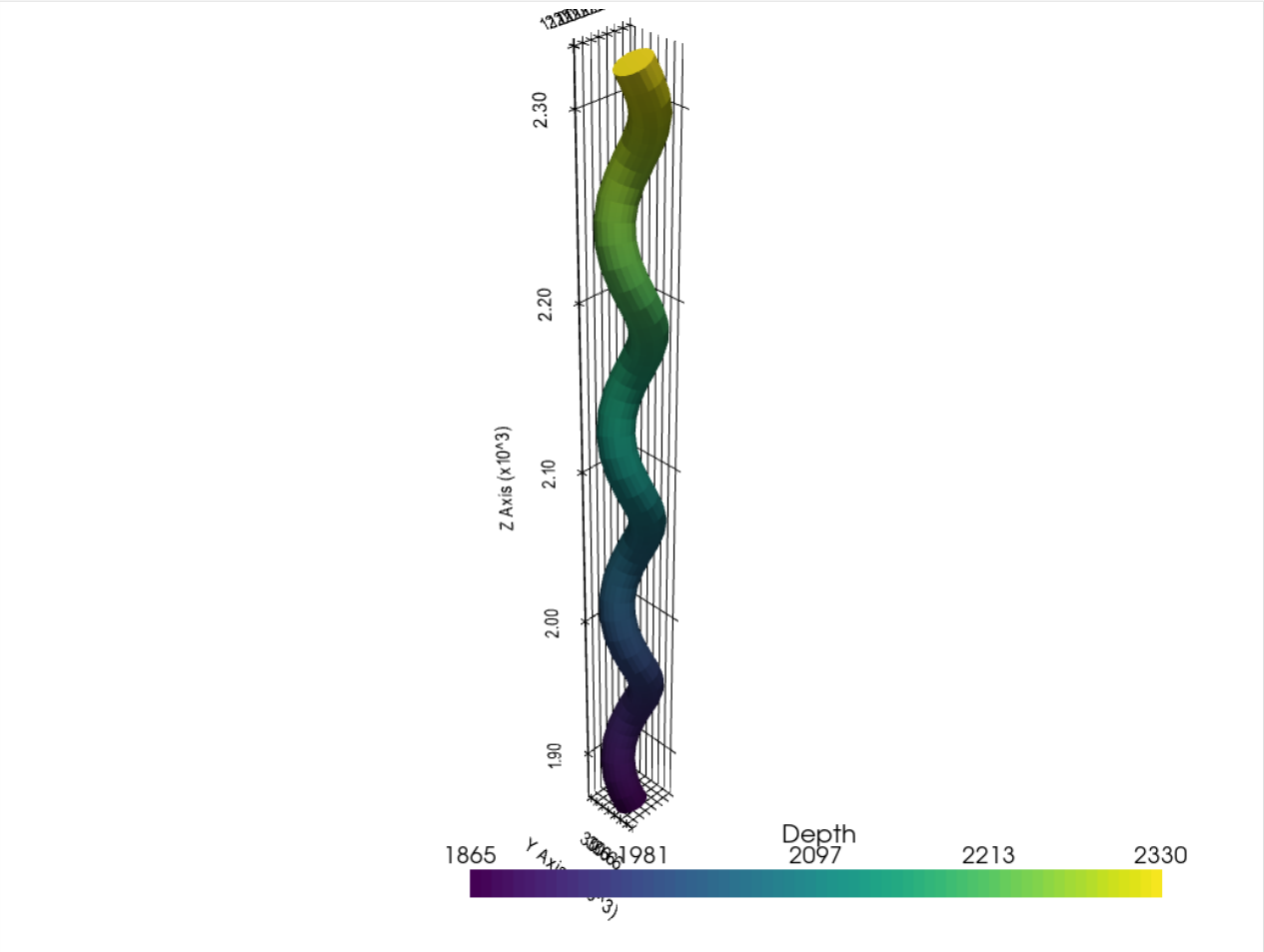
Creating tubes from lines.

[9]:
```
tubes = gg.visualization.create_borehole_tube(df=df_survey, line=lines, radius=10)
tubes
```

[9]:
```
PolyData (0x2050be29e80)
  N Cells:  22
  N Points: 1060
  X Bounds: 1.199e+04, 1.203e+04
  Y Bounds: 3.053e+04, 3.057e+04
  Z Bounds: 1.865e+03, 2.330e+03
  N Arrays: 3
```

Plotting the well.

[10]:
```
import pyvista as pv

sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

# Adding DEM
p.add_mesh(tubes, scalars='Depth', scalar_bar_args=sargs)


p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,1)
p.show()
```



6.51.4 Plotting the second well

The second well that will be plotted is `RWTHO_006`.

[11]:
```
surveyrwth = survey[survey['holeid']=='RWTHO_006']
surveyrwth = surveyrwth.reset_index().drop('index', axis=1)
surveyrwth.head()
```

[11]:
```
      holeid  depth   dip  azimuth
0  RWTHO_006      0 55.00      308
1  RWTHO_006    450 55.00      308
```

Getting the coordinates of the well at the surface.

[12]:
```
x0 = collar[['x', 'y', 'z']].loc[1].values
x0
```

[12]:
```
array([12628.75598, 30259.12988,  2259.59034])
```

Creating the DataFrame from which the well paths are created.

[13]:
```
df_survey = gg.visualization.create_deviated_borehole_df(df_survey=surveyrwth,␣
↪position=x0)
df_survey
```

[13]:
```
      holeid depth   dip azimuth depth_bottom  \
0  RWTHO_006     0 55.00     308       450.00
1  RWTHO_006     0 55.00     308       450.00

                                          vector segment_length        X  \
0  [[-0.992088794420255], [0.021957082623147627],...           -450 12628.76
1  [[-0.992088794420255], [0.021957082623147627],...           -450 13075.20

         Y       Z                                              points
0 30259.13 2259.59     [12628.75598, 30259.12988, 2259.5903399999997]
1 30249.25 2203.97  [13075.195937489114, 30249.249192819585, 2203...
```

Creating lines from DataFrame.

[14]:
```
lines = gg.visualization.create_lines_from_points(df=df_survey)
lines
```

[14]:
```
PolyData (0x20511019040)
  N Cells:  3
  N Points: 2
  X Bounds: 1.263e+04, 1.308e+04
  Y Bounds: 3.025e+04, 3.026e+04
  Z Bounds: 2.204e+03, 2.260e+03
  N Arrays: 0
```

Creating tubes from lines.

[15]:
```
tubes = gg.visualization.create_borehole_tube(df=df_survey, line=lines, radius=10)
tubes
```

[15]:
```
PolyData (0x20511019520)
  N Cells:  22
  N Points: 80
  X Bounds: 1.263e+04, 1.308e+04
  Y Bounds: 3.024e+04, 3.027e+04
  Z Bounds: 2.194e+03, 2.270e+03
  N Arrays: 3
```

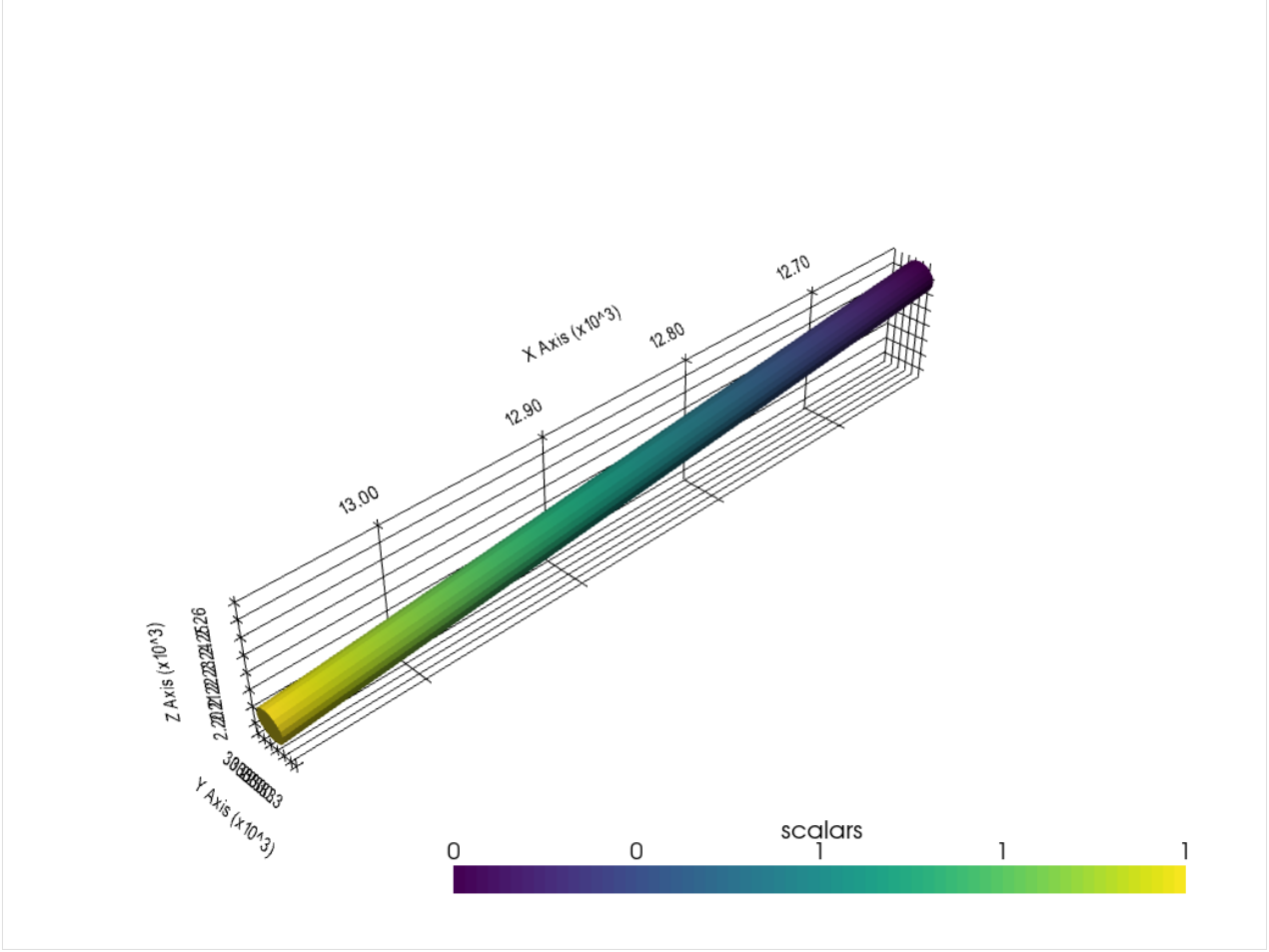
Plotting the well.

[16]:
```
import pyvista as pv

sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

# Adding DEM
p.add_mesh(tubes, scalar_bar_args=sargs)


p.set_background('white')
```

(continues on next page)

(continued from previous page)

```
p.show_grid(color='black')
p.set_scale(1,1,1)
p.show()
```



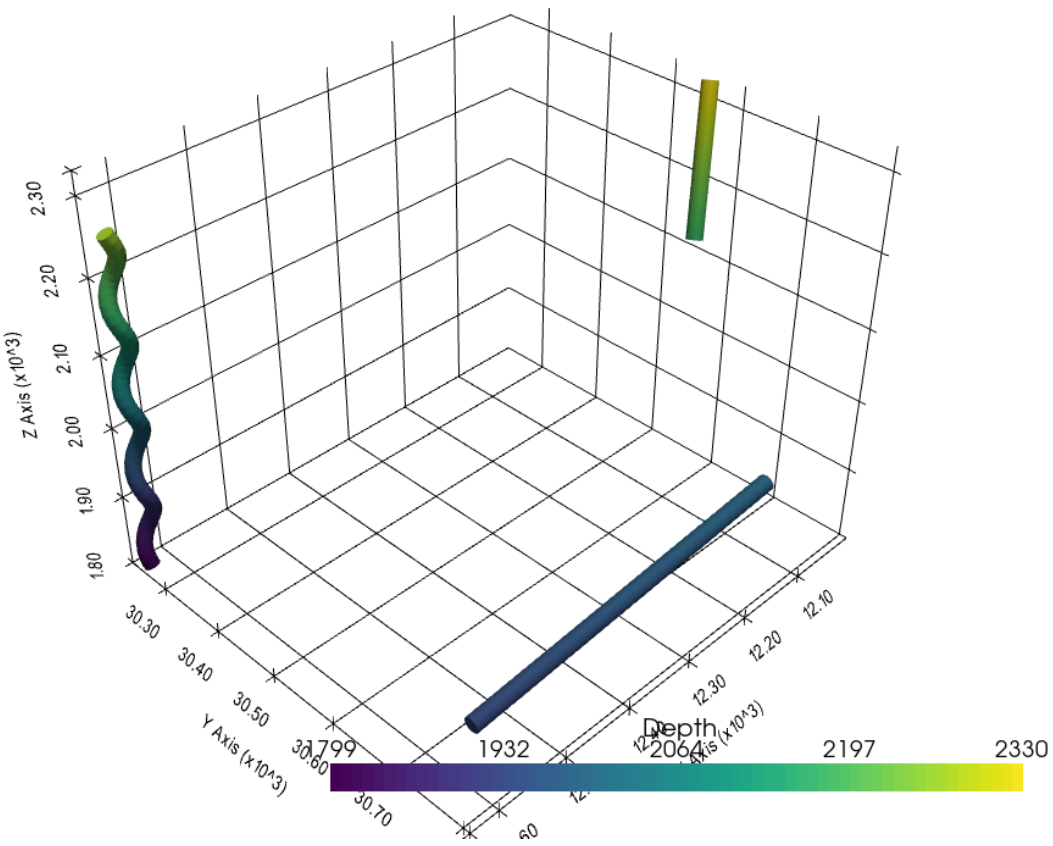
6.51.5 Creating tubes for all wells

All tubes for all wells can be created with the function `create_deviated_boreholes_3d(..)`.

[17]:
```
tubes, df_groups = gg.visualization.create_deviated_boreholes_3d(df_collar=collar,
                                                                 df_survey=survey,
                                                                 min_length=10,
                                                                 collar_depth='maxdepth',
                                                                 survey_depth='depth',
                                                                 index='holeid')
tubes
```

[17]:
```
MultiBlock (0x20511020820)
  N Blocks: 3
  X Bounds: 12003.268, 12641.589
  Y Bounds: 30234.730, 30809.708
  Z Bounds: 1799.074, 2329.613
```

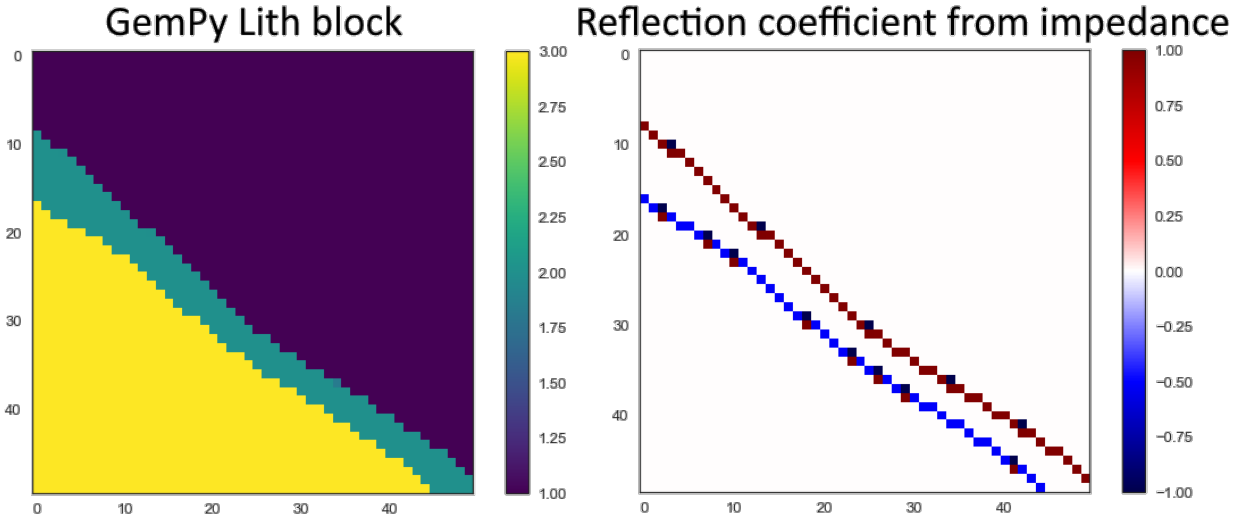
[18]:
```
import pyvista as pv

sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

# Adding DEM
p.add_mesh(tubes, scalars='Depth', scalar_bar_args=sargs)


p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,1)
p.show()
```



[ ]:

6.52 51 Assigning physical properties to GemPy lith blocks

The lith block of a GemPy model returns the spatial distribution of layers in the subsurface. Assuming homogeneous layers and no change in properties with depth, physical properties can easily be assigned to these lithologies. These could include seismic velocities and densities or thermal conductivities to provide a first estimate of temperatures at depth.



6.52.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/51_assigning_physical_properties_to_lith_block/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="51_assigning_physical_
↪properties_to_lith_block.zip", dirpath=file_path)
```

```
Downloading file '51_assigning_physical_properties_to_lith_block.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F51_assigning_physical_properties_
↪to_lith_block.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\
↪tutorial\data\51_assigning_physical_properties_to_lith_block'.
```

(continues on next page)

(continued from previous page)

6.52.2 Loading the lith block

The lith block of a computed GemPy model is loaded using NumPy and reshaped to the original dimensions (resolution) of the model.

[2]:
```
import pyvista as pv
import numpy as np

lith_block = np.load(file_path + 'lith_block.npy').reshape(50,50,50)
lith_block[0]
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
array([[1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       [1., 1., 1., ..., 1., 1., 1.],
       ...,
       [3., 3., 3., ..., 2., 2., 1.],
       [3., 3., 3., ..., 2., 2., 2.],
       [3., 3., 3., ..., 2., 2., 2.]])
```

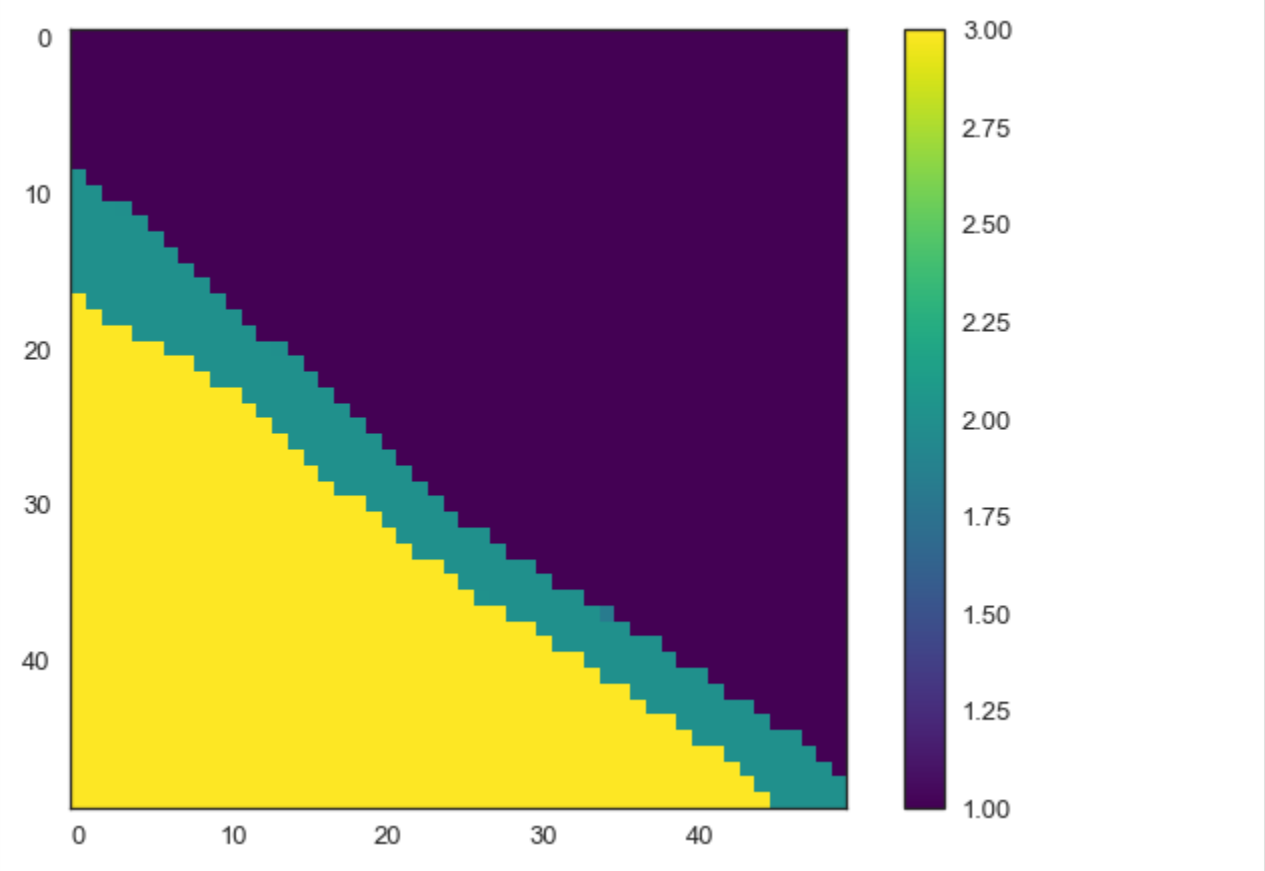
Unique values of the array. Values are rounded.

[3]:
```
np.unique(np.round(lith_block))
```

[3]:
```
array([1., 2., 3.])
```

Plotting the original lith_block.

[4]:
```
import matplotlib.pyplot as plt
im = plt.imshow(lith_block[1,:,:], cmap='viridis')
plt.colorbar(im);
```



6.52.3 Assigning density values by ID

Sample values are stored as list and will then be converted to to a dictionary.

[5]:
```
density_values = [0.1, 2.5, 1,5]

density_dict = {k: v for k,v in zip(np.unique(np.round(lith_block)), density_values)}
density_dict
```

[5]:
```
{1.0: 0.1, 2.0: 2.5, 3.0: 1}
```

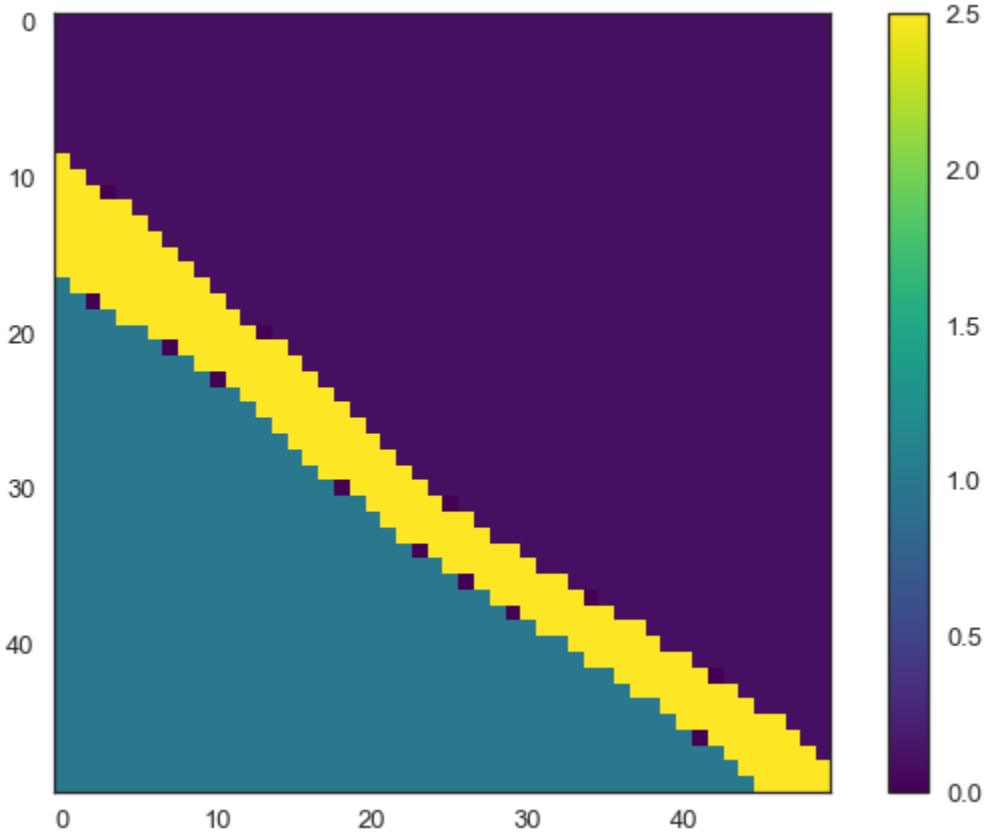
The density block will then be calculated.

[6]:
```
density_block = gg.utils.assign_properties(lith_block=lith_block, property_dict=density_
↪dict)
density_block[0]
```

[6]:
```
array([[0.1, 0.1, 0.1, ..., 0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1, ..., 0.1, 0.1, 0.1],
       [0.1, 0.1, 0.1, ..., 0.1, 0.1, 0.1],
       ...,
       [1. , 1. , 1. , ..., 2.5, 2.5, 0.1],
       [1. , 1. , 1. , ..., 2.5, 2.5, 2.5],
       [1. , 1. , 1. , ..., 2.5, 2.5, 2.5]])
```

The data can be plotted using matplotlib.

[7]:
```
import matplotlib.pyplot as plt

im = plt.imshow(density_block[1,:,:], cmap='viridis')
plt.colorbar(im);
```



6.52.4 Assigning seismic velocities by ID

Sample values are stored as list and will then be converted to to a dict.

[8]:
```
velocity_values = [300,2500,2000]

velocity_dict = {k: v for k,v in zip(np.unique(np.round(lith_block)), velocity_values)}
velocity_dict
```

[8]:
```
{1.0: 300, 2.0: 2500, 3.0: 2000}
```

The velocity block will then be calculated.

[9]:
```
velocity_block = gg.utils.assign_properties(lith_block=lith_block, property_
↪dict=velocity_dict)
velocity_block[0]
```

[9]:
```
array([[ 300,  300,  300, ...,  300,  300,  300],
       [ 300,  300,  300, ...,  300,  300,  300],
       [ 300,  300,  300, ...,  300,  300,  300],
```

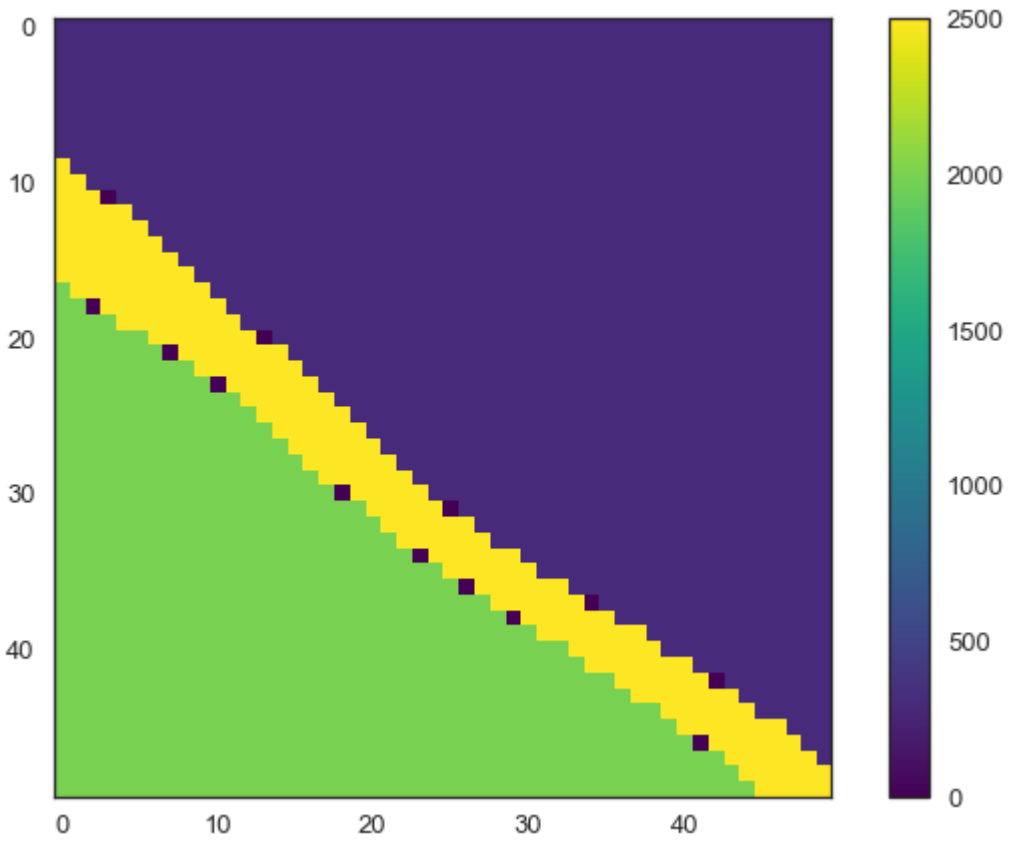
(continues on next page)

(continued from previous page)

```
       ...,
       [2000, 2000, 2000, ..., 2500, 2500,  300],
       [2000, 2000, 2000, ..., 2500, 2500, 2500],
       [2000, 2000, 2000, ..., 2500, 2500, 2500]])
```

The data can be plotted using matplotlib.

[10]:
```
import matplotlib.pyplot as plt

im = plt.imshow(velocity_block[1,:,:], cmap='viridis')
plt.colorbar(im);
```



6.52.5 Calculating acoustic impedance

The acoustic impedance is defined as the product of density and seismic velocity.

$$Z_p = \rho_p * V_p$$

[11]:
```
impendance_block = density_block*velocity_block
impendance_block[0]
```

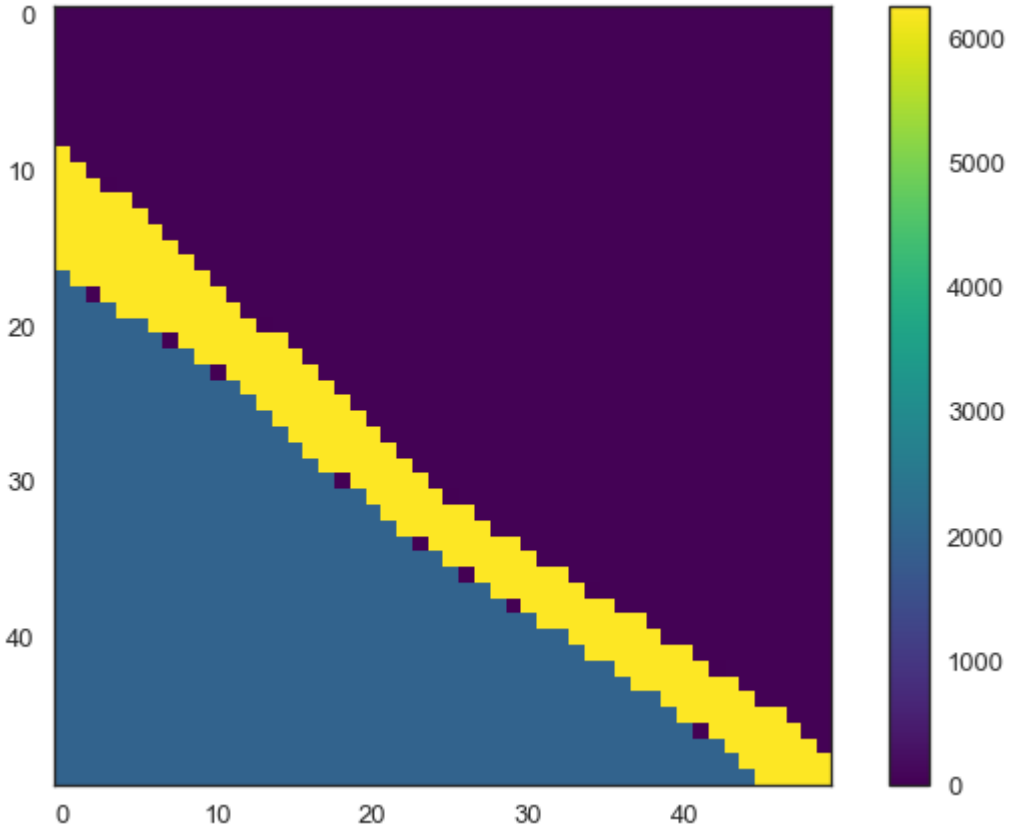
[11]:
```
array([[  30.,   30.,   30., ...,   30.,   30.,   30.],
       [  30.,   30.,   30., ...,   30.,   30.,   30.],
```

(continues on next page)

(continued from previous page)

```
       [  30.,   30.,   30., ...,   30.,   30.,   30.],
       ...,
       [2000., 2000., 2000., ..., 6250., 6250.,   30.],
       [2000., 2000., 2000., ..., 6250., 6250., 6250.],
       [2000., 2000., 2000., ..., 6250., 6250., 6250.]])
```

[12]:
```
import matplotlib.pyplot as plt

im = plt.imshow(impendance_block[1,:,:], cmap='viridis')
plt.colorbar(im);
```



6.52.6 Calculating the reflection coefficient

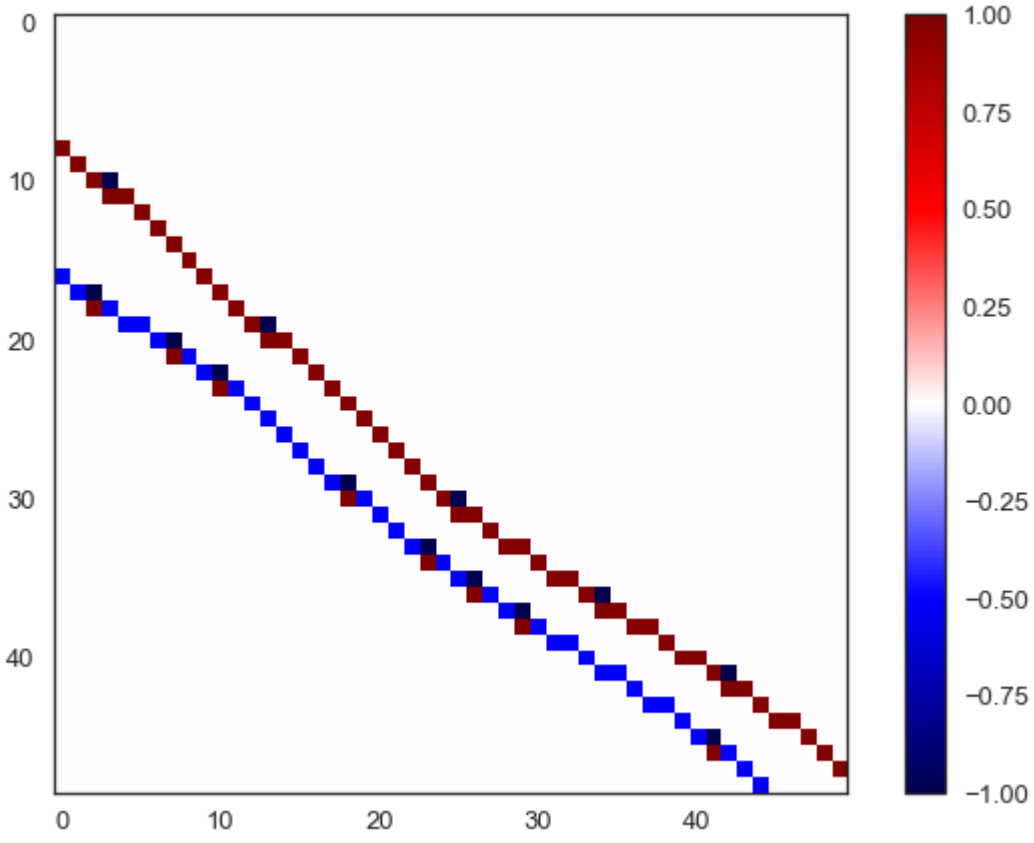
The vertical reflection coefficient is calculated as the difference between two impedance values divided by the sum of the two.

$$R_i = \frac{Z_1 - Z_2}{Z_1 + Z_2}$$

[13]:
```
reflection_coeff = (impendance_block[:,1:] - impendance_block[:,:-1])/(impendance_block[:
↪,1:] + impendance_block[:,:-1])
reflection_coeff[0]
```

[13]:
```
array([[0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ],
       ...,
       [0.        , 0.        , 0.        , ..., 0.        , 0.99044586,
        0.        ],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.99044586],
       [0.        , 0.        , 0.        , ..., 0.        , 0.        ,
        0.        ]])
```

[14]:
```
import matplotlib.pyplot as plt

im = plt.imshow(reflection_coeff[1,:,:], cmap='seismic')
plt.colorbar(im);
```



Replacing zero-values with `np.nan`

[15]:
```
reflection_coeff[reflection_coeff == 0] = np.nan
reflection_coeff[:0]
```

[15]:
```
array([], shape=(0, 49, 50), dtype=float64)
```

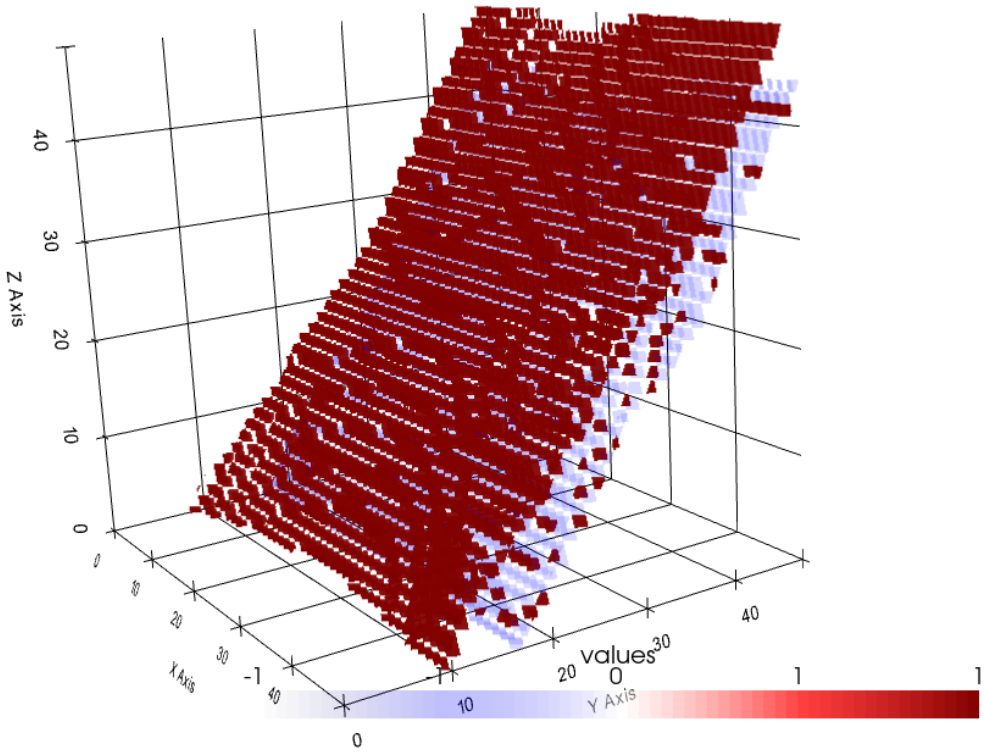
Wrapping the array.

[16]:
```
volume = pv.wrap(reflection_coeff)
volume
```

[16]:
```
UniformGrid (0x24d5b456820)
  N Cells:  115248
  N Points: 122500
  X Bounds: 0.000e+00, 4.900e+01
  Y Bounds: 0.000e+00, 4.800e+01
  Z Bounds: 0.000e+00, 4.900e+01
  Dimensions:       50, 49, 50
  Spacing:  1.000e+00, 1.000e+00, 1.000e+00
  N Arrays: 1
```

Plotting the data with PyVista.

[17]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)
p.camera_position = [(161.91360339500804, -56.76742646880152, 61.85062200360107), (24.5,␣
↪24.5, 24.5), (-0.16718411386271567, 0.1641218812347994, 0.9721694709112653)]

p.add_volume(volume, scalar_bar_args=sargs, cmap='seismic')

p.show_grid(color='black')
p.set_background(color='white')
p.show()
```



[ ]:

6.53 52 Digitizing data from PyVista Meshes

The topography of an area can be visualized in 3D with PyVista.

[ ]:
```
#<img src="../images/tutorial52_cover.png">
```

6.53.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/52_digitizing_data_from_pyvista_meshes/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="52_digitizing_data_from_pyvista_
↪meshes.zip", dirpath=file_path)
```

```
Downloading file '52_digitizing_data_from_pyvista_meshes.zip' from 'https://rwth-aachen.
↪sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F52_digitizing_data_from_pyvista_meshes.zip
↪' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\data\52_
↪digitizing_data_from_pyvista_meshes'.
```

6.53.2 Loading the data

A 50 m DEM of the Münsterland Basin is loaded to illustrate the visualizing of topography in PyVista. The data will be used under Datenlizenz Deutschland – Zero – Version 2.0. It was obtained from the WCS Service https://www.wcs.nrw.de/geobasis/wcs_nw_dgm. The data will automatically be converted to a StructuredGrid with `Elevation [m]` as data array.

[2]:
```
import pyvista as pv

mesh = gg.visualization.read_raster(path=file_path + 'DEM50.tif',
                                    nodata_val=9999.0,
                                    name='Elevation [m]')

mesh
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
StructuredGrid (0x1e06aafa6a0)
  N Cells:  5595201
  N Points: 5600000
  X Bounds: 3.236e+07, 3.250e+07
  Y Bounds: 5.700e+06, 5.800e+06
  Z Bounds: 0.000e+00, 0.000e+00
```

(continues on next page)

(continued from previous page)

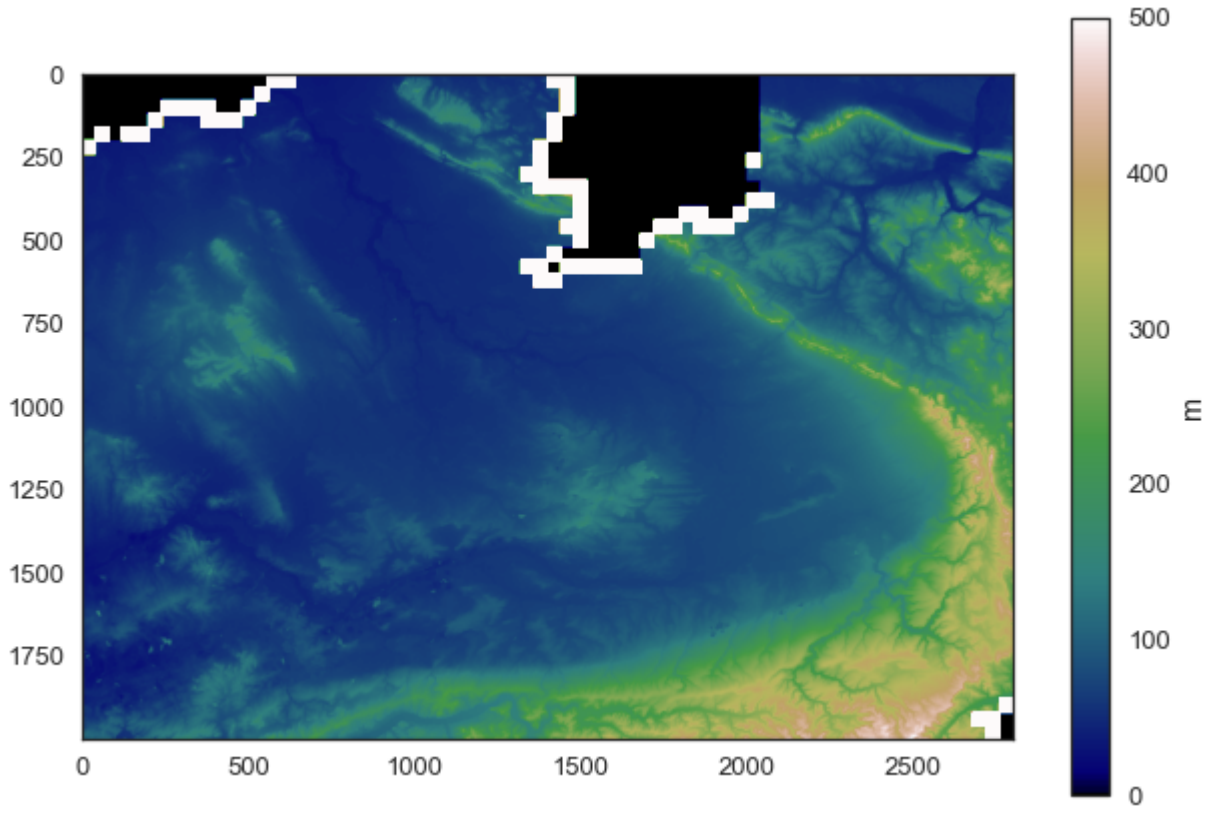
```
  Dimensions:       2000, 2800, 1
  N Arrays: 1
```

6.53.3 Plotting the data in 2D

[3]:
```
import rasterio

dem = rasterio.open(file_path + 'DEM50.tif')
dem.read(1)
```

[3]:
```
array([[  0.  ,   0.  ,   0.  , ...,  40.1 ,  40.09,  44.58],
       [  0.  ,   0.  ,   0.  , ...,  40.08,  40.07,  44.21],
       [  0.  ,   0.  ,   0.  , ...,  40.14,  44.21,  43.98],
       ...,
       [100.56, 102.14, 102.17, ...,   0.  ,   0.  ,   0.  ],
       [ 99.44,  99.85,  99.77, ...,   0.  ,   0.  ,   0.  ],
       [ 88.32,  91.76,  98.68, ...,   0.  ,   0.  ,   0.  ]],
      dtype=float32)
```

[4]:
```
import matplotlib.pyplot as plt

im = plt.imshow(dem.read(1), cmap='gist_earth', vmin=0, vmax=500)
cbar = plt.colorbar(im)
cbar.set_label('m')
```



6.53.4 Wrap Mesh by Scalars

The dataset's points are wrapped by a point data scalars array's values.

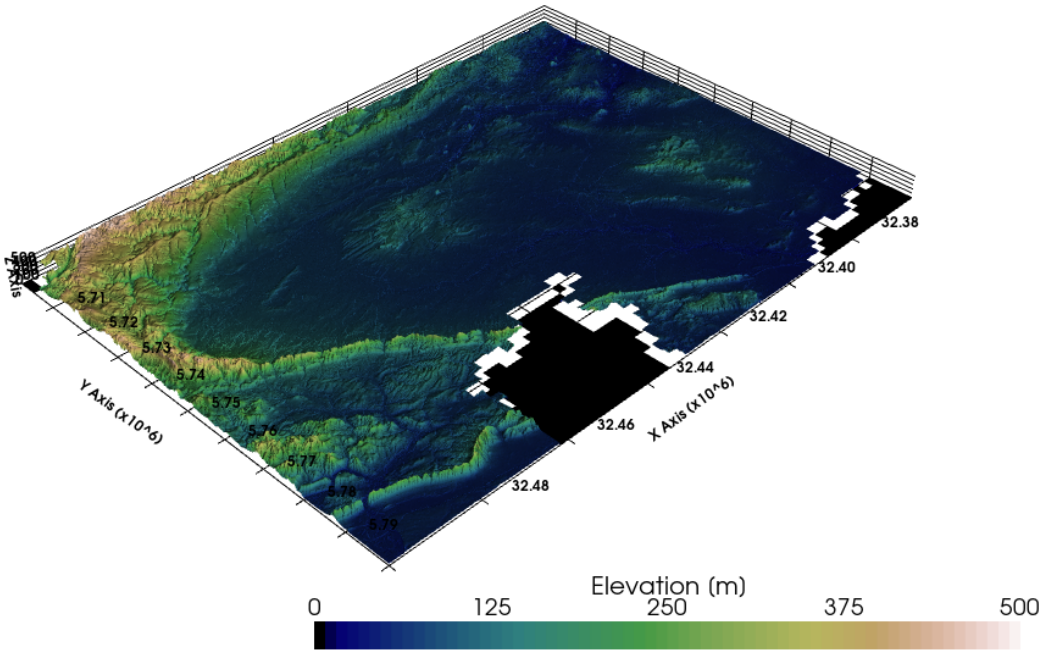
[5]:
```
topo = mesh.warp_by_scalar(scalars="Elevation [m]", factor=1.0)

topo
```

[5]:
```
StructuredGrid (0x1e00adb9dc0)
  N Cells:  5595201
  N Points: 5600000
  X Bounds: 3.236e+07, 3.250e+07
  Y Bounds: 5.700e+06, 5.800e+06
  Z Bounds: 0.000e+00, 5.038e+02
  Dimensions:       2000, 2800, 1
  N Arrays: 1
```

6.53.5 Plotting the Mesh

The mesh can then easily be plotted with PyVista

[6]:
```
sargs = dict(fmt="%.0f", color='black')


p = pv.Plotter(notebook=True)
p.add_mesh(mesh=topo, cmap='gist_earth', scalar_bar_args=sargs, clim=[-0, 500])

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```



[7]:
```
p = pv.Plotter(notebook=True)
p.add_mesh(pv.Sphere(), show_edges=True)
p.enable_geodesic_picking()
p.show()
```

[7]:
```
[(1.9410089246532642, 1.9117894925560552, 1.9264378181907793),
 (0.0, 0.0, 0.0),
 (-0.4096375714326179, -0.4068542655596967, 0.8164965809277261)]
```

[21]:
```
sargs = dict(fmt="%.0f", color='black')


p = pv.Plotter(notebook=True)
p.add_mesh(mesh=topo, cmap='gist_earth', scalar_bar_args=sargs, clim=[-0, 500])

p.enable_geodesic_picking(show_message=True)

p.set_background('white')
p.show_grid(color='black')
p.set_scale(1,1,10)
p.show()
```

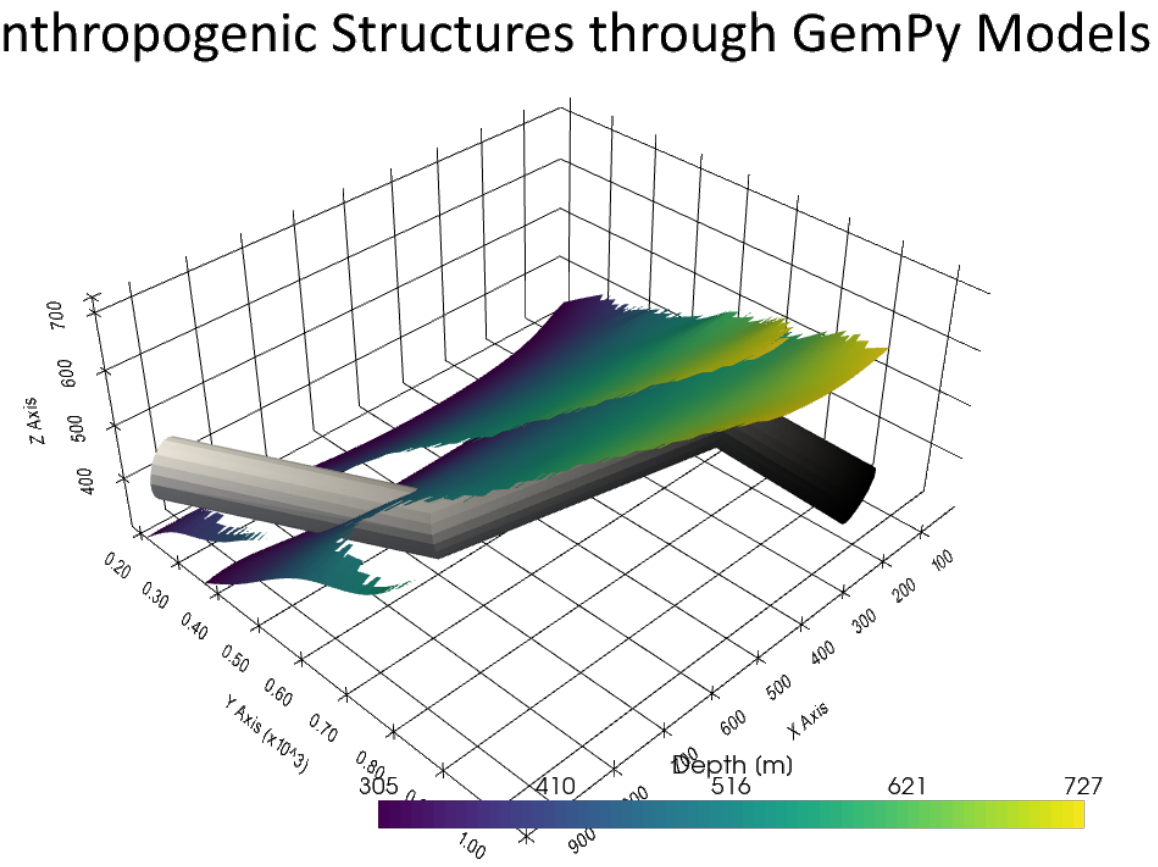
[21]:
```
[(32513908.06643164, 5851789.661366982, 30759.992596354175),
 (32430000.0, 5750000.0, 251.9149932861328),
 (-0.732512038854102, -0.5593046377635457, 0.3880778724817944)]
```

[12]:
```
p.picked_geodesic
```

[12]:
```
PolyData (0x1fb1ac2fa60)
  N Cells:  0
  N Points: 0
  X Bounds: 1.000e+00, -1.000e+00
  Y Bounds: 1.000e+00, -1.000e+00
  Z Bounds: 1.000e+00, -1.000e+00
  N Arrays: 0
```

6.54 53 Adding anthropogenic geometries to PyVista

Geological models are often constructed for the purpose of tunneling or any other type of excavation. As such a tunnel is not a geological but rather anthropegnic feature, it is desirable to visualize the planned or the actual path of the tunnel into the model.



6.54.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import gemgis as gg

file_path ='data/53_adding_anthropogenic_geometries_to_pyvista/'
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
gg.download_gemgis_data.download_tutorial_data(filename="53_adding_anthropogenic_
↪geometries_to_pyvista.zip", dirpath=file_path)
```

```
Downloading file '53_adding_anthropogenic_geometries_to_pyvista.zip' from 'https://rwth-
↪aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F53_adding_anthropogenic_geometries_
↪to_pyvista.zip' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\tutorial\
↪data\53_adding_anthropogenic_geometries_to_pyvista'.
```

6.54.2 Load Layer Data

The example model will be loaded and a tunnel will be defined to cross the layers later on.

[3]:
```
import pyvista as pv

mesh1 = pv.read(file_path+'Layer1.vtk')
mesh2 = pv.read(file_path+'Layer2.vtk')
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
C:\Users\ale93371\Anaconda3\envs\test_gempy\lib\site-packages\theano\configdefaults.py:
↪560: UserWarning: DeprecationWarning: there is no c++ compiler.This is deprecated and␣
↪with Theano 0.11 a c++ compiler will be mandatory
  warnings.warn("DeprecationWarning: there is no c++ compiler."
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

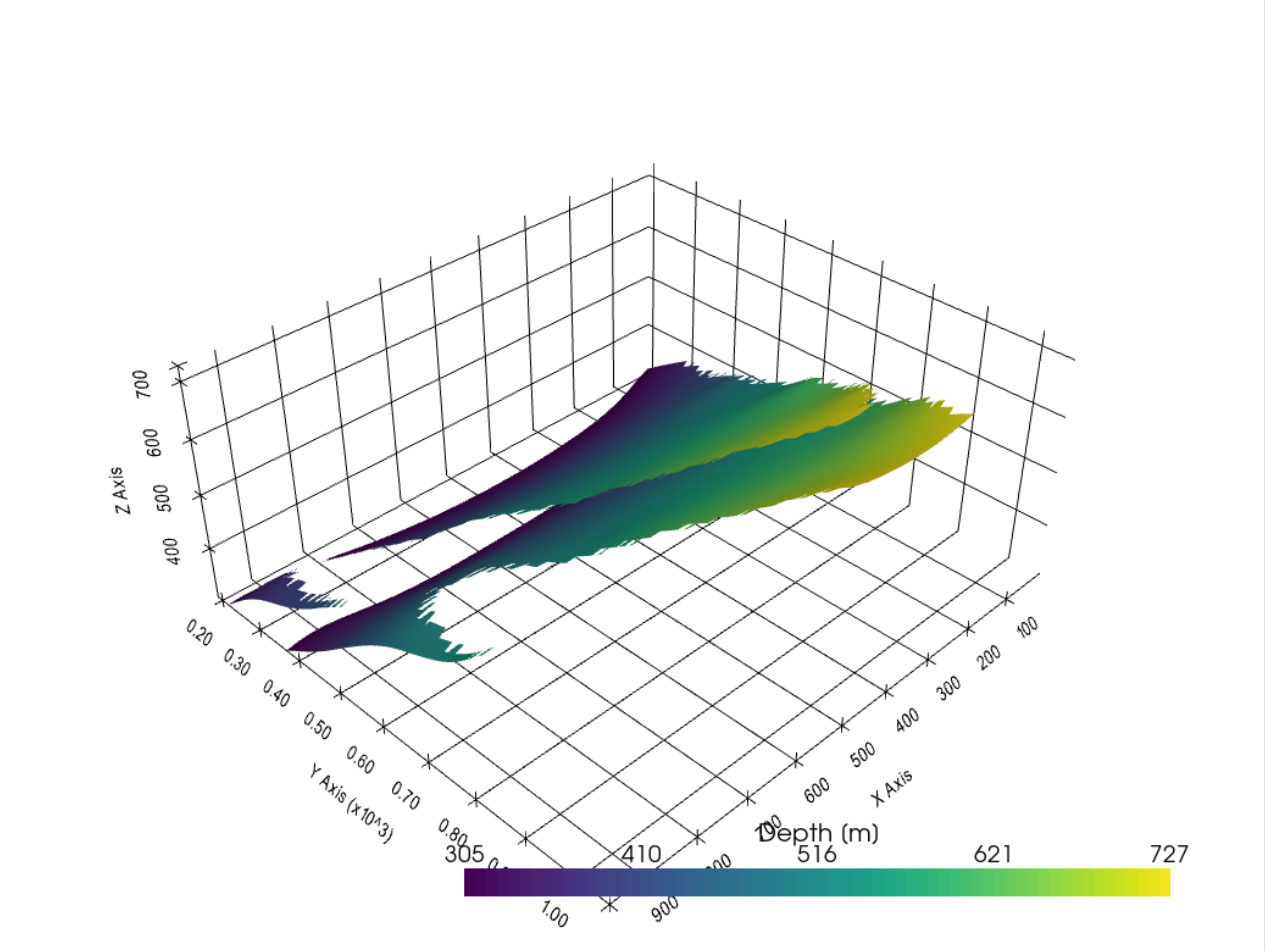
[4]:
```
mesh1
```

[4]:
```
PolyData (0x1e0f5d149a0)
  N Cells:  4174
  N Points: 2303
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 1.881e+02, 9.491e+02
  Z Bounds: 3.050e+02, 7.250e+02
  N Arrays: 1
```

[5]:
```
mesh2
```

[5]:
```
PolyData (0x1e0f5d14a00)
  N Cells:  5111
  N Points: 2739
  X Bounds: 9.720e+00, 9.623e+02
  Y Bounds: 3.578e+02, 1.058e+03
  Z Bounds: 3.050e+02, 7.265e+02
  N Arrays: 1
```

[7]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh1, scalars= 'Depth [m]', scalar_bar_args=sargs)
p.add_mesh(mesh2, scalars= 'Depth [m]', scalar_bar_args=sargs)

p.set_background('white')
p.show_grid(color='black')
p.show()
```



6.54.3 Defining Tunnel Object

The tunnel object will be created as tube from a line.

[30]:
```
import numpy as np

points = np.array([[200,1000,400],
                   [300,800,500],
                   [700,600,400],
                   [900,200,400]])
points
```

[30]:
```
array([[ 200, 1000,  400],
       [ 300,  800,  500],
       [ 700,  600,  400],
       [ 900,  200,  400]])
```

[31]:
```
poly = pv.PolyData(points)
poly.points = points
the_cell = np.arange(0, len(points), dtype=np.int_)
the_cell = np.insert(the_cell, 0, len(points))
```

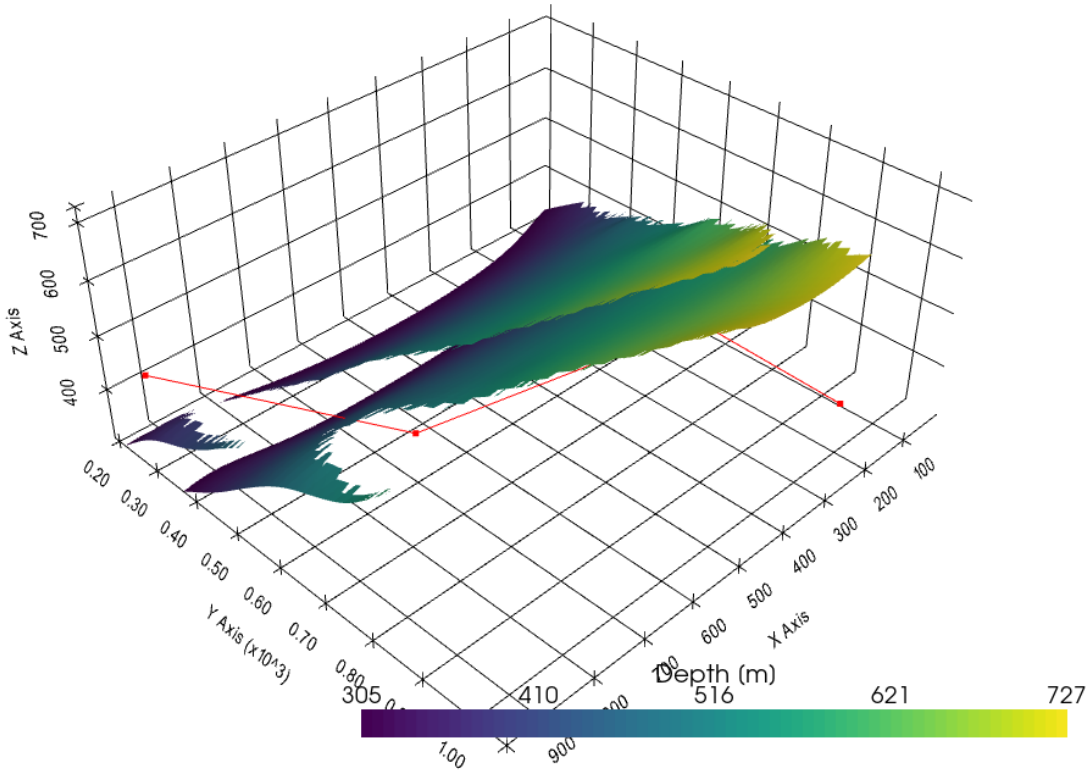
(continues on next page)

(continued from previous page)

```
poly.lines = the_cell
poly
```

[31]:
```
PolyData (0x1e080038820)
  N Cells:  5
  N Points: 4
  X Bounds: 2.000e+02, 9.000e+02
  Y Bounds: 2.000e+02, 1.000e+03
  Z Bounds: 4.000e+02, 5.000e+02
  N Arrays: 0
```

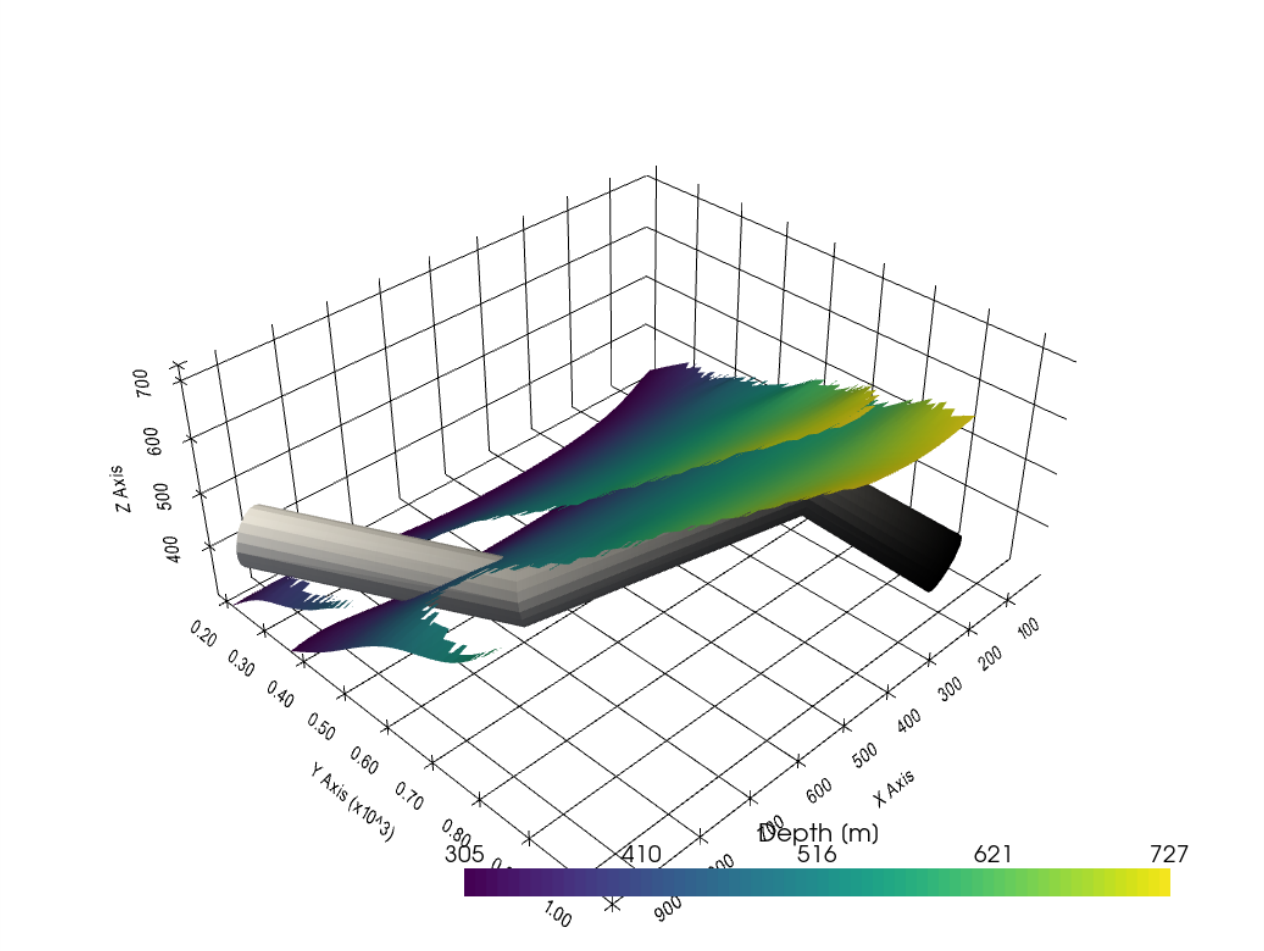
[32]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh1, scalars= 'Depth [m]', scalar_bar_args=sargs)
p.add_mesh(mesh2, scalars= 'Depth [m]', scalar_bar_args=sargs)
p.add_mesh(poly, color='red')

p.set_background('white')
p.show_grid(color='black')
p.show()
```



[35]:
```
poly['scalars'] = np.arange(len(points))
tube = poly.tube(radius=50)

tube
```

[35]:
```
PolyData (0x1e08029b700)
  N Cells:  22
  N Points: 120
  X Bounds: 1.540e+02, 9.440e+02
  Y Bounds: 1.770e+02, 1.028e+03
  Z Bounds: 3.500e+02, 5.490e+02
  N Arrays: 2
```

[41]:
```
sargs = dict(fmt="%.0f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(mesh1, scalars= 'Depth [m]', scalar_bar_args=sargs)
p.add_mesh(mesh2, scalars= 'Depth [m]', scalar_bar_args=sargs)
p.add_mesh(poly, color='red')
p.add_mesh(tube, show_scalar_bar=False, cmap='gray')

p.set_background('white')
p.show_grid(color='black')
p.show()
```



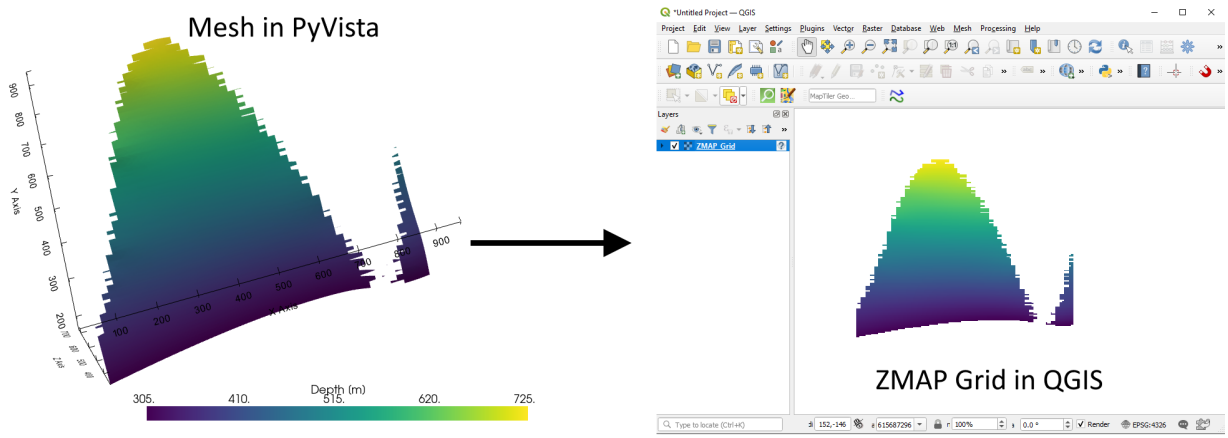
[ ]:

[ ]:

[ ]:

6.55 54 Converting PyVista Mesh to ZMAP Grid

After creating a model, the resulting surfaces are often needed for further processing or visualization. One common grid type are ZMAP Grids (https://gdal.org/drivers/raster/zmap.html) common to the O&G industry and readable in GIS packages like ArcGIS or QGIS. In this notebook, we present methods to convert a PyVista mesh, the resulting surface when using GemPy for structural geological modeling to a ZMAP grid.



6.55.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
import geopandas as gpd
import rasterio
import gempy as gp
import pyvista as pv
import matplotlib.pyplot as plt
import numpy as np
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
file_path ='data/54_converting_pyvista_mesh_to_zmap_grid/'
gg.download_gemgis_data.download_tutorial_data(filename="54_converting_pyvista_mesh_to_
↪zmap_grid.zip", dirpath=file_path)
```

6.55.2 Creating GemPy Model

Loading Topography

[3]:
```
topo_raster = rasterio.open(file_path + 'raster1.tif')
```

Creating Interfaces

[4]:
```
interfaces = gpd.read_file(file_path + 'interfaces1_lines.shp')
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords.head()
```

[4]:

	formation	geometry	X	Y	Z
0	Sand1	POINT (0.256 264.862)	0.26	264.86	353.97
1	Sand1	POINT (10.593 276.734)	10.59	276.73	359.04
2	Sand1	POINT (17.135 289.090)	17.13	289.09	364.28
3	Sand1	POINT (19.150 293.313)	19.15	293.31	364.99
4	Sand1	POINT (27.795 310.572)	27.80	310.57	372.81

Creating Orientations

[5]:
```
orientations = gpd.read_file(file_path + 'orientations1.shp')
orientations = gg.vector.extract_xyz(gdf=orientations, dem=topo_raster)
orientations['polarity'] = 1
orientations
```

[5]:

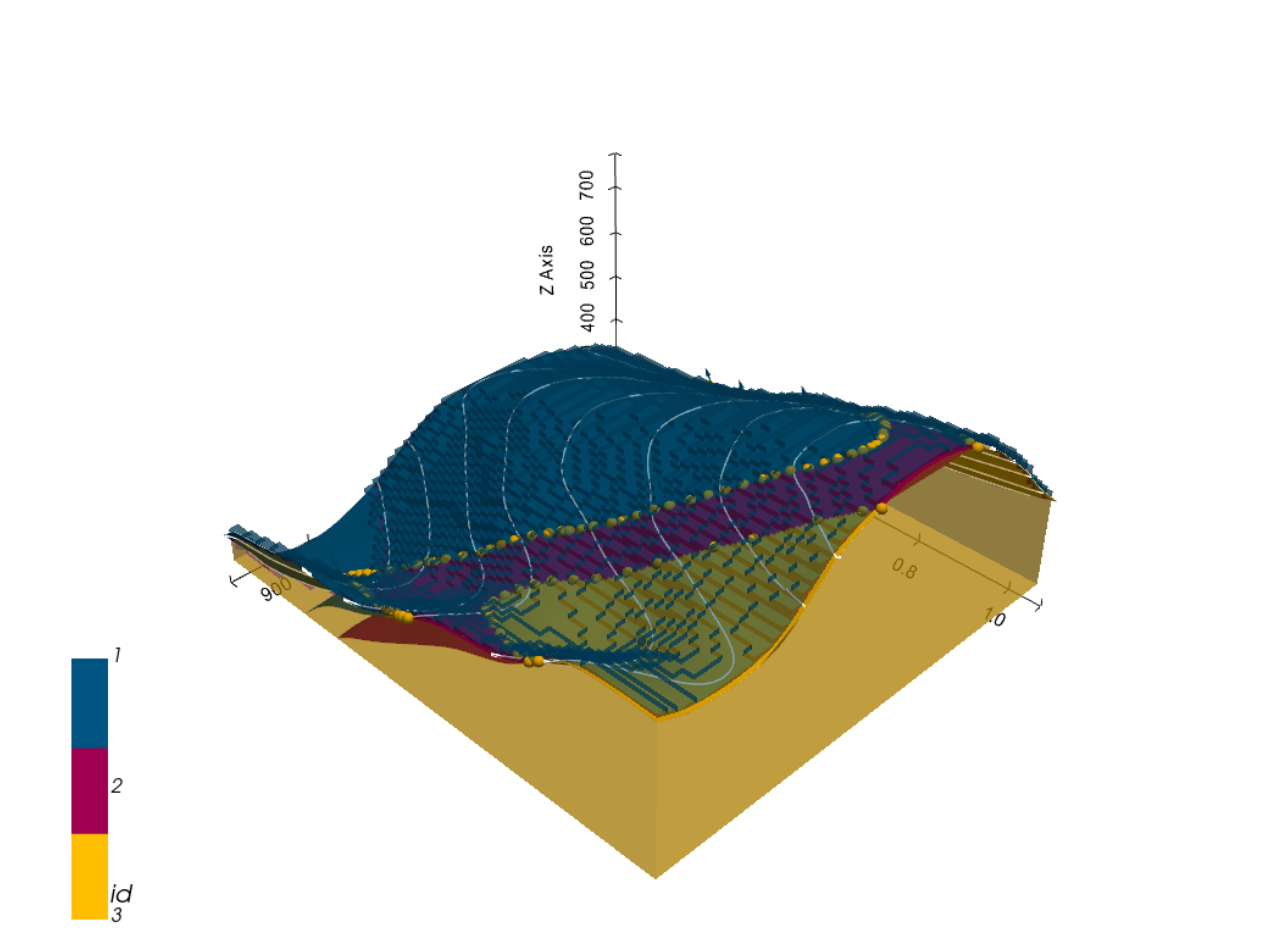
	formation	dip	azimuth	geometry	X	Y	Z	polarity
0	Ton	30.50	180.00	POINT (96.471 451.564)	96.47	451.56	440.59	1
1	Ton	30.50	180.00	POINT (172.761 661.877)	172.76	661.88	556.38	1
2	Ton	30.50	180.00	POINT (383.074 957.758)	383.07	957.76	729.02	1
3	Ton	30.50	180.00	POINT (592.356 722.702)	592.36	722.70	601.55	1
4	Ton	30.50	180.00	POINT (766.586 348.469)	766.59	348.47	378.63	1
5	Ton	30.50	180.00	POINT (843.907 167.023)	843.91	167.02	282.61	1
6	Ton	30.50	180.00	POINT (941.846 428.883)	941.85	428.88	423.45	1
7	Ton	30.50	180.00	POINT (22.142 299.553)	22.14	299.55	368.05	1

Calculating GemPy Model

[6]:
```
geo_model = gp.create_model('Model1')
gp.init_data(geo_model, [0, 972, 0, 1069, 300, 800], [50, 50, 50],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
gp.map_stack_to_surfaces(geo_model,
                         {'Strata': ('Sand1', 'Ton')},
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_topography(source='gdal', filepath=file_path + 'raster1.tif')
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
sol = gp.compute_model(geo_model, compute_mesh=True)
```

```
Active grids: ['regular']
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                   values
range              1528.9
$C_o$           55655.83
drift equations      [3]
```

[7]:
```
gpv = gp.plot_3d(geo_model,
                 image=False,
                 show_topography=True,
                 plotter_type='basic',
                 notebook=True,
                 show_lith=True,
                 show_boundaries=True)
```



6.55.3 Extracting Surfaces from the GemPy Model

[8]:
```
mesh = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model, surfaces=[
↪'Sand1', 'Ton'])
mesh
```

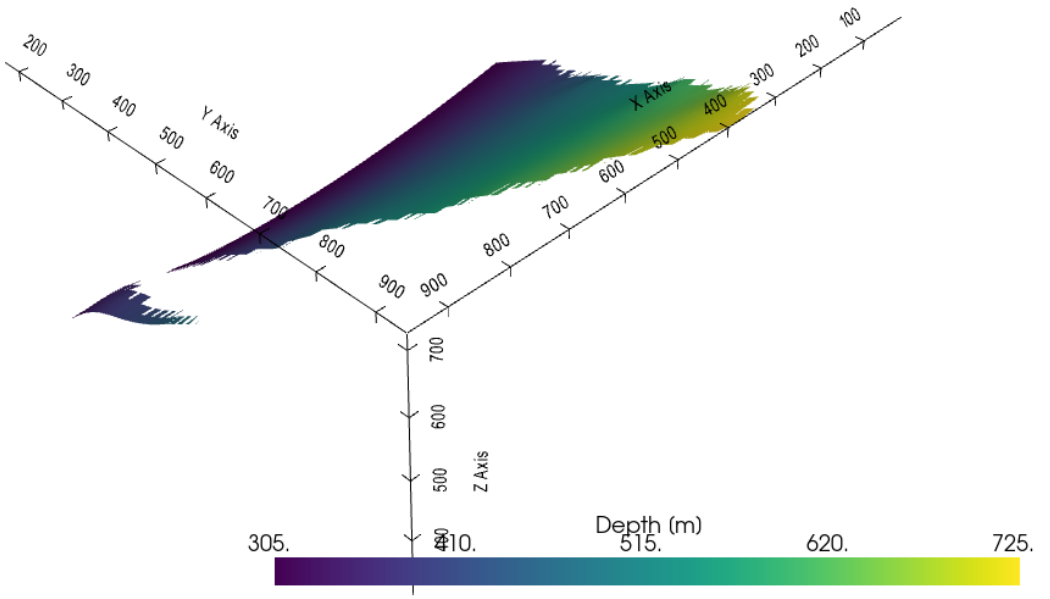
[8]:
```
{'Sand1': [PolyData (0x28273f07b20)
    N Cells:    4209
    N Points:   2325
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   1.672e+02, 9.497e+02
    Z Bounds:   3.050e+02, 7.250e+02
    N Arrays:   1,
  '#015482'],
 'Ton': [PolyData (0x28273f079a0)
    N Cells:    5396
    N Points:   2891
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   2.799e+02, 1.058e+03
    Z Bounds:   3.050e+02, 7.297e+02
    N Arrays:   1,
```

(continues on next page)

(continued from previous page)

```
  '#9f0052']}
```

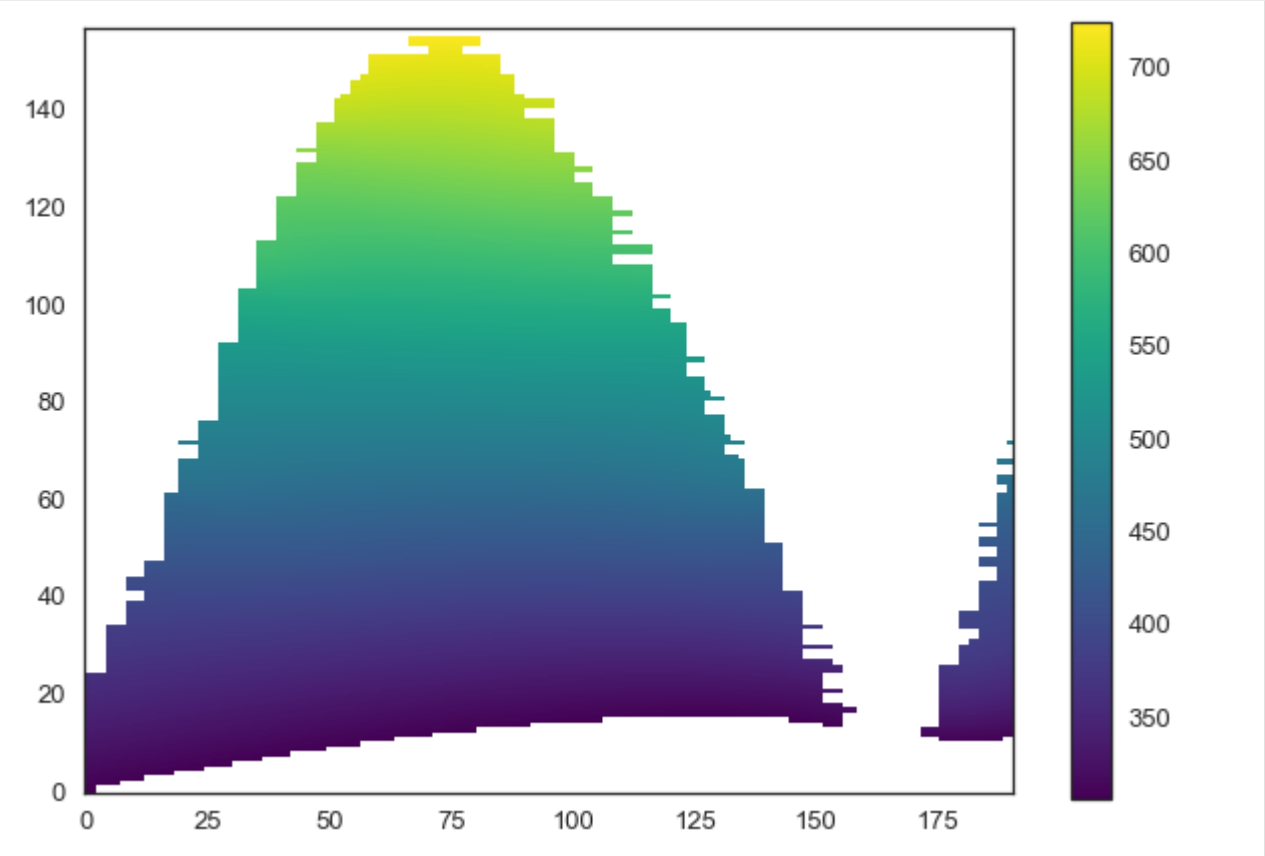
[9]:
```
p = pv.Plotter(notebook=True)

p.add_mesh(mesh['Sand1'][0], scalars = 'Depth [m]')
p.show_bounds()
p.show()
```



6.55.4 Creating the ZMAP Grid

[11]:
```
z_values = gg.utils.extract_zmap_data(surface=mesh['Sand1'][0],
                                      cell_width=5)

z_values[z_values == -9999] = np.nan
im = plt.imshow(np.flipud(z_values.T), cmap='viridis')
plt.gca().invert_yaxis()
plt.colorbar(im)
```

[11]:
```
<matplotlib.colorbar.Colorbar at 0x28276bc3c70>
```



```
[12]: zmap_grid = gg.utils.create_zmap_grid(surface=mesh['Sand1'][0],
                                          cell_width=5, comments=['Hi'], nodata=-9999
                         )
      zmap_grid
```

```
[12]: ['!',
       '! This ZMAP Grid was created using the GemGIS Package',
       '! See https://github.com/cgre-aachen/gemgis for more information',
       '!',
       '! Hi',
       '!',
       '@ZMAP_GRID, GRID, 5',
       '15, -9999, , 5, 1',
       '157, 191, 9.72, 962.2800000000001, 167.18766848564147, 949.705025253296',
       '0.0, 0.0, 0.0',
       '@',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
       '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      364.94156      362.64417      360.20297',
'      357.76175      355.32056      352.87173      350.46216      348.05258',
'      345.64304      343.21008      340.77084      338.33575      335.90067',
'      333.45825      330.98532      328.51764      326.04996      323.57886',
'      321.08279      318.58524      316.08768      313.58694      311.07065',
'      308.54611      306.02158',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      374.85455    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      364.40207      362.16260      359.83536',
'      357.39417      354.95248      352.51941      350.06134      347.65176',
'      345.24219      342.79318      340.33258      337.89749      335.46240',
```

(continues on next page)

(continued from previous page)

```
'      333.01044      330.52856      328.05936      325.59167      323.11554',
'      320.61404      318.11374      315.61615      313.11337      310.59250',
'      308.06476      305.54022',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      374.57776    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      363.84482      361.60532      359.38040',
'      357.02658      354.58905      352.17371      349.69751      347.25095',
'      344.84091      342.37628      339.91345      337.45923      335.02414',
'      332.56265      330.08078      327.60107      325.13339      322.65048',
'      320.14899      317.64224      315.14465      312.63617      310.11530',
'      307.58340      305.05887',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      374.33408    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      363.28754      361.04807      358.82315',
'      356.59824      354.24335      351.82800      349.35178      346.87558',
'      344.42853      341.95938      339.49658      337.03375      334.58191',
'      332.11487      329.63300      327.15051      324.67355      322.18542',
'      319.68393      317.17627      314.67218      312.15900      309.63812',
'      307.10611    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      384.31390      382.20441      380.05688',
'      378.03625      376.01840      374.03424      372.01978      369.74490',
'      367.44330      365.14169      362.84998      360.58636      358.36145',
'      356.13654      353.84976      351.44479      348.96860      346.49240',
'      344.01822      341.54465      339.07874      336.61591      334.15164',
'      331.67804      329.18854      326.70605      324.22180      321.73303',
'      319.22119      316.71350      314.20410      311.69092      309.16199',
'      306.62997    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      384.07721      381.95627      379.79822',
'      377.68195      375.66409      373.69183      371.60074      369.37659',
'      367.07614      364.77530      362.50558      360.21246      357.96033',
'      355.73541      353.42538      351.02267      348.56165      346.08545',
'      343.61212      341.14035      338.66031      336.19748      333.73251',
'      331.24658      328.74622      326.26373      323.77847      321.27844',
'      318.76059      316.25223      313.74179      311.22220      308.68854',
'      306.15451    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      383.84055      381.69760      379.53955',
```

(continues on next page)

(continued from previous page)

```
'      377.35785      375.30975      373.28824      371.18170      368.95755',
'      366.70895      364.40936      362.16119      359.86807      357.56174',
'      355.33426      353.00034      350.59763      348.14874      345.67850',
'      343.20758      340.73605      338.24969      335.77905      333.30685',
'      330.81512      328.31259      325.82138      323.33176      320.82388',
'      318.30603      315.79092      313.27811      310.75070      308.21701',
'      305.67905    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      383.58475      381.43890      379.28088',
'      377.09918      374.95694      372.86920      370.76266      368.53851',
'      366.31436      364.06058      361.81680      359.52371      357.21735',
'      354.92935      352.57529      350.17261      347.72369      345.27155',
'      342.80328      340.33176      337.84540      335.36063      332.87537',
'      330.38367      327.88113      325.37906      322.87720      320.36932',
'      317.85144      315.32962      312.80661      310.27921      307.74551',
'      305.20358    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      404.82413      402.79218      400.76022',
'      398.64682    -9999.00000      394.42096      392.24686      390.09021',
'      387.86954      385.61780      383.44290      381.29709      379.03018',
'      376.84882      374.67984      372.58139      370.41382      368.17276',
'      365.94861      363.71320      361.46298      359.14453      356.83817',
'      354.52213      352.16800      349.75720      347.30829      344.85837',
'      342.39142      339.91910      337.43188      334.94540      332.45370',
'      329.95609      327.44830      324.94525      322.43735      319.92548',
'      317.39865      314.87595      312.34851      309.82101      307.27634',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      404.79367      402.76172      400.67563',
'      398.47952    -9999.00000      394.29300      392.11893      390.05014',
'      387.78931      385.51572      383.34561      381.12787      378.83194',
'      376.60162      374.44208      372.32480      370.09665      367.82730',
'      365.60315      363.36472      361.09665      358.75803      356.44577',
'      354.12158      351.75653      349.34457      346.89658      344.44360',
'      341.97787      339.50690      337.01483      334.52747      332.03427',
'      329.52924      327.01495      324.51102      322.00314      319.47787',
'      316.94656      314.42294      311.89554      309.35507      306.80798',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      404.76324      402.70444      400.50833',
'      398.31223    -9999.00000      394.16504      392.06369      390.01010',
'      387.74924      385.41360      383.23087      380.92929      378.63336',
'      376.35443      374.20432      372.00763      369.77948      367.50034',
'      365.25769      363.00452      360.73032      358.39169      356.05334',
'      353.71942      351.34351      348.93155      346.48486      344.02887',
'      341.56567      339.09473      336.59778      334.10950      331.60745',
'      329.10239      326.58185      324.07681      321.55618      319.03024',
'      316.49548      313.96997      311.43225      308.88913      306.34091',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      404.73279      402.53714      400.34103',
'      398.14490    -9999.00000      394.07721      392.02362      389.97003',
'      387.70920      385.31940      383.03229      380.73071      378.43478',
'      376.14978      373.91507      371.69043      369.46228      367.18317',
'      364.91147      362.63818      360.36398      358.02536      355.66946',
'      353.30637      350.93048      348.51852      346.07227      343.61618',
'      341.15350      338.68256      336.18497      333.68564      331.18060',
'      328.67557      326.15500      323.63452      321.10858      318.58264',
'      316.04788      313.50943      310.96631      308.42316      305.87497',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      434.29800    -9999.00000    -9999.00000',
'    -9999.00000      423.22137    -9999.00000    -9999.00000      413.79324',
'      410.48105      407.34897      404.90781      402.71167      400.51556',
'      398.15729      396.04114      394.01489      391.96133      389.87259',
'      387.56769      385.17789      382.86975      380.56818      378.23175',
'      375.94675      373.69711      371.47247      369.18008      366.88055',
'      364.60229      362.32806      360.02682      357.65878      355.30289',
'      352.92953      350.54965      348.11670      345.67047      343.21228',
'      340.74957      338.26837      335.77078      333.26804      330.76297',
'      328.24997      325.72641      323.20184      320.67590      318.14291',
'      315.60184      313.05975      310.51663      307.96973      305.41235',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      434.24573    -9999.00000    -9999.00000',
'    -9999.00000      423.34689    -9999.00000    -9999.00000      414.06311',
'      410.76505      407.69913      405.16449      402.95331      400.68509',
'      398.16296      395.97620      393.94824      391.89252      389.70026',
'      387.40384      385.01645      382.71426      380.39694      378.05731',
'      375.74286      373.49860      371.23624      368.91092      366.58081',
'      364.30319      362.02167      359.67776      357.29214      354.93570',
'      352.55981      350.15616      347.71884      345.27084      342.80997',
'      340.33987      337.85712      335.35620      332.85220      330.34125',
'      327.82407      325.29745      322.77209      320.24048      317.69876',
'      315.15613      312.61325      310.06635      307.50784    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      434.21265    -9999.00000    -9999.00000',
'    -9999.00000      423.50299    -9999.00000    -9999.00000      414.33301',
'      411.11520      408.04929      405.43051      403.19492      400.69077',
'      398.16864      395.92499      393.88156      391.72021      389.52795',
'      387.23151      384.86038      382.54980      380.22427      377.88467',
'      375.53897      373.29007      370.96704      368.64172      366.28275',
'      364.00409      361.67261      359.32870      356.93799      354.56598',
'      352.17114      349.76263      347.32535      344.87036      342.40289',
'      339.92880      337.44604      334.94144      332.43250      329.91531',
```

(continues on next page)

(continued from previous page)

```
'      327.39816      324.86826      322.33804      319.79633      317.25464',
'      314.70990      312.16299      309.60446      307.04593    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000      434.17960    -9999.00000    -9999.00000',
'    -9999.00000      423.67126    -9999.00000    -9999.00000      414.61728',
'      411.46536      408.39944      405.75787      403.21857      400.69644',
'      398.17432      395.93066      393.74014      391.54788      389.35562',
'      387.05917      384.70267      382.37714      380.05161      377.71201',
'      375.34509      373.02087      370.69788      368.37253      366.01358',
'      363.66745      361.32355      358.97964      356.58893      354.18616',
'      351.77765      349.36914      346.93182      344.46594      341.99182',
'      339.51770      337.03494      334.52377      332.00659      329.48941',
'      326.97226      324.43561      321.89392      319.35220      316.81049',
'      314.25961      311.70108      309.14258      306.58405    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'      453.28830      450.14679      446.93759      443.65228      441.07126',
'      438.70612      436.34100      433.97855      431.48380      428.87115',
'      426.25635      423.51788      420.75244      417.74072      414.68347',
'      411.53156      408.58191      405.93335      403.36902      400.84689',
'      398.26587      395.94290      393.74304      391.55078      389.29324',
'      386.90903      384.55463      382.22910      379.89917      377.50854',
'      375.14163      372.81598      370.49298      368.11008      365.73865',
'      363.39001      361.04614      358.66397      356.24860      353.84280',
'      351.43427      349.01166      346.54926      344.08109      341.60696',
'      339.13031      336.62833      334.11536      331.59818      329.08102',
'      326.54913      324.00366      321.45291      318.91119      316.36951',
'      313.80945      311.24399      308.68539      306.12680    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'      453.69250      450.55096      447.00308      443.66626      441.03455',
'      438.49533      436.13019      433.72995      431.22751      428.71442',
'      426.09958      423.36124      420.64624      417.78964      414.73407',
'      411.61188      408.76016      406.09924      403.52744      400.98489',
'      398.34924      395.95551      393.75552      391.54575      389.15771',
'      386.76028      384.40793      382.08243      379.70978      377.31586',
'      374.93765      372.61462      370.24744      367.85251      365.46338',
'      363.11655      360.74536      358.33536      355.90872      353.50220',
'      351.08499      348.63474      346.16730      343.69766      341.22272',
'      338.72589      336.22192      333.70792      331.19073      328.65894',
'      326.12549      323.57626      321.01852      318.47037      315.92865',
'      313.36444      310.79166      308.22867      305.67007    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'      454.09671      450.70093      447.06854      443.68027      441.04852',
'      438.41867      435.91940      433.47366      430.97122      428.48434',
'      425.94284      423.25500      420.54001      417.74557      414.78470',
'      411.79016      408.93845      406.26868      403.68582      401.06827',
'      398.43262      395.99292      393.76804      391.41019      389.02216',
'      386.62473      384.26126      381.90057      379.52039      377.12646',
'      374.73630      372.38269      369.98987      367.59494      365.18814',
'      362.82675      360.41675      358.00674      355.56885      353.15829',
'      350.70804      348.25778      345.78552      343.31421      340.81830',
'      338.32144      335.81693      333.30048      330.76877      328.23529',
'      325.70184      323.14886      320.59113      318.03339      315.48782',
'      312.92038      310.34665      307.77390      305.21332    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000      483.46118',
'      481.17871    -9999.00000    -9999.00000      474.56229      472.33789',
'      469.86383      467.17337      464.42352      461.42407      458.21234',
'      454.41504      450.78265      447.15024      443.71674      441.08475',
'      438.45490      435.82504      433.23834      430.72540      428.23853',
'      425.75165      423.14993      420.43793      417.64349      414.84399',
'      411.95541      409.10828      406.43854      403.78452      401.14890',
'      398.51324      396.07062      393.67014      391.28210      388.89407',
'      386.48993      384.09543      381.71521      379.33502      376.93649',
```

(continues on next page)

(continued from previous page)

```
'      374.52158      372.12875      369.73593      367.33710      364.91223',
'      362.50223      360.09222      357.68222      355.23526      352.78479',
'      350.33453      347.88428      345.40933      342.91129      340.41443',
'      337.91757      335.41299      332.87881      330.34537      327.81189',
'      325.27841      322.72098      320.16324      317.60550      315.04776',
'      312.47726      309.90338      307.32950    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000      483.49008',
'      481.19510    -9999.00000    -9999.00000      474.57571      472.35135',
'      469.95758      467.28613      464.54922      461.54977      458.52478',
'      454.75122      451.11880      447.49637      444.10587      441.46921',
'      438.83936      436.01788      433.33115      430.65463      428.15698',
'      425.67010      423.06299      420.38107      417.60645      414.80569',
'      411.91711      409.14139      406.47668      403.82217      401.18652',
'      398.54047      396.05911      393.65878      391.27075      388.83133',
'      386.36658      383.97324      381.59305      379.19522      376.73703',
'      374.32440      371.93158      369.53842      367.08047      364.65161',
'      362.24161      359.83160      357.37576      354.91522      352.46497',
'      350.01471      347.53934      345.04474      342.51907      340.01938',
'      337.52252      335.01678      332.46320      329.92792      327.39410',
'      324.85907      322.28552      319.72772      317.16998      314.61185',
'      312.02747      309.45187      306.87799    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000      483.51898',
'      481.20566    -9999.00000    -9999.00000      474.58917      472.37445',
'      469.98090      467.39890      464.67493      461.72269      458.75488',
'      455.09210      451.45499      447.92520      444.49496      441.85370',
'      439.08807      436.12601      433.37961      430.68872      428.07544',
'      425.58856      422.97028      420.28244      417.56940      414.76740',
'      411.90576      409.17218      406.51480      403.85983      401.22012',
'      398.56647      396.04761      393.64740      391.23703      388.72760',
'      386.24319      383.85107      381.46652      379.00720      376.54282',
'      374.12723      371.73441      369.28485      366.82693      364.39096',
'      361.98096      359.52850      357.06024      354.59540      352.14514',
'      349.66974      347.17572      344.67587      342.14963      339.62430',
'      337.12744      334.61691      332.06317      329.51096      326.97714',
'      324.43842      321.86401      319.29221      316.73447      314.17282',
'      311.58749      309.00037      306.42648    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000      483.53830',
'      481.21619    -9999.00000    -9999.00000      474.60260      472.39777',
'      470.00421      467.45190      464.80066      461.95279      458.98499',
'      455.44647      451.83588      448.35403      444.88406      442.15829',
'      439.19620      436.23413      433.41370      430.72281      428.08075',
'      425.50702      422.87164      420.18381      417.49170      414.72906',
'      411.93658      409.20297      406.54941      403.89746      401.24612',
'      398.59247      396.05206      393.63605      391.13327      388.62384',
'      386.13464      383.72888      381.27850      378.81921      376.35480',
'      373.93005      371.48767      369.03131      366.57336      364.13034',
'      361.68124      359.21301      356.74475      354.27554      351.80014',
'      349.30612      346.81210      344.30643      341.78018      339.25394',
'      336.73236      334.21783      331.66321      329.10944      326.56018',
'      324.01779      321.44336      318.86819      316.29895      313.73285',
'      311.14752      308.55829      305.97498    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000      502.68698    -9999.00000    -9999.00000    -9999.00000',
 '      492.94501      490.53168      488.20200      485.87241      483.54752',
 '      481.22540      478.99631      476.80179      474.60614      472.40536',
 '      470.00092      467.44864      464.88757      462.11880      459.18192',
 '      455.68262      452.19653      448.71365      445.26804      442.29022',
 '      439.32816      436.36606      433.47083      430.75616      428.11414',
 '      425.47208      422.79523      420.09912      417.40701      414.70880',
 '      411.95901      409.22571      406.57214      403.91852      401.26486',
 '      398.61115      396.06537      393.54865      391.03918      388.52975',
 '      386.02811      383.56161      381.10233      378.64304      376.16620',
 '      373.70139      371.24503      368.78867      366.32004      363.83569',
 '      361.36743      358.89917      356.43091      353.93271      351.43832',
 '      348.94431      346.45029      343.93463      341.40839      338.88214',
 '      336.35590      333.81845      331.26385      328.70926      326.15463',
 '      323.59891      321.02133      318.44617      315.87100      313.29477',
 '      310.70660      308.11734      305.52808    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000      502.67960    -9999.00000    -9999.00000    -9999.00000',
 '      492.95154      490.54175      488.20483      485.87527      483.55035',
 '      481.22089      478.92841      476.73386      474.53802      472.32892',
 '      469.87296      467.31738      464.74518      461.97641      459.15268',
 '      455.71991      452.23367      448.75079      445.52820      442.53683',
 '      439.57474      436.57355      433.63892      430.81177      428.14410',
```

(continues on next page)

(continued from previous page)

```
'      425.50204      422.82568      420.08743      417.38947      414.69086',
'      411.94107      409.21378      406.55594      403.90231      401.24866',
'      398.58420      396.01752      393.50110      390.99167      388.44302',
'      385.90799      383.44235      380.98306      378.49640      375.97476',
'      373.51132      371.05496      368.58661      366.06418      363.56360',
'      361.06174      358.59348      356.12521      353.61084      351.08890',
'      348.59399      346.09906      343.57004      341.02539      338.49915',
'      335.97290      333.42682      330.85550      328.30087      325.74628',
'      323.18350      320.59271      318.01755      315.44235      312.86292',
'      310.26105      307.67178      305.08252    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000      502.67651    -9999.00000    -9999.00000    -9999.00000',
'      492.95862      490.55182      488.21091      485.87811      483.55319',
'      481.21445      478.91614      476.66595      474.46991      472.21078',
'      469.75482      467.18613      464.60278      461.86160      459.04813',
'      455.75723      452.27081      448.95807      445.78839      442.78345',
'      439.79999      436.75522      433.79922      430.95844      428.17407',
'      425.53201      422.83719      420.08368      417.37189      414.67294',
'      411.92685      409.20294      406.53973      403.88614      401.22739',
'      398.55728      395.96967      393.45358      390.92120      388.33789',
'      385.78787      383.32309      380.84518      378.31250      375.78329',
'      373.32126      370.85504      368.33725      365.81857      363.30722',
'      360.78525      358.28778      355.81952      353.29956      350.76245',
'      348.24512      345.75018      343.21545      340.65686      338.11615',
'      335.58990      333.03754      330.45895      327.89252      325.33792',
'      322.76807      320.17435      317.58890      315.01373      312.42661',
'      309.82465      307.22623    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
```

(continues on next page)

(continued from previous page)

```
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000      502.67624    -9999.00000    -9999.00000    -9999.00000',
'      492.96869      490.56186      488.22098      485.88098      483.55020',
'      481.20801      478.90967      476.62070      474.40179      472.09262',
'      469.63666      467.06238      464.46039      461.75705      458.94357',
'      455.79456      452.38794      449.21826      446.04855      443.02646',
'      439.98166      436.93689      433.94589      431.10510      428.30881',
'      425.56198      422.83340      420.07990      417.36462      414.65500',
'      411.91602      409.19211      406.52655      403.86993      401.20044',
'      398.53033      395.92981      393.39938      390.81604      388.23273',
'      385.67310      383.19394      380.66125      378.12857      375.59674',
'      373.12350      370.60571      368.08789      365.56921      363.05081',
'      360.52884      358.00690      355.51379      352.98828      350.45117',
'      347.91406      345.40134      342.86121      340.30261      337.74402',
'      335.20688      332.64828      330.06967      327.49109      324.92950',
'      322.35263      319.75891      317.16458      314.58484      311.99030',
'      309.38834      306.78305    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000      533.26862',
'      530.59131      527.91675      525.24219      522.64404      520.14777',
'      517.66376      515.17981      512.67719      510.17911      507.67731',
'      505.17551      502.66888      500.19955      497.79105      495.38257',
'      492.96930      490.55930      488.21664      485.87573      483.53403',
'      481.19183      478.87350      476.58453      474.28839      471.97617',
'      469.50241      466.92813      464.32388      461.63022      458.84109',
'      455.70776      452.54630      449.37662      446.24841      443.17401',
'      440.12924      437.08447      434.07947      431.22305      428.42676',
'      425.63046      422.84360      420.07574      417.36047      414.64252',
'      411.90680      409.17401      406.50845      403.84091      401.17081',
'      398.49814      395.88986      393.30350      390.72015      388.13684',
'      385.56168      383.02631      380.49362      377.96094      375.41171',
'      372.87555      370.35773      367.83994      365.32071      362.78357',
'      360.26160      357.73965      355.21768      352.67200      350.13416',
'      347.59634      345.05850      342.50217      339.94357      337.38501',
'      334.82651      332.25638      329.67776      327.09915      324.52057',
```

(continues on next page)

(continued from previous page)

```
'      321.93973      319.34198      316.74765      314.15320      311.55841',
'      308.95129      306.34604    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000      533.31281',
'      530.56714      527.89258      525.21802      522.60168      520.10712',
'      517.62323      515.13928      512.63727      510.14975      507.65219',
'      505.15039      502.64371      500.17972      497.76794      495.35947',
'      492.94614      490.53308      488.17596      485.83508      483.49335',
'      481.13742      478.76242      476.47345      474.17618      471.84531',
'      469.33075      466.75327      464.14090      461.44742      458.70651',
'      455.61053      452.44794      449.27823      446.29871      443.23560',
'      440.19080      437.16464      434.18005      431.28787      428.47235',
'      425.67603      422.88898      420.08691      417.35538      414.63739',
'      411.90167      409.15161      406.47205      403.80444      401.13437',
'      398.44061      395.81708      393.23096      390.64761      388.02802',
'      385.43439      382.89963      380.36694      377.80090      375.22382',
'      372.65240      370.11325      367.59546      365.07440      362.50439',
'      359.96701      357.44504      354.92331      352.34833      349.79742',
'      347.25961      344.72226      342.14606      339.57248      337.01392',
'      334.45563      331.87296      329.27917      326.70056      324.12195',
'      321.52982      318.92123      316.32693      313.73236      311.12939',
'      308.51245      305.90717    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
'    -9999.00000    -9999.00000    -9999.00000    -9999.00000      533.35699',
'      530.60095      527.86841      525.19385      522.55933      520.06476',
'      517.58276      515.09875      512.59967      510.11218      507.62708',
'      505.12531      502.62366      500.16129      497.74551      495.33636',
'      492.92203      490.50690      488.14267      485.79440      483.45267',
```

(continues on next page)

(continued from previous page)

```
 '      481.08179      478.70209      476.36234      474.06393      471.68353',
 '      469.16901      466.57840      463.95792      461.27835      458.53748',
 '      455.51331      452.34955      449.31866      446.34897      443.29715',
 '      440.26239      437.26660      434.28067      431.38809      428.52106',
 '      425.72162      422.92261      420.11218      417.35028      414.63226',
 '      411.88647      409.13086      406.43567      403.76801      401.08481',
 '      398.38312      395.74429      393.15842      390.54947      387.91068',
 '      385.30713      382.77298      380.21124      377.61627      375.03592',
 '      372.46118      369.88577      367.35098      364.82541      362.25421',
 '      359.68307      357.15045      354.62405      352.04742      349.46780',
 '      346.92288      344.38339      341.79987      339.20816      336.64282',
 '      334.08371      331.49091      328.88644      326.30194      323.72314',
 '      321.11993      318.50552      315.90619      313.31155      310.69739',
 '      308.07779      305.46832    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000    -9999.00000',
 '    -9999.00000    -9999.00000    -9999.00000    -9999.00000      533.40112',
 '      530.64514      527.88910      525.16968      522.51691      520.02240',
 '      517.54089      515.05823      512.56213      510.07462      507.59085',
 '      505.10019      502.60526      500.14288      497.72711      495.31326',
 '      492.89584      490.48071      488.11649      485.75375      483.39954',
 '      481.02615      478.64645      476.26538      473.93359      471.52176',
 '      469.00723      466.40814      463.77863      461.10928      458.36841',
 '      455.41605      452.33862      449.36893      446.39926      443.36014',
 '      440.36438      437.36859      434.38092      431.48834      428.62131',
 '      425.76718      422.94788      420.13745      417.37100      414.62711',
 '      411.86572      409.11008      406.40848      403.72900      401.02731',
 '      398.32562      395.67496      393.07092      390.43213      387.79333',
 '      385.17984      382.62155      380.02661      377.43164      374.84531',
 '      372.26993      369.69455      367.11914      364.57877      362.00604',
 '      359.43335      356.86176      354.32315      351.74655      349.16693',
 '      346.58731      344.03720      341.45367      338.86197      336.27173',
 '      333.70169      331.10889      328.50443      325.90335      323.31326',
 ...]
```

[13]:
```python
gg.utils.save_zmap_grid(zmap_grid, path='ZMAP_Grid.dat')
```

```
ZMAP Grid successfully saved to file
```

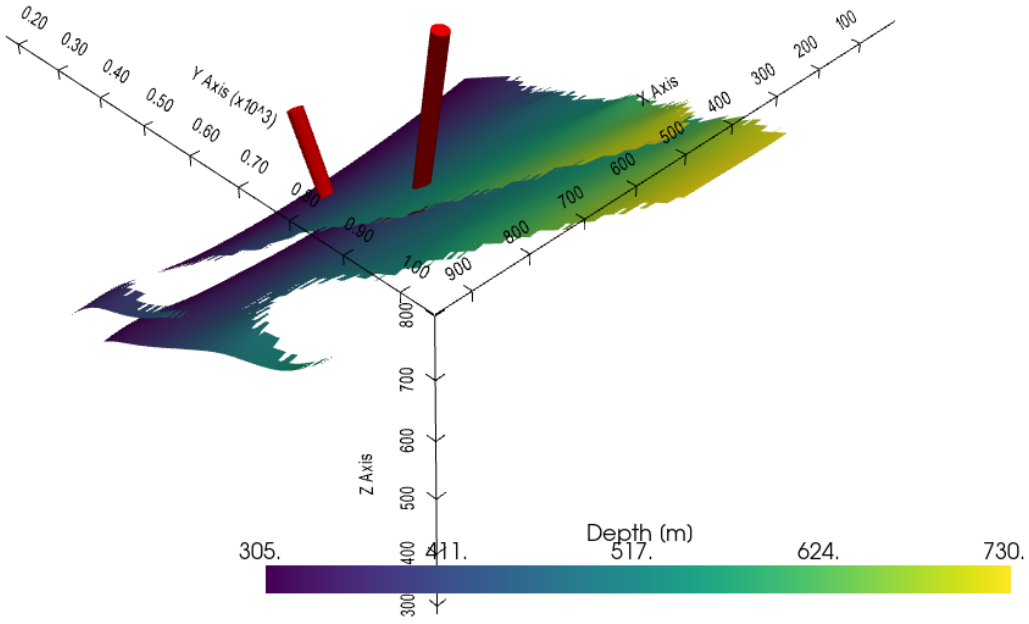
6.55.5 Alternative

Alternatively, the resulting `NumPy` array can also be saved using the `gg.raster.save_as_tiff` function.

[ ]:

6.56 55 Extracting Well Tops from PyVista Meshes

The following notebook illustrates how to extract well tops or in the case of GemPy well bases from PyVista meshes of GemPy models. In particular, a ray tracing algorithm in PyVista is used to extract the intersections of a borehole (PyVista Polydata) with a PyVista mesh. Straight boreholes as well as deviated boreholes can be used.



6.56.1 Set File Paths and download Tutorial Data

If you downloaded the latest GemGIS version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install GemGIS via `pip install gemgis` and import GemGIS using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
import geopandas as gpd
import rasterio
import gempy as gp
import pyvista as pv
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
file_path ='data/55_extracting_well_tops_from_pyvista_meshes/'
gg.download_gemgis_data.download_tutorial_data(filename="55_extracting_well_tops_from_
↪pyvista_meshes.zip", dirpath=file_path)
```

6.56.2 Creating GemPy Model

Loading Topography

[3]:
```
topo_raster = rasterio.open(file_path + 'raster1.tif')
```

Creating Interfaces

[4]:
```
interfaces = gpd.read_file(file_path + 'interfaces1_lines.shp')
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords.head()
```

[4]:

	formation	geometry	X	Y	Z
0	Sand1	POINT (0.256 264.862)	0.26	264.86	353.97
1	Sand1	POINT (10.593 276.734)	10.59	276.73	359.04
2	Sand1	POINT (17.135 289.090)	17.13	289.09	364.28
3	Sand1	POINT (19.150 293.313)	19.15	293.31	364.99
4	Sand1	POINT (27.795 310.572)	27.80	310.57	372.81

Creating Orientations

[5]:
```
orientations = gpd.read_file(file_path + 'orientations1.shp')
orientations = gg.vector.extract_xyz(gdf=orientations, dem=topo_raster)
orientations['polarity'] = 1
orientations
```

[5]:
```
  formation   dip  azimuth                 geometry      X      Y      Z  \
0       Ton 30.50   180.00   POINT (96.471 451.564)  96.47 451.56 440.59
1       Ton 30.50   180.00  POINT (172.761 661.877) 172.76 661.88 556.38
2       Ton 30.50   180.00  POINT (383.074 957.758) 383.07 957.76 729.02
3       Ton 30.50   180.00  POINT (592.356 722.702) 592.36 722.70 601.55
4       Ton 30.50   180.00  POINT (766.586 348.469) 766.59 348.47 378.63
5       Ton 30.50   180.00  POINT (843.907 167.023) 843.91 167.02 282.61
6       Ton 30.50   180.00  POINT (941.846 428.883) 941.85 428.88 423.45
7       Ton 30.50   180.00   POINT (22.142 299.553)  22.14 299.55 368.05

   polarity
0         1
1         1
2         1
3         1
4         1
5         1
6         1
7         1
```

Calculating GemPy Model

[6]:
```
geo_model = gp.create_model('Model1')
gp.init_data(geo_model, [0, 972, 0, 1069, 300, 800], [50, 50, 50],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
gp.map_stack_to_surfaces(geo_model,
                         {'Strata': ('Sand1', 'Ton')},
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_topography(source='gdal', filepath=file_path + 'raster1.tif')
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
sol = gp.compute_model(geo_model, compute_mesh=True)
```

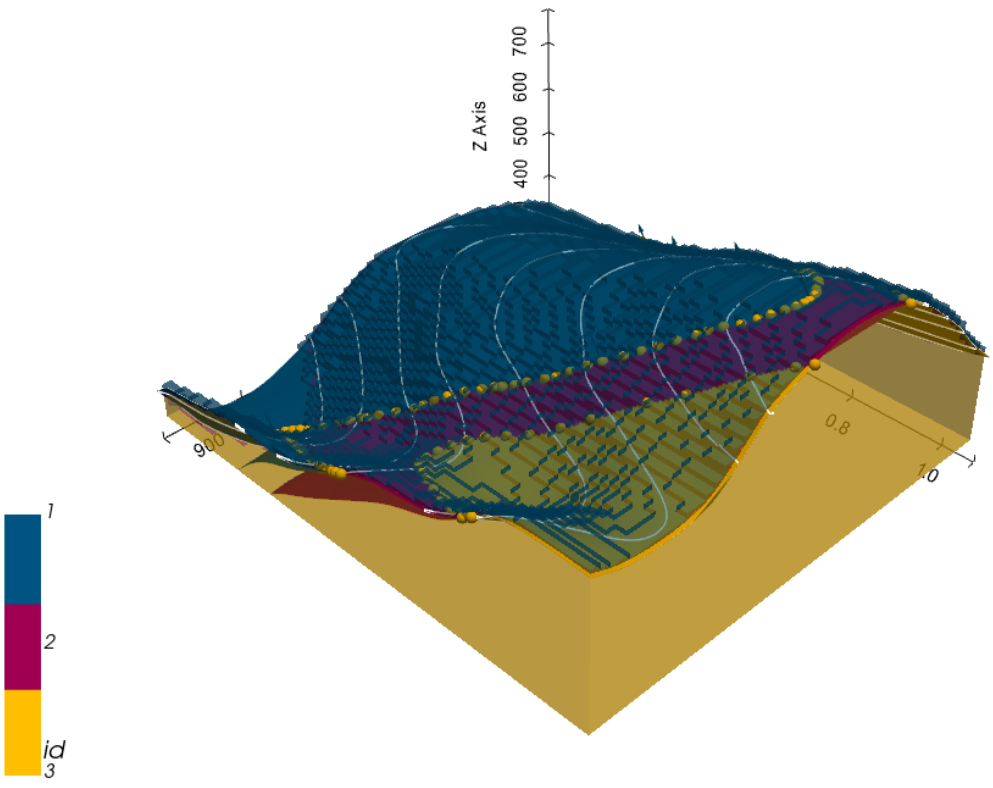
```
Active grids: ['regular']
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

(continues on next page)

(continued from previous page)

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              1528.9
$C_o$             55655.83
drift equations       [3]
```

[7]:
```
gpv = gp.plot_3d(geo_model,
                 image=False,
                 show_topography=True,
                 plotter_type='basic',
                 notebook=True,
                 show_lith=True,
                 show_boundaries=True)
```



6.56.3 Extracting Surfaces from the GemPy Model

[8]:
```
mesh = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model, surfaces=[
↪'Sand1', 'Ton'])
mesh
```

[8]:
```
{'Sand1': [PolyData (0x1e4a0b2b3a0)
    N Cells:    4209
    N Points:   2325
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   1.672e+02, 9.497e+02
    Z Bounds:   3.050e+02, 7.250e+02
    N Arrays:   1,
  '#015482'],
 'Ton': [PolyData (0x1e49c0c30a0)
    N Cells:    5396
    N Points:   2891
    X Bounds:   9.720e+00, 9.623e+02
    Y Bounds:   2.799e+02, 1.058e+03
    Z Bounds:   3.050e+02, 7.297e+02
    N Arrays:   1,
  '#9f0052']}
```

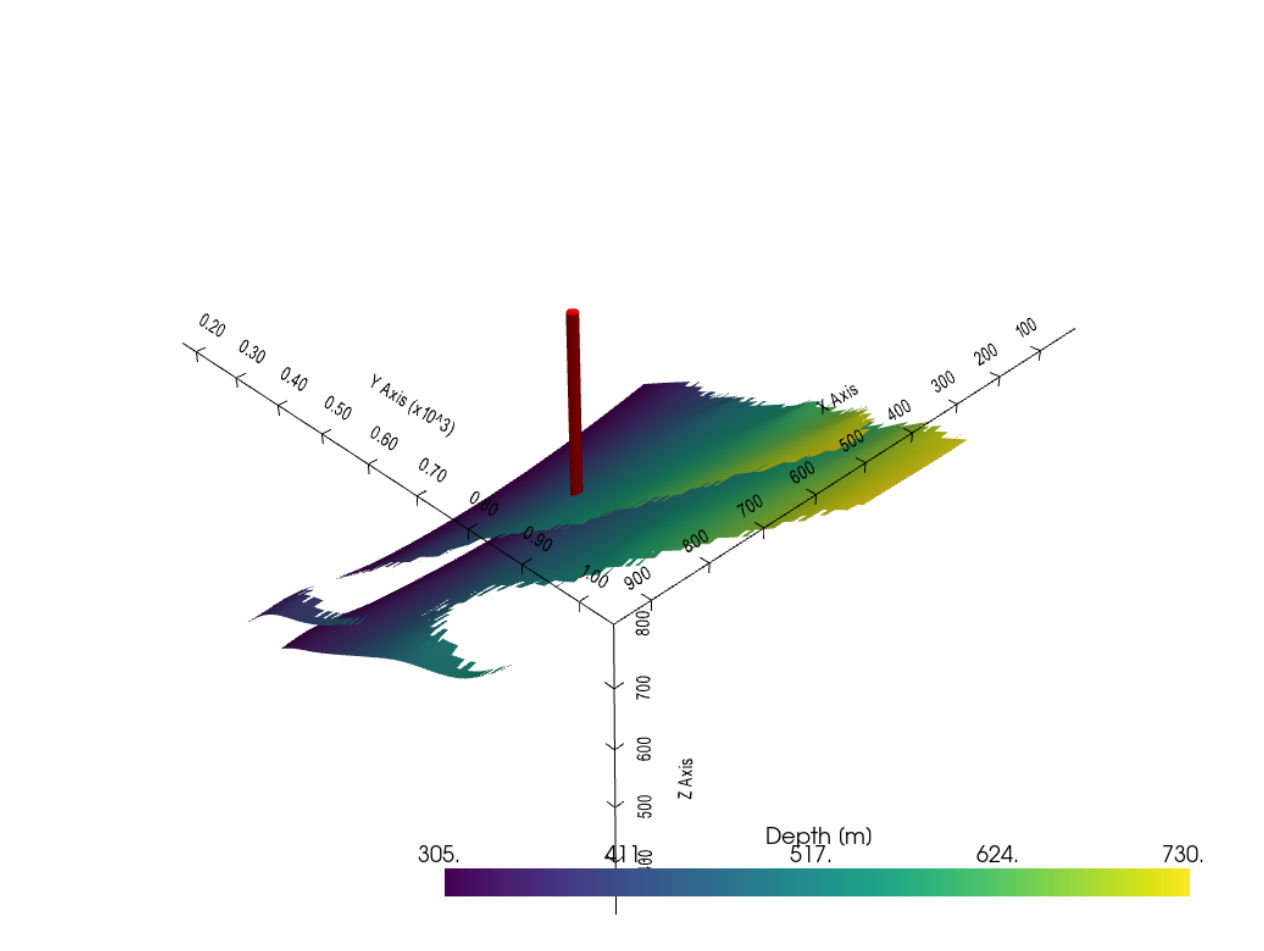
6.56.4 Extracting intersections between boreholes and meshes

First, a PolyData set is created and visualized. The first attempt shows a straight vertical well.

[9]:
```
well = pv.Line((500,500, 800), (500, 500, 300))
well_tube = well.tube(radius=10)
well_tube
```

[9]:
```
PolyData (0x1e49cb57580)
  N Cells:  22
  N Points: 80
  X Bounds: 4.900e+02, 5.100e+02
  Y Bounds: 4.900e+02, 5.100e+02
  Z Bounds: 3.000e+02, 8.000e+02
  N Arrays: 3
```

[10]:
```
p = pv.Plotter(notebook=True)

p.add_mesh(mesh['Sand1'][0], scalars = 'Depth [m]')
p.add_mesh(mesh['Ton'][0], scalars = 'Depth [m]')
p.add_mesh(well_tube, color='red')
p.show_bounds()
p.show()
```



[12]:
```
gg.utils.ray_trace_one_surface(mesh['Sand1'][0],
                               well.points[0],
                               well.points[1])
```

[12]:
```
(array([[500.     , 500.     , 465.47574]], dtype=float32), array([2895]))
```

[13]:
```
gg.utils.ray_trace_multiple_surfaces([mesh['Sand1'][0], mesh['Ton'][0]],
                                     well.points[0],
                                     well.points[1])
```

[13]:
```
[(array([[500.     , 500.     , 465.47574]], dtype=float32), array([2895])),
 (array([[500.     , 500.     , 408.50534]], dtype=float32), array([3181]))]
```

[14]:
```
gg.utils.create_virtual_profile(names_surfaces=list(mesh.keys()),
                                surfaces=[mesh['Sand1'][0], mesh['Ton'][0]],
                                borehole=well)
```

[14]:

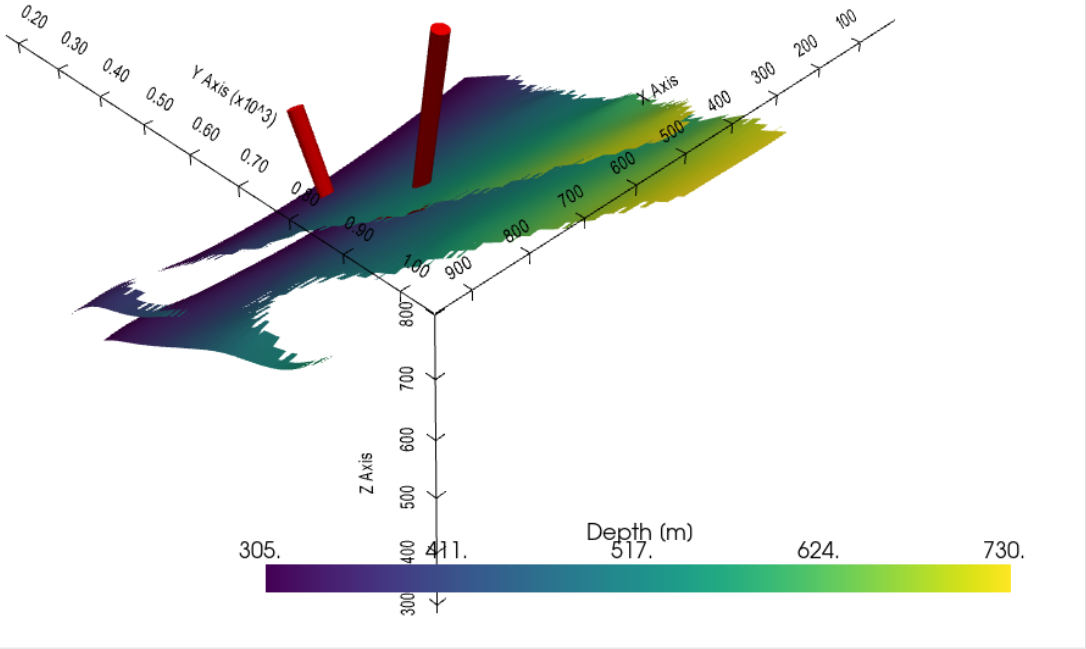
	Surface	Z
0	Sand1	465.48
1	Ton	408.51

The second test will be made with a not so much realistic well path but to illustrate that also deviated boreholes work to extract the intersections. Here, we define a well with three segments (four points).

[15]:
```
points = np.array(((500,600, 800),
                   (500, 550, 460),
                   (500, 350, 300),
                   (500, 250, 500)))

polyline = polyline_from_points(points)
tube = polyline.tube(radius=15)
polyline
```

[15]:
```
PolyData (0x1e49c92c340)
  N Cells:  1
  N Points: 4
  X Bounds: 5.000e+02, 5.000e+02
  Y Bounds: 2.500e+02, 6.000e+02
  Z Bounds: 3.000e+02, 8.000e+02
  N Arrays: 1
```

[16]:
```
p = pv.Plotter(notebook=True)

p.add_mesh(mesh['Sand1'][0], scalars = 'Depth [m]')
p.add_mesh(mesh['Ton'][0], scalars = 'Depth [m]')
p.add_mesh(tube, color='red')
p.show_bounds()
p.show()
```



[17]:
```
gg.utils.create_virtual_profile(names_surfaces=list(mesh.keys()),
                                surfaces=[mesh['Sand1'][0], mesh['Ton'][0]],
                                borehole=polyline)
```

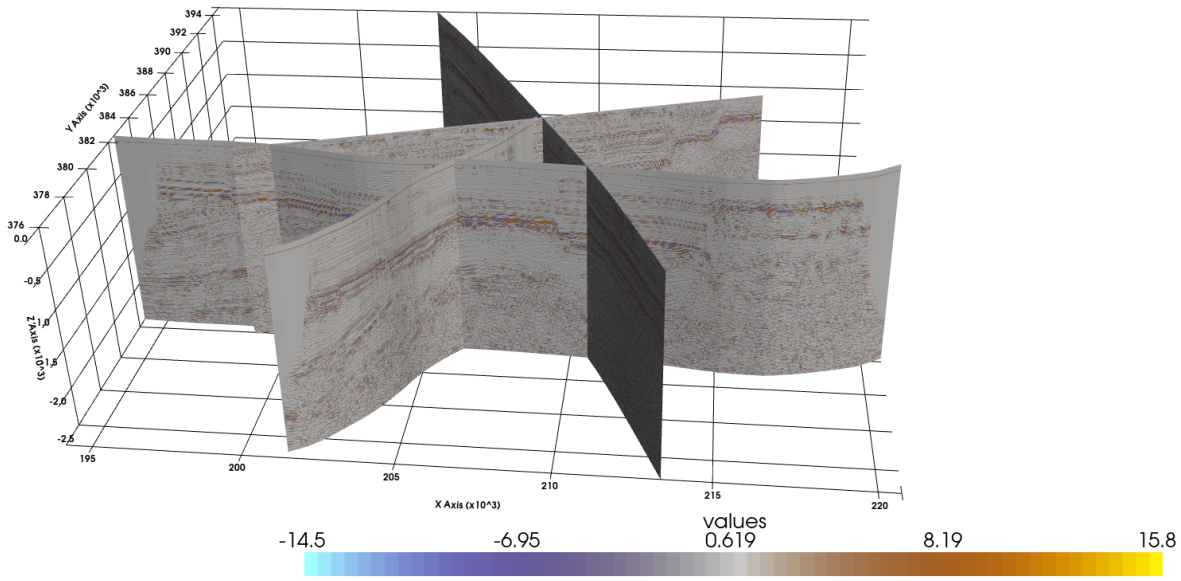
[17]:

	Surface	Z
0	Sand1	497.21
1	Ton	377.33
2	Sand1	357.01
3	Ton	313.96

[ ]:

6.57 56 Displaying Seismic Data in PyVista

The following notebooks illustrates how to display publicly available seismic data loaded with `segysak` and visualized using `PyVista`.



6.57.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

# import gemgis as gg

from segysak.segy import segy_loader

import numpy as np

from matplotlib.colors import ListedColormap

import matplotlib as mpl
import matplotlib.pyplot as plt

import pyvista as pv

from shapely.geometry import LineString
```

6.57.2 Conflict between GeoPandas and Segysak

Importing `gemgis` and its dependencies together with `segysak` will result in an `ImportError`. Therefore, please comment the `segysak` import, import `gemgis`, download the data, restart the notebook, comment the gemgis import and uncomment the `segysak` import to continue running the notebook.

[2]:
```
file_path ='data/56_displaying_seismic_data_in_pyvista/'
# gg.download_gemgis_data.download_tutorial_data(filename="56_displaying_seismic_data_in_
↪pyvista.zip", dirpath=file_path)
```

6.57.3 Loading Seismic Data

The used seismic data can be freely downloaded from https://www.nlog.nl/en/scan-2d-seismic-data and https://www.nlog.nl/en/map-2d-seismic-data. It will be opened using the `segysak` package (https://segysak.readthedocs.io/en/latest/).

We are loading the recently acquired SCAN Seismic Line 29 as `xarray` object. The remaining SCAN Line 20 as well as the Seismic Lines shot within the Framework of the Californie Geothermal project will be loaded directly as vtk file for the final plotting. All data have been prepared the same way as shown here in this notebook.

[3]:
```
scan29 = segy_loader(file_path+'L2EBN2020ASCAN029_PreSTM_final_full_AGC.sgy', vert_
↪domain="TWT")
scan29
```

```
HBox(children=(HTML(value=''), FloatProgress(value=0.0, max=19393.0), HTML(value='')))
```

```
Loading as 2D
```

```
HBox(children=(HTML(value='Converting SEGY'), FloatProgress(value=0.0, max=19393.0),␣
↪HTML(value='')))
```

[3]:
```
<xarray.Dataset>
Dimensions:  (cdp: 19393, twt: 5001)
Coordinates:
  * cdp      (cdp) uint16 100 101 102 103 104 ... 19488 19489 19490 19491 19492
  * twt      (twt) float64 0.0 2.0 4.0 6.0 ... 9.996e+03 9.998e+03 1e+04
    cdp_x    (cdp) float32 1.697e+05 1.697e+05 1.697e+05 ... 2.164e+05 2.164e+05
    cdp_y    (cdp) float32 3.753e+05 3.753e+05 3.753e+05 ... 3.874e+05 3.874e+05
Data variables:
    data     (cdp, twt) float32 0.0 0.03689 -0.05879 -0.3133 ... 0.0 0.0 0.0 0.0
Attributes: (12/13)
    ns:                   None
    sample_rate:          2.0
    text:                 C01 CLIENT: EBN              AREA: SCAN AREAS, NETHE...
    measurement_system:   m
    d3_domain:            None
    epsg:                 None
    ...                   ...
    corner_points_xy:     None
    source_file:          L2EBN2020ASCAN029_PreSTM_final_full_AGC.sgy
    srd:                  None
    datatype:             None
```

(continues on next page)

(continued from previous page)

```
    percentiles:         [-4.339022828226371, -3.723384707669676, -0.46175039...
    coord_scalar:        -100.0
```

6.57.4 Converting xarray to DataFrame

The `xarray` datasets are converted to Pandas DataFrames using the built-in `to_dataframe()` function. The DataFrame contains the X and Y data for every CDP as well as the data value for every sample.

[4]:
```
df_scan29 = scan29.to_dataframe()
df_scan29
```

[4]:
```
                   data          cdp_x        cdp_y
cdp   twt
100   0.0      0.000000  169653.796875  375267.0000
      2.0      0.036886  169653.796875  375267.0000
      4.0     -0.058794  169653.796875  375267.0000
      6.0     -0.313349  169653.796875  375267.0000
      8.0     -0.186058  169653.796875  375267.0000
...                ...           ...          ...
19492 9992.0   0.000000  216394.500000  387366.1875
      9994.0   0.000000  216394.500000  387366.1875
      9996.0   0.000000  216394.500000  387366.1875
      9998.0   0.000000  216394.500000  387366.1875
      10000.0  0.000000  216394.500000  387366.1875

[96984393 rows x 3 columns]
```

6.57.5 Define Seismic colorbar

The seismic colorbar was obtained from https://github.com/lperozzi/Seismic_colormaps/blob/master/colormaps.py and will be used to display the seismic properly in `PyVista`.

[5]:
```
seismic = np.array([[0.63137255, 1.        , 1.        ],
       [0.62745098, 0.97647059, 0.99607843],
       [0.61960784, 0.95686275, 0.98823529],
       [0.61568627, 0.93333333, 0.98431373],
       [0.60784314, 0.91372549, 0.98039216],
       [0.60392157, 0.89019608, 0.97254902],
       [0.59607843, 0.87058824, 0.96862745],
       [0.59215686, 0.85098039, 0.96078431],
       [0.58431373, 0.83137255, 0.95686275],
       [0.58039216, 0.81568627, 0.94901961],
       [0.57254902, 0.79607843, 0.94117647],
       [0.56862745, 0.77647059, 0.9372549 ],
       [0.56078431, 0.76078431, 0.92941176],
       [0.55686275, 0.74509804, 0.92156863],
       [0.54901961, 0.72941176, 0.91764706],
       [0.54509804, 0.70980392, 0.90980392],
       [0.5372549 , 0.69411765, 0.90196078],
       [0.53333333, 0.68235294, 0.89803922],
```

(continues on next page)

(continued from previous page)

```
       [0.5254902 , 0.66666667, 0.89019608],
       [0.52156863, 0.65098039, 0.88235294],
       [0.51372549, 0.63921569, 0.87843137],
       [0.50980392, 0.62352941, 0.87058824],
       [0.50196078, 0.61176471, 0.8627451 ],
       [0.49803922, 0.59607843, 0.85490196],
       [0.49411765, 0.58431373, 0.85098039],
       [0.48627451, 0.57254902, 0.84313725],
       [0.48235294, 0.56078431, 0.83529412],
       [0.47843137, 0.54901961, 0.82745098],
       [0.4745098 , 0.54117647, 0.82352941],
       [0.46666667, 0.52941176, 0.81568627],
       [0.4627451 , 0.51764706, 0.80784314],
       [0.45882353, 0.50980392, 0.8       ],
       [0.45490196, 0.49803922, 0.79607843],
       [0.45098039, 0.49019608, 0.78823529],
       [0.44705882, 0.48235294, 0.78039216],
       [0.44313725, 0.4745098 , 0.77254902],
       [0.43921569, 0.46666667, 0.76862745],
       [0.43529412, 0.45882353, 0.76078431],
       [0.43529412, 0.45098039, 0.75294118],
       [0.43137255, 0.44313725, 0.74901961],
       [0.42745098, 0.43529412, 0.74117647],
       [0.42352941, 0.43137255, 0.7372549 ],
       [0.42352941, 0.42352941, 0.72941176],
       [0.41960784, 0.41960784, 0.72156863],
       [0.41960784, 0.41176471, 0.71764706],
       [0.41568627, 0.40784314, 0.70980392],
       [0.41568627, 0.4       , 0.70588235],
       [0.41176471, 0.39607843, 0.69803922],
       [0.41176471, 0.39215686, 0.69411765],
       [0.41176471, 0.38823529, 0.68627451],
       [0.40784314, 0.38431373, 0.68235294],
       [0.40784314, 0.38039216, 0.67843137],
       [0.40784314, 0.38039216, 0.67058824],
       [0.40784314, 0.37647059, 0.66666667],
       [0.40784314, 0.37254902, 0.6627451 ],
       [0.40392157, 0.37254902, 0.65490196],
       [0.40392157, 0.36862745, 0.65098039],
       [0.40392157, 0.36862745, 0.64705882],
       [0.40392157, 0.36470588, 0.64313725],
       [0.40784314, 0.36470588, 0.63529412],
       [0.40784314, 0.36470588, 0.63137255],
       [0.40784314, 0.36078431, 0.62745098],
       [0.40784314, 0.36078431, 0.62352941],
       [0.40784314, 0.36078431, 0.61960784],
       [0.40784314, 0.36078431, 0.61568627],
       [0.41176471, 0.36078431, 0.61176471],
       [0.41176471, 0.36078431, 0.60784314],
       [0.41176471, 0.36470588, 0.60392157],
       [0.41568627, 0.36470588, 0.60392157],
       [0.41568627, 0.36470588, 0.6       ],
```

(continues on next page)

(continued from previous page)

```
        [0.41960784, 0.36862745, 0.59607843],
        [0.41960784, 0.36862745, 0.59215686],
        [0.42352941, 0.36862745, 0.59215686],
        [0.42352941, 0.37254902, 0.58823529],
        [0.42745098, 0.37647059, 0.58431373],
        [0.43137255, 0.37647059, 0.58431373],
        [0.43137255, 0.38039216, 0.58039216],
        [0.43529412, 0.38431373, 0.58039216],
        [0.43921569, 0.38823529, 0.57647059],
        [0.44313725, 0.39215686, 0.57647059],
        [0.44705882, 0.39607843, 0.57647059],
        [0.44705882, 0.4       , 0.57254902],
        [0.45098039, 0.40392157, 0.57254902],
        [0.45490196, 0.40784314, 0.57254902],
        [0.45882353, 0.41176471, 0.57254902],
        [0.4627451 , 0.41568627, 0.57254902],
        [0.46666667, 0.41960784, 0.57254902],
        [0.47058824, 0.42745098, 0.57254902],
        [0.4745098 , 0.43137255, 0.57254902],
        [0.48235294, 0.43529412, 0.57254902],
        [0.48627451, 0.44313725, 0.57254902],
        [0.49019608, 0.44705882, 0.57254902],
        [0.49411765, 0.45490196, 0.57254902],
        [0.50196078, 0.4627451 , 0.57647059],
        [0.50588235, 0.46666667, 0.57647059],
        [0.50980392, 0.4745098 , 0.58039216],
        [0.51764706, 0.48235294, 0.58039216],
        [0.52156863, 0.49019608, 0.58431373],
        [0.52941176, 0.49803922, 0.58431373],
        [0.53333333, 0.50196078, 0.58823529],
        [0.54117647, 0.50980392, 0.59215686],
        [0.54509804, 0.51764706, 0.59607843],
        [0.55294118, 0.52941176, 0.59607843],
        [0.56078431, 0.5372549 , 0.6       ],
        [0.56862745, 0.54509804, 0.60392157],
        [0.57647059, 0.55294118, 0.61176471],
        [0.58039216, 0.56078431, 0.61568627],
        [0.58823529, 0.57254902, 0.61960784],
        [0.59607843, 0.58039216, 0.62352941],
        [0.60392157, 0.58823529, 0.63137255],
        [0.61176471, 0.6       , 0.63529412],
        [0.62352941, 0.60784314, 0.64313725],
        [0.63137255, 0.61960784, 0.64705882],
        [0.63921569, 0.62745098, 0.65490196],
        [0.64705882, 0.63921569, 0.6627451 ],
        [0.65882353, 0.65098039, 0.67058824],
        [0.66666667, 0.65882353, 0.67843137],
        [0.67843137, 0.67058824, 0.68627451],
        [0.68627451, 0.68235294, 0.69411765],
        [0.69803922, 0.69411765, 0.70196078],
        [0.70588235, 0.70588235, 0.71372549],
        [0.71764706, 0.71372549, 0.72156863],
```

(continues on next page)

(continued from previous page)

```
        [0.72941176, 0.7254902 , 0.73333333],
        [0.74117647, 0.7372549 , 0.74117647],
        [0.75294118, 0.74901961, 0.75294118],
        [0.76470588, 0.76470588, 0.76470588],
        [0.77647059, 0.77647059, 0.77647059],
        [0.78823529, 0.78823529, 0.78823529],
        [0.79215686, 0.78823529, 0.78039216],
        [0.78431373, 0.77254902, 0.76078431],
        [0.77647059, 0.76078431, 0.74117647],
        [0.76862745, 0.74901961, 0.7254902 ],
        [0.76078431, 0.73333333, 0.70588235],
        [0.75294118, 0.72156863, 0.68627451],
        [0.74509804, 0.70980392, 0.67058824],
        [0.74117647, 0.69803922, 0.65490196],
        [0.73333333, 0.68627451, 0.63529412],
        [0.72941176, 0.6745098 , 0.61960784],
        [0.72156863, 0.66666667, 0.60392157],
        [0.71764706, 0.65490196, 0.58823529],
        [0.70980392, 0.64313725, 0.57254902],
        [0.70588235, 0.63137255, 0.55686275],
        [0.70196078, 0.62352941, 0.54509804],
        [0.69411765, 0.61176471, 0.52941176],
        [0.69019608, 0.60392157, 0.51372549],
        [0.68627451, 0.59215686, 0.50196078],
        [0.68235294, 0.58431373, 0.48627451],
        [0.67843137, 0.57647059, 0.4745098 ],
        [0.6745098 , 0.56470588, 0.4627451 ],
        [0.67058824, 0.55686275, 0.45098039],
        [0.66666667, 0.54901961, 0.43921569],
        [0.66666667, 0.54117647, 0.42745098],
        [0.6627451 , 0.53333333, 0.41568627],
        [0.65882353, 0.5254902 , 0.40392157],
        [0.65490196, 0.51764706, 0.39215686],
        [0.65490196, 0.50980392, 0.38039216],
        [0.65098039, 0.50196078, 0.36862745],
        [0.64705882, 0.49803922, 0.36078431],
        [0.64705882, 0.49019608, 0.34901961],
        [0.64313725, 0.48235294, 0.34117647],
        [0.64313725, 0.47843137, 0.32941176],
        [0.64313725, 0.47058824, 0.32156863],
        [0.63921569, 0.46666667, 0.31372549],
        [0.63921569, 0.45882353, 0.30196078],
        [0.63921569, 0.45490196, 0.29411765],
        [0.63529412, 0.45098039, 0.28627451],
        [0.63529412, 0.44313725, 0.27843137],
        [0.63529412, 0.43921569, 0.27058824],
        [0.63529412, 0.43529412, 0.2627451 ],
        [0.63529412, 0.43137255, 0.25490196],
        [0.63529412, 0.42745098, 0.24705882],
        [0.63529412, 0.42352941, 0.23921569],
        [0.63529412, 0.41960784, 0.23137255],
        [0.63529412, 0.41568627, 0.22745098],
```

(continues on next page)

(continued from previous page)

```
        [0.63529412, 0.41176471, 0.21960784],
        [0.63529412, 0.40784314, 0.21176471],
        [0.63921569, 0.40392157, 0.20784314],
        [0.63921569, 0.40392157, 0.2       ],
        [0.63921569, 0.4       , 0.19607843],
        [0.63921569, 0.39607843, 0.18823529],
        [0.64313725, 0.39607843, 0.18431373],
        [0.64313725, 0.39607843, 0.17647059],
        [0.64705882, 0.39215686, 0.17254902],
        [0.64705882, 0.39215686, 0.16862745],
        [0.65098039, 0.38823529, 0.16078431],
        [0.65098039, 0.38823529, 0.15686275],
        [0.65490196, 0.38823529, 0.15294118],
        [0.65490196, 0.38823529, 0.14901961],
        [0.65882353, 0.38823529, 0.14117647],
        [0.6627451 , 0.38823529, 0.1372549 ],
        [0.66666667, 0.38823529, 0.13333333],
        [0.66666667, 0.38823529, 0.12941176],
        [0.67058824, 0.38823529, 0.1254902 ],
        [0.6745098 , 0.39215686, 0.12156863],
        [0.67843137, 0.39215686, 0.11764706],
        [0.68235294, 0.39215686, 0.11372549],
        [0.68627451, 0.39607843, 0.10980392],
        [0.69019608, 0.39607843, 0.10588235],
        [0.69411765, 0.4       , 0.10196078],
        [0.69803922, 0.4       , 0.09803922],
        [0.70196078, 0.40392157, 0.09411765],
        [0.70588235, 0.40784314, 0.09019608],
        [0.70980392, 0.41176471, 0.08627451],
        [0.71372549, 0.41568627, 0.08235294],
        [0.71764706, 0.41960784, 0.07843137],
        [0.7254902 , 0.42352941, 0.0745098 ],
        [0.72941176, 0.42745098, 0.07058824],
        [0.73333333, 0.43137255, 0.06666667],
        [0.7372549 , 0.43529412, 0.0627451 ],
        [0.74509804, 0.43921569, 0.05882353],
        [0.74901961, 0.44705882, 0.05882353],
        [0.75294118, 0.45098039, 0.05490196],
        [0.76078431, 0.45882353, 0.05098039],
        [0.76470588, 0.4627451 , 0.04705882],
        [0.77254902, 0.47058824, 0.04313725],
        [0.77647059, 0.47843137, 0.03921569],
        [0.78431373, 0.48627451, 0.03529412],
        [0.78823529, 0.49411765, 0.03137255],
        [0.79607843, 0.50196078, 0.02745098],
        [0.8       , 0.50980392, 0.02352941],
        [0.80784314, 0.51764706, 0.01960784],
        [0.81176471, 0.5254902 , 0.01568627],
        [0.81960784, 0.53333333, 0.01568627],
        [0.82352941, 0.54509804, 0.01176471],
        [0.83137255, 0.55294118, 0.00784314],
        [0.83921569, 0.56078431, 0.00392157],
```

(continues on next page)

(continued from previous page)

```
       [0.84313725, 0.57254902, 0.00392157],
       [0.85098039, 0.58431373, 0.        ],
       [0.85490196, 0.59215686, 0.        ],
       [0.8627451 , 0.60392157, 0.        ],
       [0.86666667, 0.61568627, 0.        ],
       [0.8745098 , 0.62745098, 0.        ],
       [0.88235294, 0.63921569, 0.        ],
       [0.88627451, 0.65098039, 0.        ],
       [0.89411765, 0.6627451 , 0.        ],
       [0.89803922, 0.67843137, 0.        ],
       [0.90588235, 0.69019608, 0.        ],
       [0.91372549, 0.70196078, 0.        ],
       [0.91764706, 0.71764706, 0.        ],
       [0.9254902 , 0.73333333, 0.        ],
       [0.92941176, 0.74509804, 0.        ],
       [0.93333333, 0.76078431, 0.        ],
       [0.94117647, 0.77647059, 0.        ],
       [0.94509804, 0.79215686, 0.        ],
       [0.95294118, 0.80784314, 0.        ],
       [0.95686275, 0.82352941, 0.        ],
       [0.96078431, 0.83921569, 0.        ],
       [0.96862745, 0.85490196, 0.        ],
       [0.97254902, 0.87058824, 0.        ],
       [0.97647059, 0.89019608, 0.        ],
       [0.98039216, 0.90588235, 0.        ],
       [0.98431373, 0.9254902 , 0.        ],
       [0.98823529, 0.94509804, 0.        ],
       [0.99215686, 0.96078431, 0.        ],
       [0.99607843, 0.98039216, 0.        ],
       [1.        , 1.        , 0.        ]])

cmap_seismic = ListedColormap(seismic)


# Adapted from https://matplotlib.org/stable/tutorials/colors/colormaps.html
gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))

fig, ax = plt.subplots(nrows=1, figsize=(6, 1));
fig.subplots_adjust(top=0.5, bottom=0.15,
                    left=0.2, right=1);
ax.set_title('Seismic Colorbar', fontsize=14);


ax.imshow(gradient, aspect='auto', cmap=cmap_seismic);

# Turn off *all* ticks & spines, not just the ones with colormaps.
ax.set_axis_off();
```

6.57.6 Define Batlow Colorbar

The Batlow colorbar was adapted from https://github.com/callumrollo/cmcrameri/blob/master/cmcrameri/cmaps/batlow.txt-

From https://www.fabiocrameri.ch/colourmaps/:

- Fairly representing data - The colour gradients are perceptually uniform and ordered to represent data both fairly, without visual distortion, and intuitively
- Universally readable - The colour combinations are readable both by colour-vision deficient and colour-blind people, and even when printed in black&white
- Citable & reproducible - The colour maps and their diagnostics are permanently archived and versioned to enable upgrades and acknowledge developers and contributors

[6]:
```
batlow = np.array([[0.005193,0.098238,0.349842],
[0.009065,0.104487,0.350933],
[0.012963,0.110779,0.351992],
[0.016530,0.116913,0.353070],
[0.019936,0.122985,0.354120],
[0.023189,0.129035,0.355182],
[0.026291,0.135044,0.356210],
[0.029245,0.140964,0.357239],
[0.032053,0.146774,0.358239],
[0.034853,0.152558,0.359233],
[0.037449,0.158313,0.360216],
[0.039845,0.163978,0.361187],
[0.042104,0.169557,0.362151],
[0.044069,0.175053,0.363084],
[0.045905,0.180460,0.364007],
[0.047665,0.185844,0.364915],
[0.049378,0.191076,0.365810],
[0.050795,0.196274,0.366684],
[0.052164,0.201323,0.367524],
[0.053471,0.206357,0.368370],
[0.054721,0.211234,0.369184],
[0.055928,0.216046,0.369974],
[0.057033,0.220754,0.370750],
[0.058032,0.225340,0.371509],
[0.059164,0.229842,0.372252],
[0.060167,0.234299,0.372978],
[0.061052,0.238625,0.373691],
[0.062060,0.242888,0.374386],
[0.063071,0.247085,0.375050],
[0.063982,0.251213,0.375709],
[0.064936,0.255264,0.376362],
[0.065903,0.259257,0.376987],
[0.066899,0.263188,0.377594],
[0.067921,0.267056,0.378191],
[0.069002,0.270922,0.378774],
[0.070001,0.274713,0.379342],
[0.071115,0.278497,0.379895],
[0.072192,0.282249,0.380434],
[0.073440,0.285942,0.380957],
```

(continues on next page)

(continued from previous page)

```
[0.074595,0.289653,0.381452],
[0.075833,0.293321,0.381922],
[0.077136,0.296996,0.382376],
[0.078517,0.300622,0.382814],
[0.079984,0.304252,0.383224],
[0.081553,0.307858,0.383598],
[0.083082,0.311461,0.383936],
[0.084778,0.315043,0.384240],
[0.086503,0.318615,0.384506],
[0.088353,0.322167,0.384731],
[0.090281,0.325685,0.384910],
[0.092304,0.329220,0.385040],
[0.094462,0.332712,0.385116],
[0.096618,0.336161,0.385134],
[0.099015,0.339621,0.385090],
[0.101481,0.343036,0.384981],
[0.104078,0.346410,0.384801],
[0.106842,0.349774,0.384548],
[0.109695,0.353098,0.384217],
[0.112655,0.356391,0.383807],
[0.115748,0.359638,0.383310],
[0.118992,0.362849,0.382713],
[0.122320,0.366030,0.382026],
[0.125889,0.369160,0.381259],
[0.129519,0.372238,0.380378],
[0.133298,0.375282,0.379395],
[0.137212,0.378282,0.378315],
[0.141260,0.381240,0.377135],
[0.145432,0.384130,0.375840],
[0.149706,0.386975,0.374449],
[0.154073,0.389777,0.372934],
[0.158620,0.392531,0.371320],
[0.163246,0.395237,0.369609],
[0.167952,0.397889,0.367784],
[0.172788,0.400496,0.365867],
[0.177752,0.403041,0.363833],
[0.182732,0.405551,0.361714],
[0.187886,0.408003,0.359484],
[0.193050,0.410427,0.357177],
[0.198310,0.412798,0.354767],
[0.203676,0.415116,0.352253],
[0.209075,0.417412,0.349677],
[0.214555,0.419661,0.347019],
[0.220112,0.421864,0.344261],
[0.225707,0.424049,0.341459],
[0.231362,0.426197,0.338572],
[0.237075,0.428325,0.335634],
[0.242795,0.430418,0.332635],
[0.248617,0.432493,0.329571],
[0.254452,0.434529,0.326434],
[0.260320,0.436556,0.323285],
[0.266241,0.438555,0.320085],
```

(continues on next page)

(continued from previous page)

```
[0.272168,0.440541,0.316831],
[0.278171,0.442524,0.313552],
[0.284175,0.444484,0.310243],
[0.290214,0.446420,0.306889],
[0.296294,0.448357,0.303509],
[0.302379,0.450282,0.300122],
[0.308517,0.452205,0.296721],
[0.314648,0.454107,0.293279],
[0.320834,0.456006,0.289841],
[0.327007,0.457900,0.286377],
[0.333235,0.459794,0.282937],
[0.339469,0.461685,0.279468],
[0.345703,0.463563,0.275998],
[0.351976,0.465440,0.272492],
[0.358277,0.467331,0.269037],
[0.364589,0.469213,0.265543],
[0.370922,0.471085,0.262064],
[0.377291,0.472952,0.258588],
[0.383675,0.474842,0.255131],
[0.390070,0.476711,0.251665],
[0.396505,0.478587,0.248212],
[0.402968,0.480466,0.244731],
[0.409455,0.482351,0.241314],
[0.415967,0.484225,0.237895],
[0.422507,0.486113,0.234493],
[0.429094,0.488011,0.231096],
[0.435714,0.489890,0.227728],
[0.442365,0.491795,0.224354],
[0.449052,0.493684,0.221074],
[0.455774,0.495585,0.217774],
[0.462539,0.497497,0.214518],
[0.469368,0.499393,0.211318],
[0.476221,0.501314,0.208148],
[0.483123,0.503216,0.205037],
[0.490081,0.505137,0.201976],
[0.497089,0.507058,0.198994],
[0.504153,0.508984,0.196118],
[0.511253,0.510898,0.193296],
[0.518425,0.512822,0.190566],
[0.525637,0.514746,0.187990],
[0.532907,0.516662,0.185497],
[0.540225,0.518584,0.183099],
[0.547599,0.520486,0.180884],
[0.555024,0.522391,0.178854],
[0.562506,0.524293,0.176964],
[0.570016,0.526186,0.175273],
[0.577582,0.528058,0.173775],
[0.585199,0.529927,0.172493],
[0.592846,0.531777,0.171449],
[0.600520,0.533605,0.170648],
[0.608240,0.535423,0.170104],
[0.615972,0.537231,0.169826],
```

(continues on next page)

(continued from previous page)

```
[0.623739,0.539002,0.169814],
[0.631513,0.540752,0.170075],
[0.639301,0.542484,0.170622],
[0.647098,0.544183,0.171465],
[0.654889,0.545863,0.172603],
[0.662691,0.547503,0.174044],
[0.670477,0.549127,0.175747],
[0.678244,0.550712,0.177803],
[0.685995,0.552274,0.180056],
[0.693720,0.553797,0.182610],
[0.701421,0.555294,0.185478],
[0.709098,0.556772,0.188546],
[0.716731,0.558205,0.191851],
[0.724322,0.559628,0.195408],
[0.731878,0.561011,0.199174],
[0.739393,0.562386,0.203179],
[0.746850,0.563725,0.207375],
[0.754268,0.565033,0.211761],
[0.761629,0.566344,0.216322],
[0.768942,0.567630,0.221045],
[0.776208,0.568899,0.225930],
[0.783416,0.570162,0.230962],
[0.790568,0.571421,0.236160],
[0.797665,0.572682,0.241490],
[0.804709,0.573928,0.246955],
[0.811692,0.575187,0.252572],
[0.818610,0.576462,0.258303],
[0.825472,0.577725,0.264197],
[0.832272,0.579026,0.270211],
[0.838999,0.580339,0.276353],
[0.845657,0.581672,0.282631],
[0.852247,0.583037,0.289036],
[0.858747,0.584440,0.295572],
[0.865168,0.585882,0.302255],
[0.871505,0.587352,0.309112],
[0.877741,0.588873,0.316081],
[0.883878,0.590450,0.323195],
[0.889900,0.592087,0.330454],
[0.895809,0.593765,0.337865],
[0.901590,0.595507,0.345429],
[0.907242,0.597319,0.353142],
[0.912746,0.599191,0.360986],
[0.918103,0.601126,0.368999],
[0.923300,0.603137,0.377139],
[0.928323,0.605212,0.385404],
[0.933176,0.607369,0.393817],
[0.937850,0.609582,0.402345],
[0.942332,0.611867,0.411006],
[0.946612,0.614218,0.419767],
[0.950697,0.616649,0.428624],
[0.954574,0.619137,0.437582],
[0.958244,0.621671,0.446604],
```

(continues on next page)

(continued from previous page)

```
[0.961696,0.624282,0.455702],
[0.964943,0.626934,0.464860],
[0.967983,0.629639,0.474057],
[0.970804,0.632394,0.483290],
[0.973424,0.635183,0.492547],
[0.975835,0.638012,0.501826],
[0.978052,0.640868,0.511090],
[0.980079,0.643752,0.520350],
[0.981918,0.646664,0.529602],
[0.983574,0.649590,0.538819],
[0.985066,0.652522,0.547998],
[0.986392,0.655470,0.557142],
[0.987567,0.658422,0.566226],
[0.988596,0.661378,0.575265],
[0.989496,0.664329,0.584246],
[0.990268,0.667280,0.593174],
[0.990926,0.670230,0.602031],
[0.991479,0.673165,0.610835],
[0.991935,0.676091,0.619575],
[0.992305,0.679007,0.628251],
[0.992595,0.681914,0.636869],
[0.992813,0.684815,0.645423],
[0.992967,0.687705,0.653934],
[0.993064,0.690579,0.662398],
[0.993111,0.693451,0.670810],
[0.993112,0.696314,0.679177],
[0.993074,0.699161,0.687519],
[0.993002,0.702006,0.695831],
[0.992900,0.704852,0.704114],
[0.992771,0.707689,0.712380],
[0.992619,0.710530,0.720639],
[0.992447,0.713366,0.728892],
[0.992258,0.716210,0.737146],
[0.992054,0.719049,0.745403],
[0.991837,0.721893,0.753673],
[0.991607,0.724754,0.761959],
[0.991367,0.727614,0.770270],
[0.991116,0.730489,0.778606],
[0.990855,0.733373,0.786976],
[0.990586,0.736265,0.795371],
[0.990307,0.739184,0.803810],
[0.990018,0.742102,0.812285],
[0.989720,0.745039,0.820804],
[0.989411,0.747997,0.829372],
[0.989089,0.750968,0.837979],
[0.988754,0.753949,0.846627],
[0.988406,0.756949,0.855332],
[0.988046,0.759964,0.864078],
[0.987672,0.762996,0.872864],
[0.987280,0.766047,0.881699],
[0.986868,0.769105,0.890573],
[0.986435,0.772184,0.899493],
```

(continues on next page)

(continued from previous page)

```
[0.985980,0.775272,0.908448],
[0.985503,0.778378,0.917444],
[0.985002,0.781495,0.926468],
[0.984473,0.784624,0.935531],
[0.983913,0.787757,0.944626],
[0.983322,0.790905,0.953748],
[0.982703,0.794068,0.962895],
[0.982048,0.797228,0.972070],
[0.981354,0.800406,0.981267]])

cmap_batlow = ListedColormap(batlow)


# Adapted from https://matplotlib.org/stable/tutorials/colors/colormaps.html
gradient = np.linspace(0, 1, 256)
gradient = np.vstack((gradient, gradient))

fig, ax = plt.subplots(nrows=1, figsize=(6, 1));
fig.subplots_adjust(top=0.5, bottom=0.15,
                    left=0.2, right=1);
ax.set_title('Batlow Colorbar', fontsize=14);


ax.imshow(gradient, aspect='auto', cmap=cmap_batlow);

# Turn off *all* ticks & spines, not just the ones with colormaps.
ax.set_axis_off();
```

6.57.7 Selecting and reshaping the seismic data

[7]:
```
cdp = 100

# Defining the number of rows
rows = int(len(df_scan29.loc[cdp]))

# Cropping the data
df_scan29_selection = df_scan29.loc[12000:]

# Defining the number of columns
columns = int(len(df_scan29_selection)/rows)

# Extract values from DataFrame
df_scan29_data = df_scan29_selection['data'].values

# Reshape data
df_scan29_data_reshaped = df_scan29_data.reshape(columns, rows)
```

(continues on next page)

(continued from previous page)

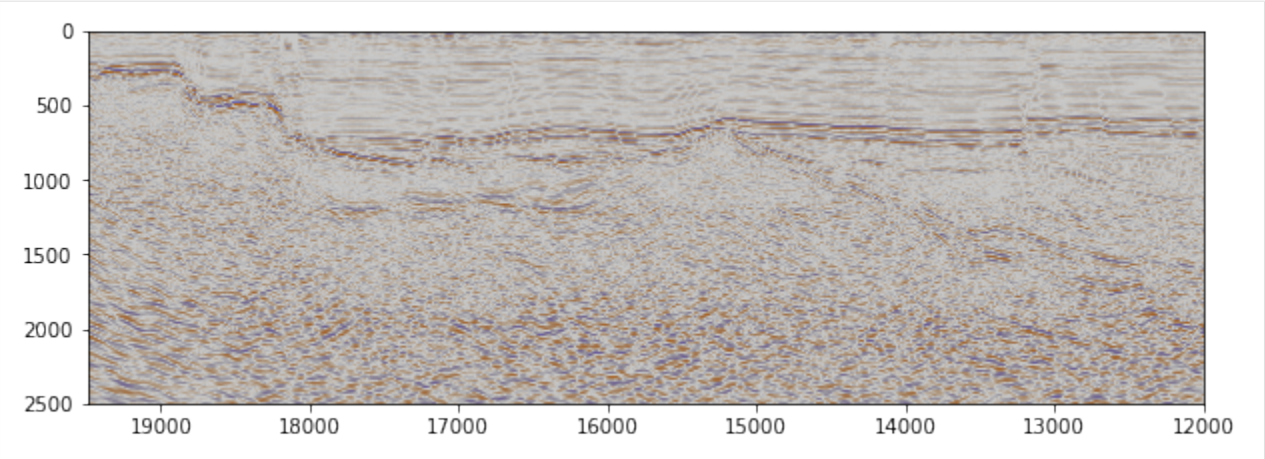
```
# Selecting data only up to 2.5 s TWT
depth_twt = 2500
indices_twt = int((rows-1)/(df_scan29.loc[cdp].index.max()-df_scan29.loc[cdp].index.
↪min())*depth_twt+1)

df_scan29_data_reshaped_selected = df_scan29_data_reshaped[:,:indices_twt]
df_scan29_data_reshaped_selected
```

[7]:
```
array([[ 0.        , -0.6604564 , -0.5132353 , ...,  0.38492066,
         0.18723893, -0.3822595 ],
       [ 0.        , -0.6229209 , -0.5288663 , ...,  0.35308176,
         0.14418834, -0.46200967],
       [ 0.        , -0.58638173, -0.5448488 , ...,  0.3399232 ,
         0.09482116, -0.56471187],
       ...,
       [ 0.        ,  1.5639963 ,  1.7435122 , ...,  1.85919   ,
         1.66117   ,  0.4377218 ],
       [ 0.        ,  1.715455  ,  1.7838211 , ...,  1.9031734 ,
         1.6262188 ,  0.33231133],
       [ 0.        ,  1.8398867 ,  1.790576  , ...,  1.9389725 ,
         1.5917187 ,  0.23935503]], dtype=float32)
```

6.57.8 Plotting Seismic Sections

[8]:
```
# Defining the extent
df_scan29_reset_index = df_scan29_selection.reset_index()
minx = df_scan29_reset_index['cdp'].min()
maxx = df_scan29_reset_index['cdp'].max()

fig, ax = plt.subplots(1, figsize=(10,10))
plt.imshow(np.fliplr(df_scan29_data_reshaped_selected.T),
           cmap=cmap_seismic,
           vmin=-7,
           vmax=7,
           extent=[maxx,
                   minx,
                   depth_twt,
                   df_scan29.loc[cdp].index.min()])
```

[8]:
```
<matplotlib.image.AxesImage at 0x27108431e50>
```



6.57.9 Draping 2D Surface from Line

Drape the 2D seismic data over a mesh along the seismic line. This workflow is adapted from the PyVista Tutorial at https://docs.pyvista.org/examples/00-load/create-surface-draped.html#sphx-glr-examples-00-load-create-surface-draped-py.

[9]:
```
ntraces, nsamples = df_scan29_data_reshaped_selected.shape
print(nsamples, ntraces)
```

```
1251 7493
```

[10]:
```
sample_spacing = 2 # ms - milliseconds
sample_spacing
```

[10]:
```
2
```

[11]:
```
df_scan29_selection[['cdp_x', 'cdp_y']]
```

[11]:
```
                     cdp_x         cdp_y
cdp   twt
12000 0.0      198476.09375  381918.71875
      2.0      198476.09375  381918.71875
      4.0      198476.09375  381918.71875
      6.0      198476.09375  381918.71875
      8.0      198476.09375  381918.71875
...                     ...           ...
19492 9992.0   216394.50000  387366.18750
      9994.0   216394.50000  387366.18750
      9996.0   216394.50000  387366.18750
      9998.0   216394.50000  387366.18750
      10000.0  216394.50000  387366.18750

[37472493 rows x 2 columns]
```

[12]:
```
cdps =  np.unique(df_scan29_selection[['cdp_x', 'cdp_y']].values,axis=0)
print(len(cdps))
cdps
```

```
7493
```

[12]:
```
array([[198476.1 , 381918.72],
       [198478.5 , 381919.28],
       [198480.89, 381920.  ],
       ...,
       [216389.69, 387364.72],
       [216392.1 , 387365.5 ],
       [216394.5 , 387366.2 ]], dtype=float32)
```

[13]:
```
path = np.c_[cdps, np.zeros(len(cdps))]
path
```

[13]:
```
array([[198476.09375 , 381918.71875 ,      0.      ],
       [198478.5     , 381919.28125 ,      0.      ],
       [198480.890625, 381920.      ,      0.      ],
       ...,
       [216389.6875  , 387364.71875 ,      0.      ],
       [216392.09375 , 387365.5     ,      0.      ],
       [216394.5     , 387366.1875  ,      0.      ]])
```

[14]:
```
points = np.repeat(path, nsamples, axis=0)
points
```

[14]:
```
array([[198476.09375, 381918.71875,      0.     ],
       [198476.09375, 381918.71875,      0.     ],
       [198476.09375, 381918.71875,      0.     ],
       ...,
       [216394.5    , 387366.1875 ,      0.     ],
       [216394.5    , 387366.1875 ,      0.     ],
       [216394.5    , 387366.1875 ,      0.     ]])
```

[15]:
```
# repeat the Z locations across
tp = np.arange(0, sample_spacing * nsamples, sample_spacing)
tp = path[:, 2][:, None] - tp
points[:, -1] = tp.ravel()
```

6.57.10 Creating Structured Grid

[16]:
```
grid = pv.StructuredGrid()
grid.points = points
grid.dimensions = nsamples, ntraces, 1

# Add the data array - note the ordering!
grid["values"] = df_scan29_data_reshaped_selected.ravel(order="C")
grid
```

[16]:
```
StructuredGrid (0x271084f7e20)
  N Cells:  9365000
  N Points: 9373743
  X Bounds: 1.985e+05, 2.164e+05
  Y Bounds: 3.819e+05, 3.874e+05
```

(continues on next page)

(continued from previous page)

```
  Z Bounds: -2.500e+03, 0.000e+00
  Dimensions:       1251, 7493, 1
  N Arrays: 1
```

6.57.11 Load Remaining Meshes

[17]:
```
cal01 = pv.read(file_path+'Californie_01.vtk')
cal01
```

[17]:
```
StructuredGrid (0x271084f7820)
  N Cells:  1537500
  N Points: 1539981
  X Bounds: 2.015e+05, 2.086e+05
  Y Bounds: 3.752e+05, 3.851e+05
  Z Bounds: -2.500e+03, 0.000e+00
  Dimensions:       1251, 1231, 1
  N Arrays: 1
```

[18]:
```
cal02 = pv.read(file_path+'Californie_02.vtk')
cal02
```

[18]:
```
StructuredGrid (0x2710850e160)
  N Cells:  3376250
  N Points: 3380202
  X Bounds: 1.941e+05, 2.207e+05
  Y Bounds: 3.808e+05, 3.825e+05
  Z Bounds: -2.500e+03, 0.000e+00
  Dimensions:       1251, 2702, 1
  N Arrays: 1
```

[20]:
```
scan20 = pv.read(file_path+'Scan_Line_20.vtk')
scan20
```

[20]:
```
StructuredGrid (0x27108732820)
  N Cells:  11191250
  N Points: 11201454
  X Bounds: 2.034e+05, 2.134e+05
  Y Bounds: 3.749e+05, 3.949e+05
  Z Bounds: -2.500e+03, 0.000e+00
  Dimensions:       1251, 8954, 1
  N Arrays: 1
```

[21]:
```
base_tertiary = pv.read(file_path + 'Base_Tertiary.vtk')
base_tertiary
```

[21]:
```
StructuredGrid (0x271086afa60)
  N Cells:  2895750
  N Points: 2899288
  X Bounds: -4.242e+04, 2.793e+05
  Y Bounds: 3.056e+05, 8.681e+05
  Z Bounds: -1.751e+03, 1.000e+00
```

(continues on next page)

(continued from previous page)

```
  Dimensions:       2251, 1288, 1
  N Arrays: 1
```

[23]:
```
outline_nrw_clipped = pv.read(file_path + 'Outline_NRW_Mesh.vtk')
outline_nrw_clipped
```

[23]:
```
PolyData (0x2710850e1c0)
  N Cells:  966
  N Points: 968
  X Bounds: 2.062e+05, 2.134e+05
  Y Bounds: 3.700e+05, 4.000e+05
  Z Bounds: -2.500e+03, 0.000e+00
  N Arrays: 1
```

[25]:
```
top_dinant = pv.read(file_path + 'Top_Dinant.vtk')
top_dinant
```

[25]:
```
StructuredGrid (0x27108477c40)
  N Cells:  1899920
  N Points: 1902687
  X Bounds: -3.485e+04, 2.826e+05
  Y Bounds: 3.000e+05, 6.740e+05
  Z Bounds: -5.122e+03, -1.322e+02
  Dimensions:       1497, 1271, 1
  N Arrays: 1
```

6.57.12 Clipping meshes and calculating contours

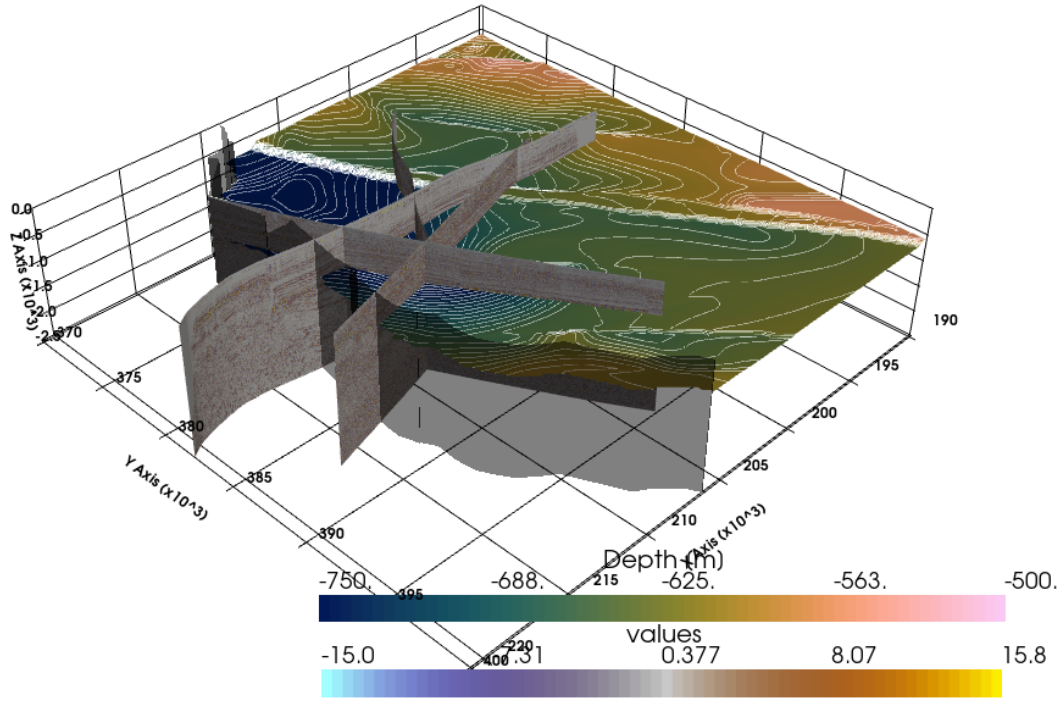
[27]:
```
base_tertiary_clipped = base_tertiary.clip_box([190000, 225000, 370000, 400000, -2500.0,␣
↪100.0], invert=False)
top_dinant_clipped = top_dinant.clip_box([190000, 225000, 370000, 400000, -2500.0, 100.
↪0], invert=False)
```

[28]:
```
base_tertiary_clipped_contours = base_tertiary_clipped.contour(np.linspace(-850, -500,␣
↪50))
base_tertiary_clipped_contours
```

[28]:
```
PolyData (0x2710876f760)
  N Cells:  12771
  N Points: 12856
  X Bounds: 1.900e+05, 2.131e+05
  Y Bounds: 3.700e+05, 4.000e+05
  Z Bounds: -8.143e+02, -5.429e+02
  N Arrays: 1
```

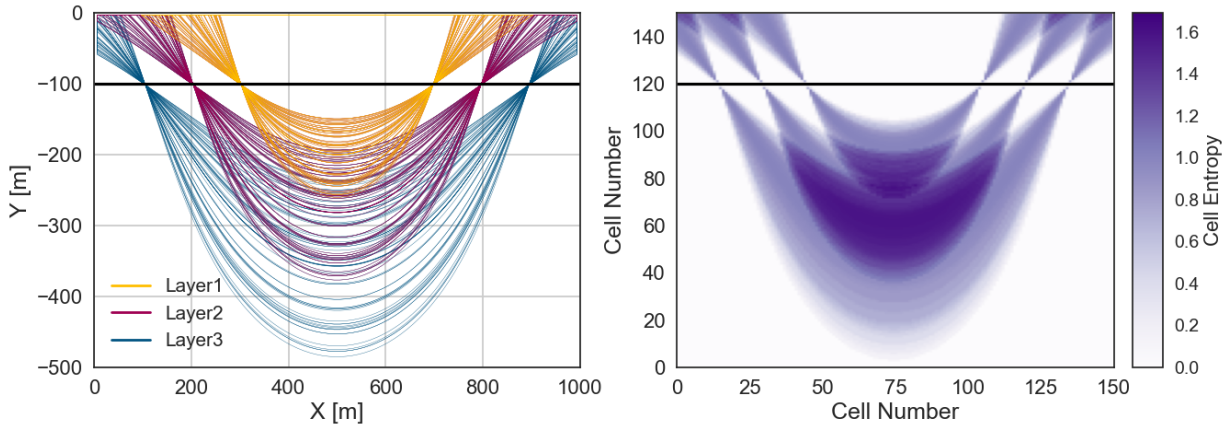
6.57.13 Plotting Seismic Data and Surfaces

[31]:
```
import pyvista as pv

sargs = dict(color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(grid, cmap=cmap_seismic,scalar_bar_args=sargs, clim=[-15,15])
p.add_mesh(cal01, cmap=cmap_seismic)
p.add_mesh(cal02, cmap=cmap_seismic)
p.add_mesh(scan20, cmap=cmap_seismic)
p.add_mesh(base_tertiary_clipped, cmap=cmap_batlow,scalar_bar_args=sargs, clim=[-750,-
↪500])
p.add_mesh(base_tertiary_clipped_contours, color='white')
# p.add_mesh(top_dinant_clipped, cmap='viridis')
p.add_mesh(outline_nrw_clipped, color='black', opacity=0.5)
p.show_grid(color='black')
p.set_background(color='white')
p.set_scale(1,1,3)

p.show()
```



6.58 57 Creating Spaghetti plots in GemPy

This notebook illustrates how to create spaghetti plots in GemPy illustrating the variety of model outcomes when performing probabilistic modeling in GemPy. This notebook was adapted from the work done here: https://github.com/elimh/stochastic-simulations-in-gempy.



6.58.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import geopandas as gpd
import pandas as pd
import gempy as gp
import gemgis as gg
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.configdefaults): install mkl with `conda install mkl-service`: No module␣
↪named 'mkl'
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
file_path ='data/57_creating_spaghettii_plots_in_gempy/'
gg.download_gemgis_data.download_tutorial_data(filename="57_creating_spaghettii_plots_in_
↪gempy.zip", dirpath=file_path)
```

6.58.2 Loading the Interface and Orientations Data

The interface and orientation data is provided for a simple syncline model as Shapefile.

[3]:
```
interfaces = gpd.read_file('data/57_creating_spaghettii_plots_in_gempy/
↪SynclineInterfaces.shp')
orientations = gpd.read_file('data/57_creating_spaghettii_plots_in_gempy/
↪SynclineInterfaces.shp')
```

6.58.3 Deleting columns and dropping geometries

[4]:
```
del interfaces['polarity']
del interfaces['dip']
del interfaces['azimuth']
interfaces['Z'] = -100
orientations['Z'] = -100

interfaces = pd.DataFrame(interfaces.drop(columns='geometry'))

orientations = pd.DataFrame(orientations.drop(columns='geometry'))
```

6.58.4 Inspecting DataFrames

[5]:
```
interfaces.head()
```

[5]:

	X	Y	Z	formation
0	200	100	-100	Layer3
1	400	100	-100	Layer3
2	600	100	-100	Layer3
3	800	100	-100	Layer3
4	200	200	-100	Layer2

[6]:
```
orientations.head()
```

[6]:

	X	Y	Z	formation	polarity	azimuth	dip
0	200	100	-100	Layer3	1	0	45
1	400	100	-100	Layer3	1	0	45
2	600	100	-100	Layer3	1	0	45
3	800	100	-100	Layer3	1	0	45
4	200	200	-100	Layer2	1	0	45

6.58.5 Creating GemPy Model

Initiate Model

A relatively low resolution is chosen to speed up the modeling process.

[7]:
```
geo_model = gp.create_model('Model1')
```

[8]:
```
gp.init_data(geo_model, [0, 1000, 0, 1000, -500,0], [30,30,30],
             surface_points_df = interfaces,
             orientations_df = orientations,
             default_values=True)
geo_model.surfaces
```

```
Active grids: ['regular']
```

[8]:

	surface	series	order_surfaces	color	id
0	Layer3	Default series	1	#015482	1
1	Layer2	Default series	2	#9f0052	2
2	Layer1	Default series	3	#ffbe00	3

Map Stack to Surfaces

[9]:
```
gp.map_stack_to_surfaces(geo_model,
                         {"Strat_Series": ('Layer1', 'Layer2', 'Layer3')},
                         remove_unused_series=True)
geo_model.add_surfaces('basement')
```

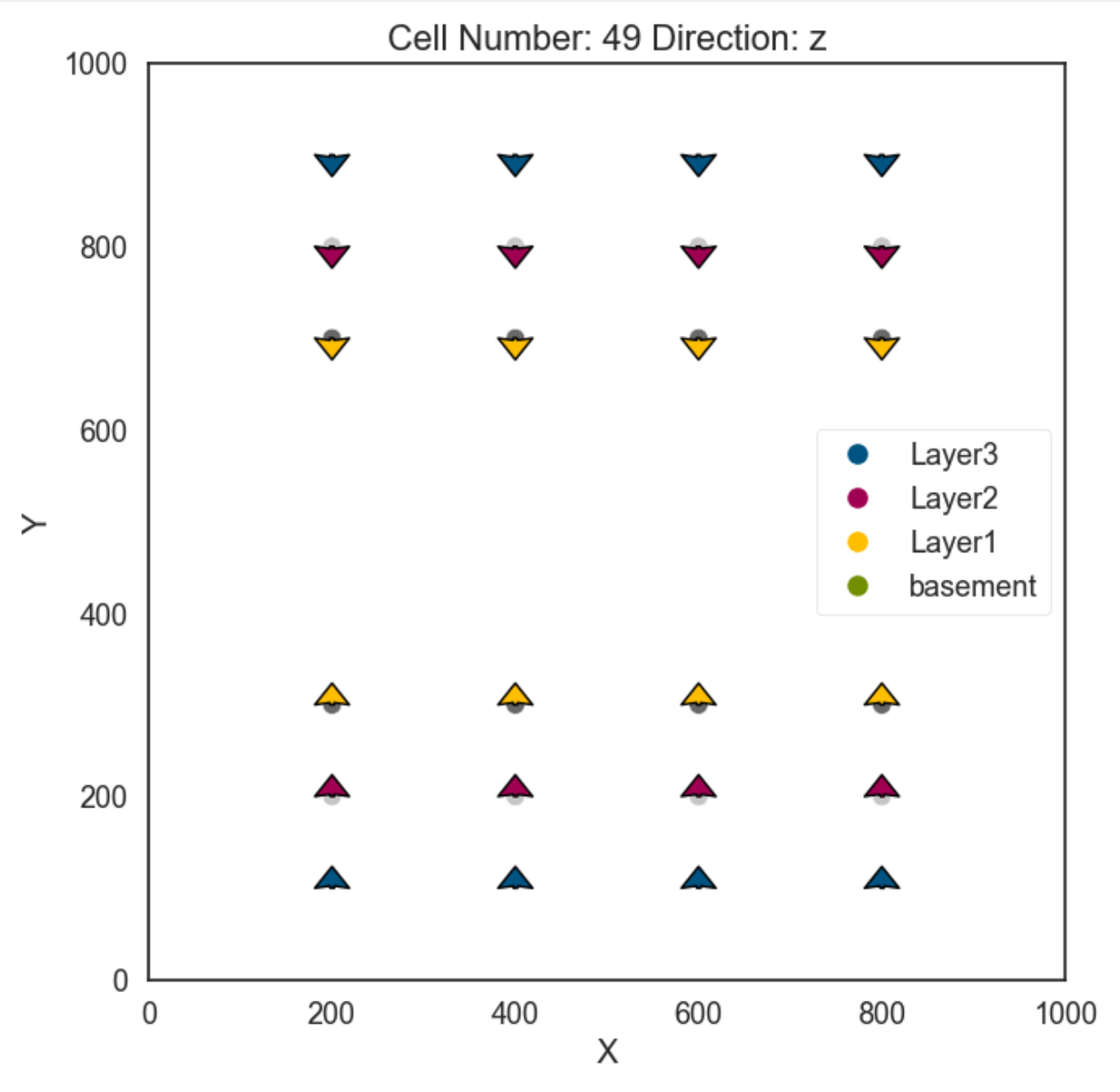
[9]:

	surface	series	order_surfaces	color	id
0	Layer3	Strat_Series	1	#015482	1
1	Layer2	Strat_Series	2	#9f0052	2
2	Layer1	Strat_Series	3	#ffbe00	3
3	basement	Strat_Series	4	#728f02	4

Plot Input Data

[10]:
```
gp.plot_2d(geo_model, direction='z', cell_number = 49)
```

[10]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1fd2af02560>
```



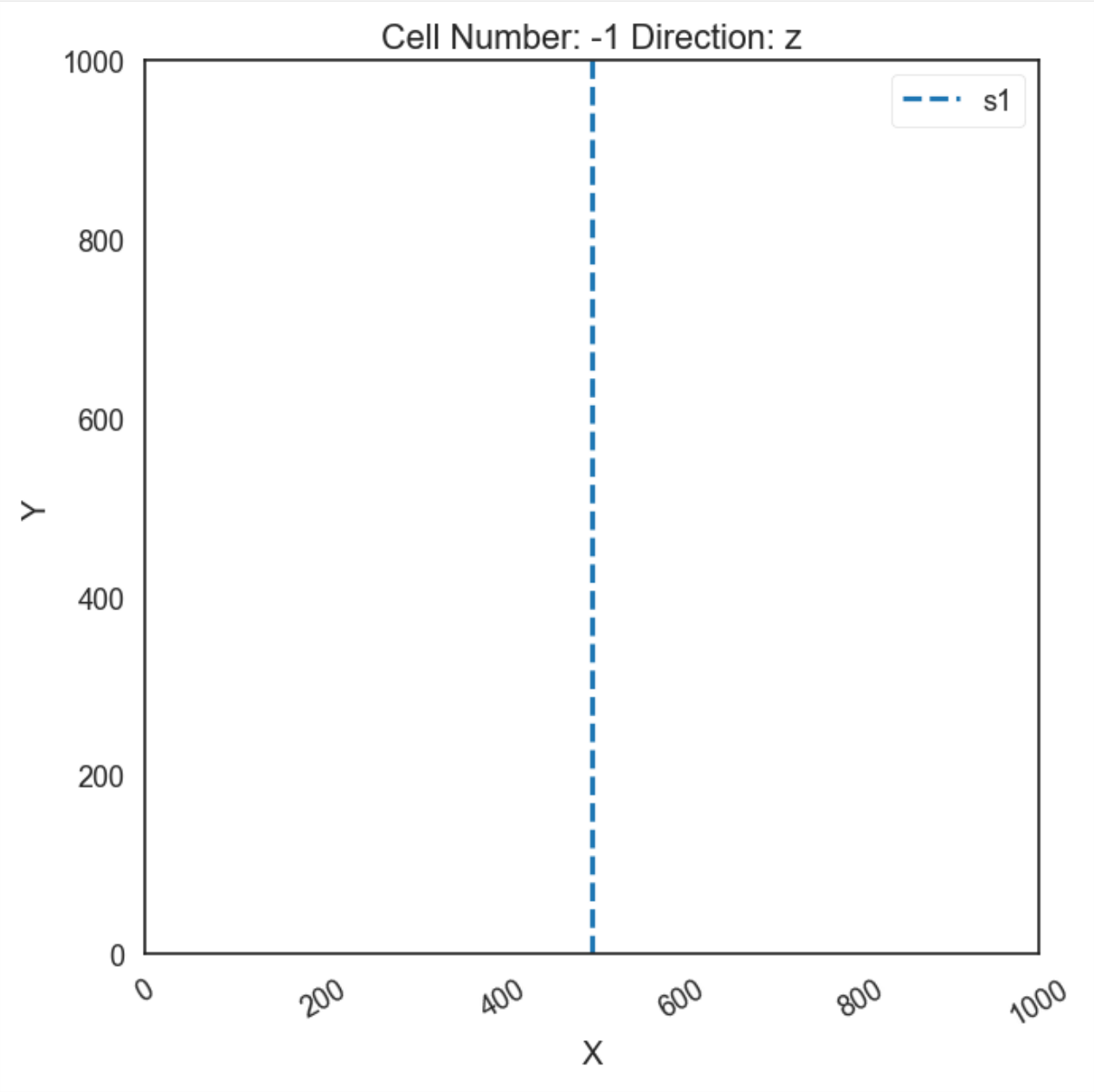
Creat custom section for Spaghetti Plot

A custom section needs to be created to later obtain the Spaghetti Plot

[11]:
```
geo_model.set_section_grid({'s1':([500,0],[500,1000],[80,80])})
gp.plot.plot_section_traces(geo_model)
```

```
Active grids: ['regular' 'sections']
```

[11]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1fd1f5dee60>
```



Set Interpolator

[12]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
```

(continues on next page)

(continued from previous page)

```
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range             1500.00
$C_o$            53571.43
drift equations       [3]
```

[12]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1fd293c2b60>
```

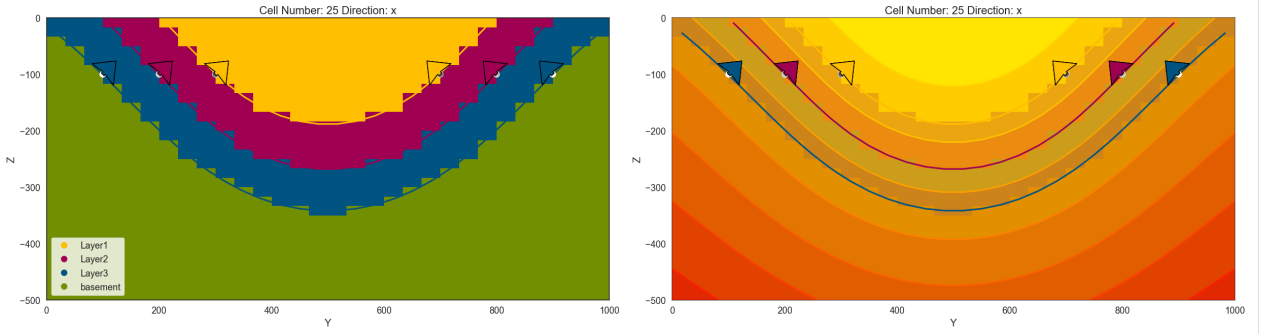
Compute Model

[13]:
```
sol = gp.compute_model(geo_model)
```

Plot Model

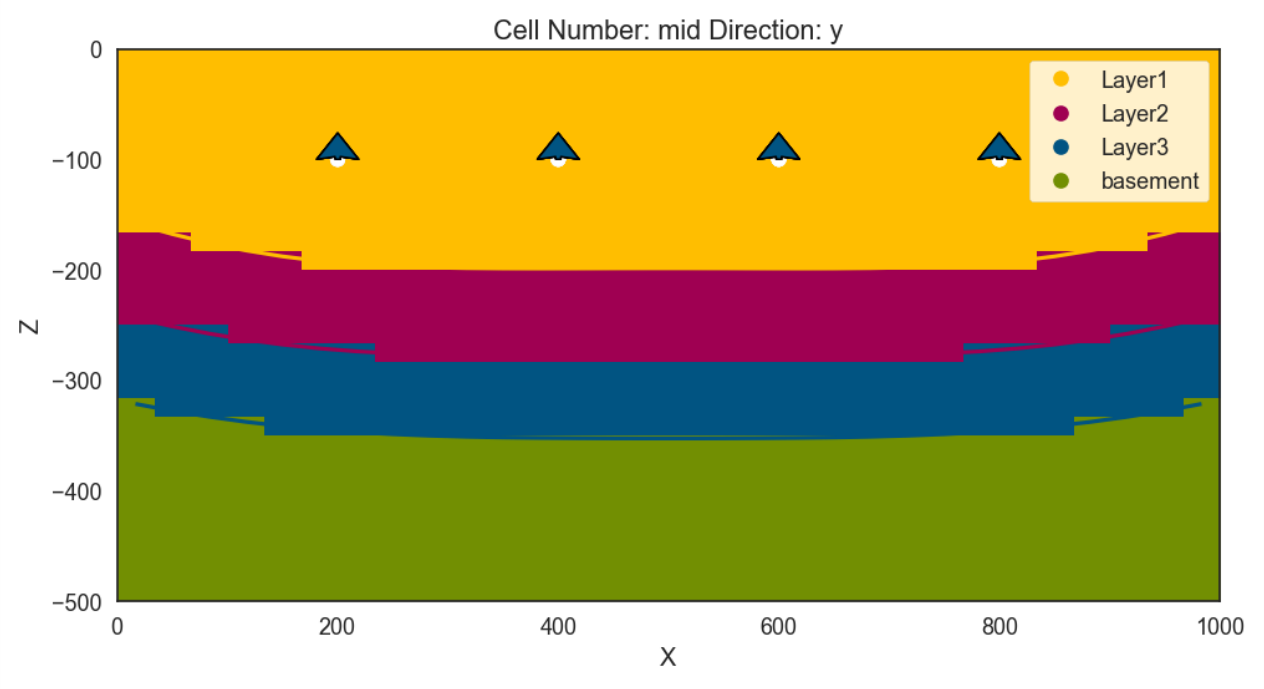
[14]:
```
gp.plot_2d(geo_model, direction=['x', 'x'], cell_number=[25, 25,], show_data=True, show_
↪scalar = [False, True])
```

[14]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1fd2af01c90>
```

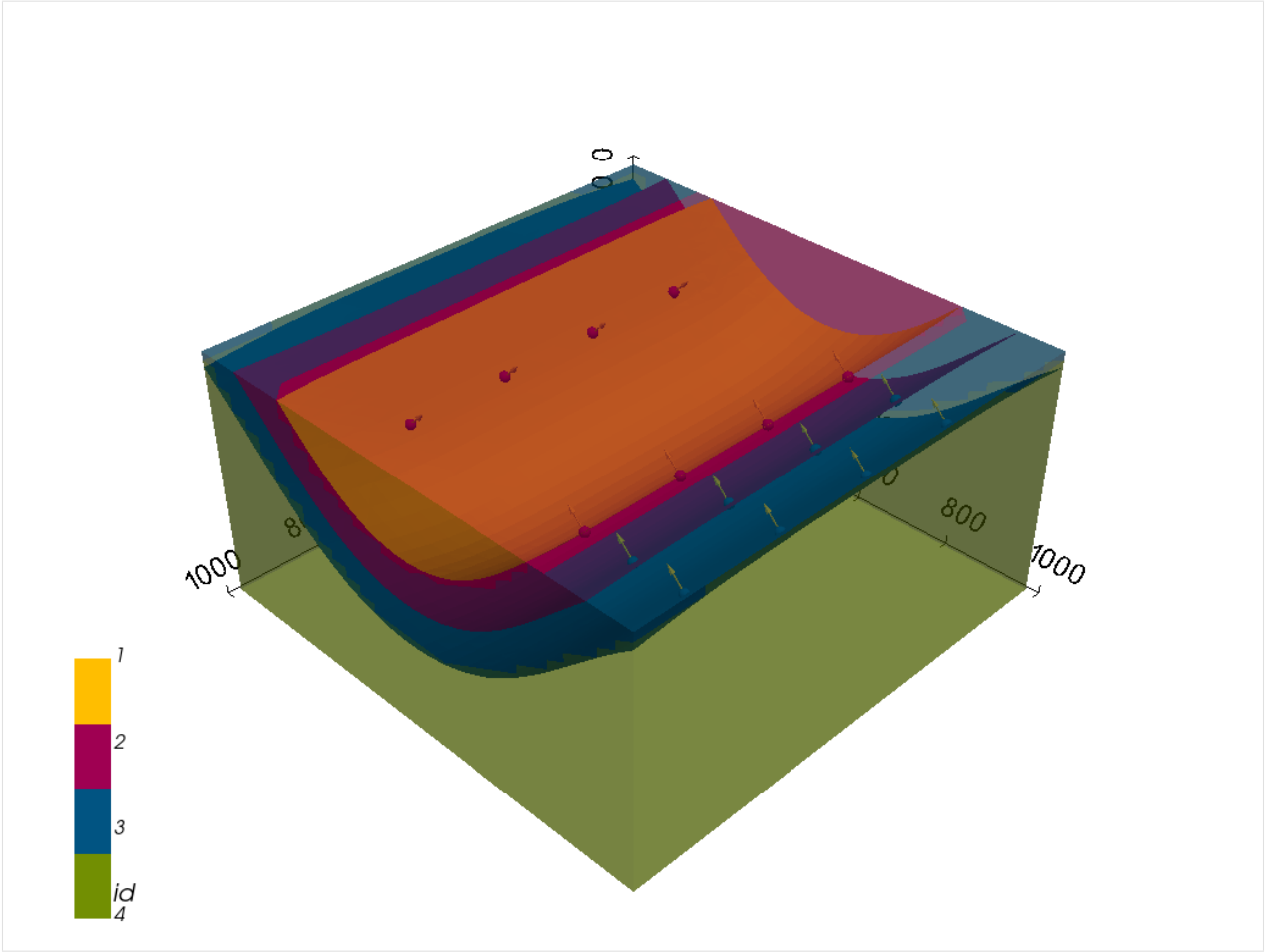


[15]:
```
gp.plot_2d(geo_model, section=['s1'])
```

[15]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1fd2c5f11e0>
```



[16]:
```
gpv = gp.plot_3d(geo_model,
                 image=False,
                 show_topography=True,
                 plotter_type='basic',
                 notebook=True)
```



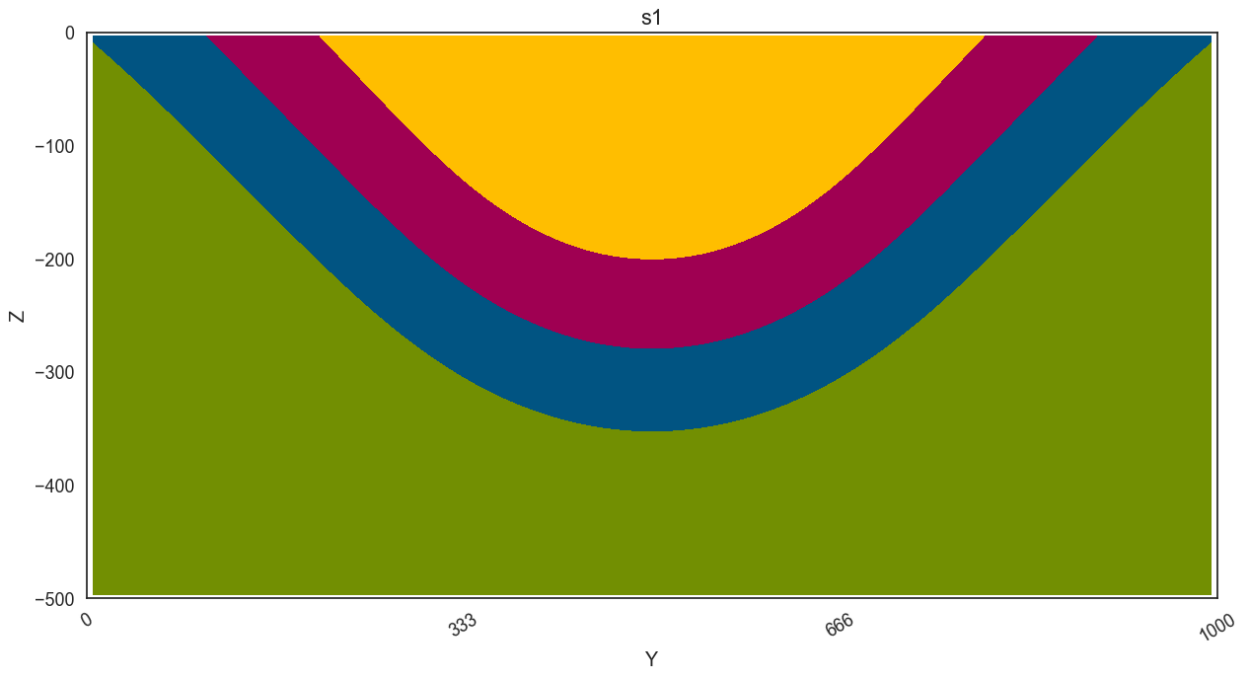
6.58.6 Probabilistic Modeling

During the probabilistic modeling, multiple model realizations will be created by varying the input parameter and recomputing the models with the new input data. The Monte Carlo approach is the base for the Spaghetti plot.

[17]:
```
from gempy.bayesian.fields import compute_prob, calculate_ie_masked
from gempy.core.grid_modules import section_utils
from tqdm.notebook import tqdm
import numpy as np
import copy
```

Getting polygondict for the spaghetti plot

[18]:
```
polygondict_s1, cdict, extent = section_utils.get_polygon_dictionary(geo_model, 's1');
```



Copying the original model data

A deep copy is created for all interface points and locations as well as their indices.

[19]:
```
df_int_X = copy.copy(geo_model.surface_points.df['X'])
df_int_Y = copy.copy(geo_model.surface_points.df['Y'])
df_int_Z = copy.copy(geo_model.surface_points.df['Z'])

df_or_X = copy.copy(geo_model.orientations.df['X'])
df_or_Y = copy.copy(geo_model.orientations.df['Y'])
df_or_Z = copy.copy(geo_model.orientations.df['Z'])
df_or_dip     = copy.copy(geo_model.orientations.df['dip'])
df_or_azimuth = copy.copy(geo_model.orientations.df['azimuth'])

surfindexes = list(geo_model.surface_points.df.index)
orindexes = list(geo_model.orientations.df.index)
```

Monte Carlo Simulation

[20]:
```
# Preventing matplotlib from plotting
%matplotlib agg

# Defining number of iterations
n_iterations = 50

# Getting indices
indices = list(geo_model.surface_points.df.index.sort_values()[:24])
indices_or = list(geo_model.orientations.df.index)

# Defining empty lith_block array
lith_blocks = np.array([])

# Defining empty list to store meshes
meshes = []

for iterations in tqdm(range(n_iterations)):

    # Setting seed for reproducability
    np.random.seed(iterations)

    # Resetting Dataframes
    geo_model.modify_surface_points(surfindexes, X=df_int_X, Y=df_int_Y, Z=df_int_Z)
    geo_model.modify_orientations(orindexes, X=df_or_X, Y=df_or_Y, Z=df_or_Z,dip = df_or_
↪dip, azimuth = df_or_azimuth)
    geo_model.update_to_interpolator()

    # Modifying Points
    dip_sample = np.random.uniform(25,65, size=1)
    for i in range(len(indices_or)):
        geo_model.modify_orientations(indices_or[i], dip = dip_sample)

   # Setting Interpolator and compute model
    geo_model.update_to_interpolator()
    sol = gp.compute_model(geo_model, compute_mesh=True)


    #Spaghetti Plots
    polygondict, cdict, extent = section_utils.get_polygon_dictionary(geo_model, 's1')
    for form in polygondict.keys():
        polygondict_s1.get(form).append(polygondict.get(form))

    # Saving lith_block after each model run
    lith_blocks = np.append(lith_blocks, geo_model.solutions.lith_block)
#     np.save('lith_blocks_syncline_dip_all_new.npy', lith_blocks)

    # Exporting meshes using GemGIS
    mesh = gg.visualization.create_depth_maps_from_gempy(geo_model, surfaces=['Layer2'])
    meshes.append(mesh)
```

```
  0%|          | 0/50 [00:00<?, ?it/s]
```

Reshaping lith_blocks array

[21]:
```
# lith_blocks_syncline_dip_all = np.load('lith_blocks_syncline_dip_all.npy').reshape(n_
↪iterations, -1)
prob_block_syncline_dip_all = compute_prob(lith_blocks.reshape(n_iterations, -1))
print(prob_block_syncline_dip_all.shape)
prob_block_syncline_dip_all
```

```
(4, 27000)
```

[21]:
```
array([[0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.  ],
       [0.  , 0.  , 0.  , ..., 0.  , 0.  , 0.12],
       [0.  , 0.  , 0.  , ..., 0.5 , 0.64, 0.64],
       [1.  , 1.  , 1.  , ..., 0.5 , 0.36, 0.24]])
```

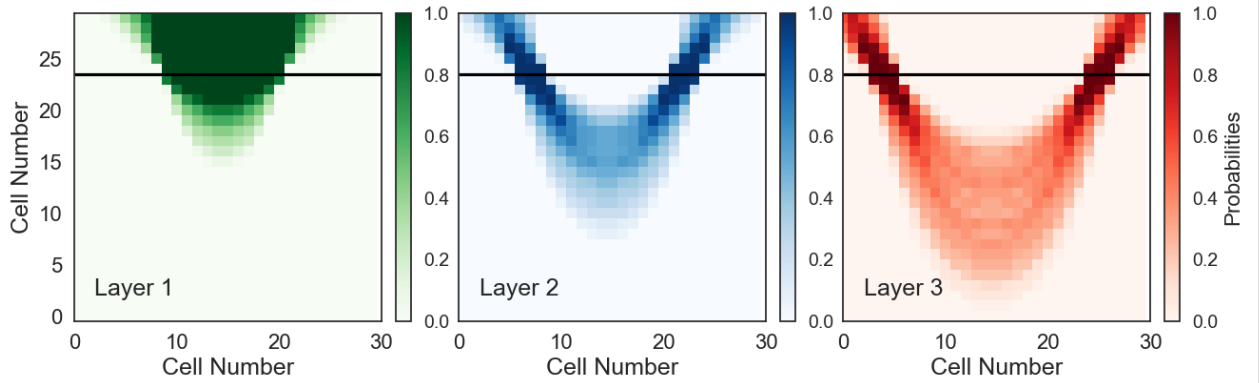
Plotting Probabilities

The probabilities of each layer being present in a cell are plotted.

[22]:
```
from mpl_toolkits.axes_grid1 import ImageGrid
import matplotlib.pyplot as plt
```

[23]:
```
def plot_probs(array):
    fig = plt.figure(figsize=(15, 5))

    grid = ImageGrid(fig, 111,
                     nrows_ncols=(1,3),
                     axes_pad=0.5,
                     share_all=True,
                     cbar_location="right",
                     cbar_mode="each",
                     cbar_size="5%",
                     cbar_pad=0.15,
                     )
    cmaps = ['Greens', 'Blues', 'Reds']
    for counter, ax in enumerate(grid):
        im = ax.imshow(np.fliplr(array[counter].reshape(30,30,30)[
                       25].T), cmap=cmaps[counter], origin='lower', vmin=0, vmax=1)
        ax.set_xlabel('Cell Number',fontsize=18 )
        ax.set_ylabel('Cell Number',fontsize=18)
        ax.hlines(y=23.5,xmin=0,xmax=200, color='k')
        ax.set_xlim(0,30)
        ax.text(x=2, y=2, s='Layer ' + str(counter+1))


        cbar = ax.cax.colorbar(im)
        ax.tick_params(axis='both', which='major', labelsize=16)
        ax.tick_params(axis='both', which='minor', labelsize=16)
        cbar.ax.tick_params(axis='both', which='major', labelsize=14)
        cbar.ax.tick_params(axis='both', which='minor', labelsize=14)
        ax.cax.toggle_label(True)
    cbar.ax.set_ylabel('Probabilities', fontsize=16)
```

[24]:
```
%matplotlib inline
plot_probs(prob_block_syncline_dip_all)
```



6.58.7 Plotting Entropy and Spaghetti Plot

[25]:
```
import matplotlib.patches as patches
from matplotlib.lines import Line2D
from mpl_toolkits.axes_grid1 import make_axes_locatable
from matplotlib.path import Path
```

[26]:
```
%matplotlib agg
fig, ax = plt.subplots()
def plot_pathdict(pathdict, cdict, extent, ax=None, surfaces=None, color=None):

    for formation in surfaces:
        print(formation)
        for path in pathdict.get(formation):
            if path !=[]:
                if type(path) == Path:
                    patch = patches.PathPatch(path, fill=False, lw=0.25, edgecolor=color)
                    ax.add_patch(patch)
                elif type(path) == list:
                    for subpath in path:
                        assert type(subpath == Path)
                        patch = patches.PathPatch(subpath, fill=False, lw=0.25,␣
↪edgecolor=color)
                        ax.add_patch(patch)
    ax.set_ylim(-500,0)
    ax.set_xlim(extent[:2])
```

[27]:
```
entropy_block_syncline_dip_all = calculate_ie_masked(prob_block_syncline_dip_all)
```

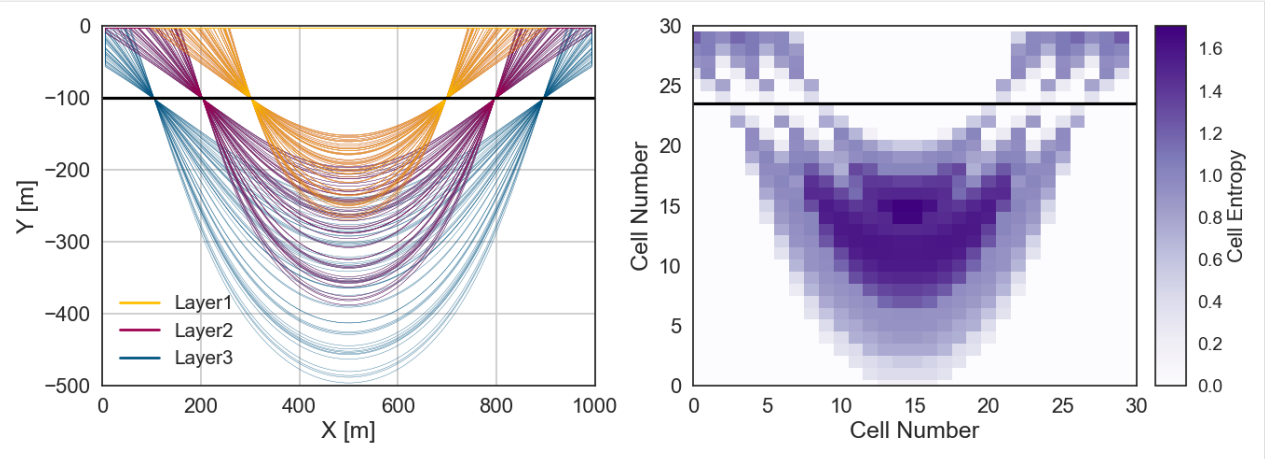
[28]:
```
%matplotlib inline
fix, ax = plt.subplots(nrows=1, ncols=2,sharex=False, sharey=False, squeeze=True ,␣
↪figsize = (15,5))
```

(continues on next page)

(continued from previous page)

```
custom_lines = [Line2D([0], [0], color='#ffbe00', lw=2),
                Line2D([0], [0], color='#9f0052', lw=2),
                Line2D([0], [0], color='#015482', lw=2)]


plot_pathdict(polygondict_s1, cdict, extent, ax=ax[0], surfaces=['Layer3'], color=cdict[
↪'Layer3'])
plot_pathdict(polygondict_s1, cdict, extent, ax=ax[0], surfaces=['Layer2'], color=cdict[
↪'Layer2'])
plot_pathdict(polygondict_s1, cdict, extent, ax=ax[0], surfaces=['Layer1'], color=cdict[
↪'Layer1'])

ax[0].set_aspect('auto')
ax[0].hlines(y=-100,xmin=0,xmax=1000, color='k')
ax[1].set_xlim(0,1000)
ax[0].tick_params(axis='both', which='major', labelsize=16)
ax[0].tick_params(axis='both', which='minor', labelsize=16)
ax[0].set_xlabel('X [m]',fontsize=18)
ax[0].set_ylabel('Y [m]',fontsize=18)
ax[0].grid()
ax[0].legend(custom_lines, ['Layer1', 'Layer2', 'Layer3'], fontsize=15)


im = ax[1].imshow(np.fliplr(entropy_block_syncline_dip_all.reshape(30,30,30)[
                15].T), cmap='Purples', origin='lower', vmax = entropy_block_syncline_dip_
↪all.max(),aspect="auto")
ax[1].set_xlabel('Cell Number',fontsize=18)
ax[1].set_ylabel('Cell Number',fontsize=18)
ax[1].hlines(y=23.5,xmin=0,xmax=200, color='k')
ax[1].set_xlim(0,30)
ax[1].set_ylim(0,30)

ax[1].tick_params(axis='both', which='major', labelsize=16)
ax[1].tick_params(axis='both', which='minor', labelsize=16)

divider = make_axes_locatable(ax[1])
cax = divider.append_axes('right', size="7%", pad=0.2,)
cbar = fig.colorbar(im, cax=cax)

cbar.ax.tick_params(axis='both', which='major', labelsize=14)
cbar.ax.tick_params(axis='both', which='minor', labelsize=14)
cbar.ax.set_ylabel('Cell Entropy', fontsize=16)


fig.tight_layout()

plt.savefig('fig4.pdf', bbox_inches='tight', pad_inches=0.2)
```

```
Layer3
Layer2
Layer1
```

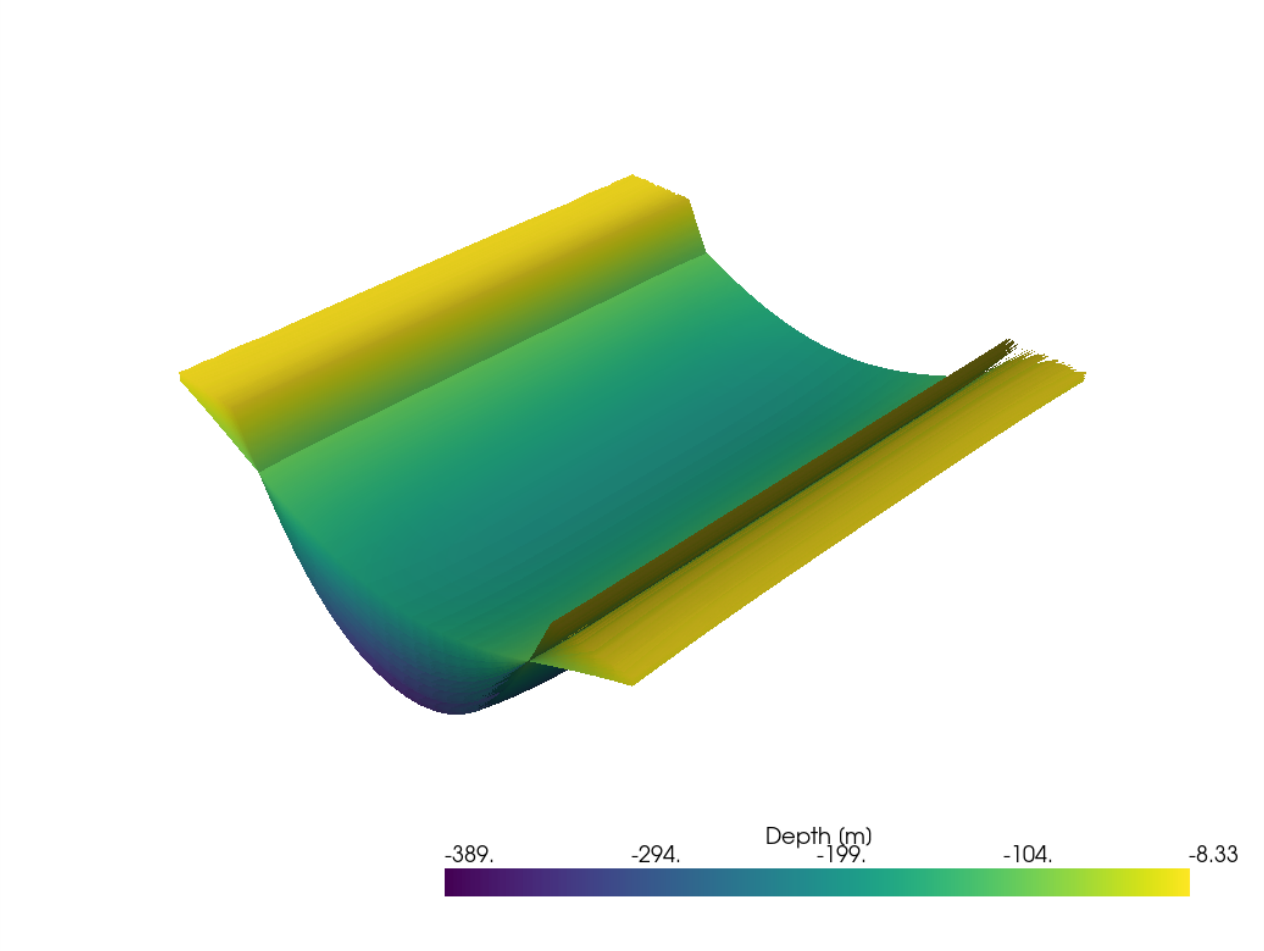


Plotting Meshes in 3D

[29]:
```
meshes_layer2 = [mesh['Layer2'][0] for mesh in meshes]
meshes_layer2[0]
```

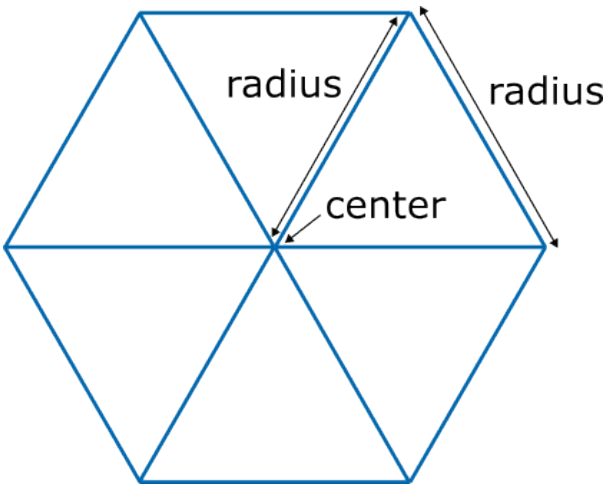
[29]:
```
PolyData (0x1fd2d3a42e0)
  N Cells:  3362
  N Points: 1764
  N Strips: 0
  X Bounds: 1.667e+01, 9.833e+02
  Y Bounds: 1.103e+02, 8.897e+02
  Z Bounds: -2.882e+02, -8.333e+00
  N Arrays: 1
```

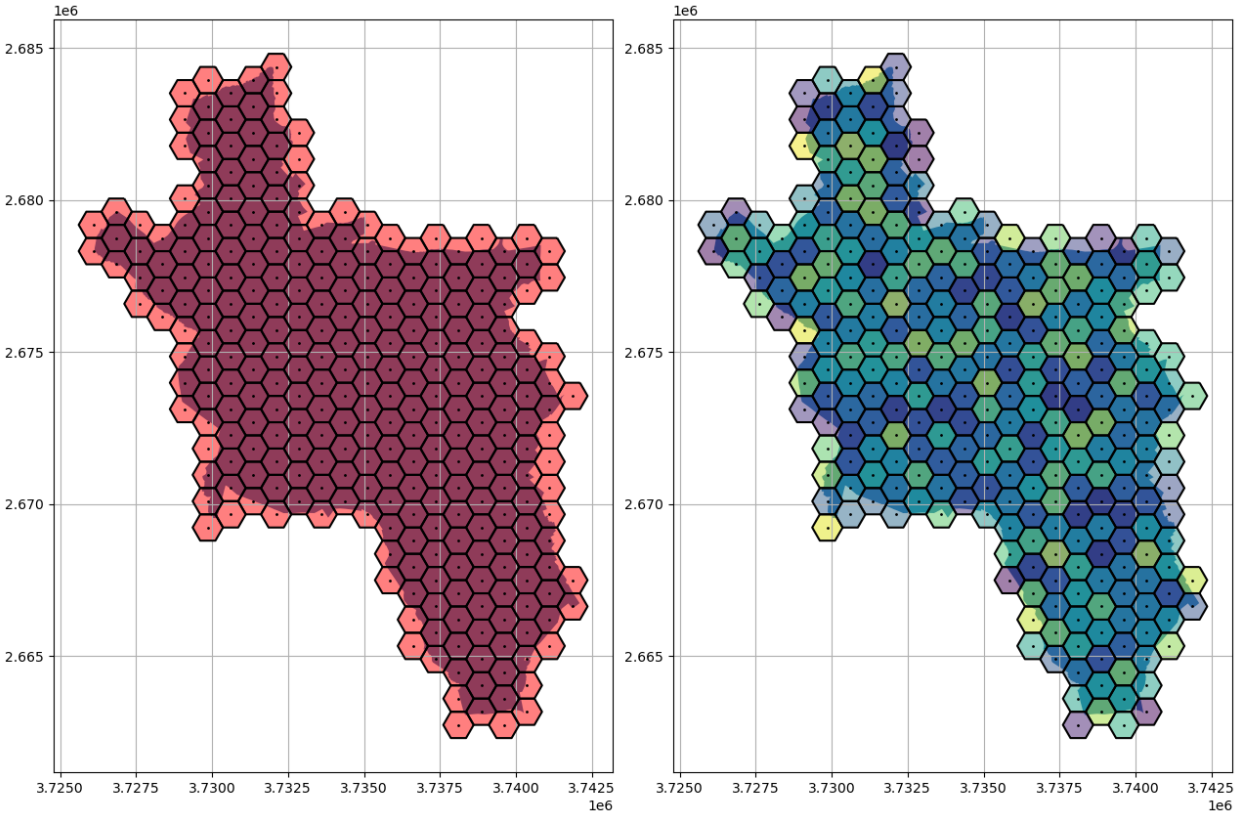
[30]:
```
import pyvista as pv
p = pv.Plotter(notebook=True)

for mesh in meshes_layer2:
    p.add_mesh(mesh)

p.show()
```



6.59 58 Creating hexagonal grid in GemGIS

This notebook illustrates how to create a hexagonal grid using GemGIS. This will be demonstrated for the boundaries of the city of Aachen provided as Shapely Polygon. A radius for the hexagons can be defined to adjust the number of total hexagons. This radius equals the distance from the center point to each vertex and is by definition the length of each side of the hexagon.





6.59.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")
```

(continues on next page)

(continued from previous page)

```
import geopandas as gpd
import pandas as pd
import gemgis as gg
import numpy as np
import matplotlib.pyplot as plt
from shapely.geometry import Point, Polygon
```

[2]:
```
file_path ='data/58_creating_hexagonal_grid/'
gg.download_gemgis_data.download_tutorial_data(filename="58_creating_hexagonal_grid.zip",
↪ dirpath=file_path)
```

6.59.2 Loading Outline of Aachen from Shapefile

The outline of the city of Aachen is loaded as GeoDataFrame and displayed as Shapely polyon.

[3]:
```
outline = gpd.read_file(file_path + 'outline.shp')
outline.loc[0].geometry
```

[3]:

6.59.3 Creating a single hexagon

A single hexagon can be created by providing the coordinates of the center point and the radius of hexagon using the GemGIS method `gg.vector.create_hexagon(...)`. The radius is equal to the radius of the inner circle within a hexagon.

[4]:
```
gg.vector.create_hexagon(center=Point(0,0),
                         radius=1)
```

[4]:

6.59.4 Obtaining the total bounds polygon for the calculation of hexagons

Based on the input shape, a total bounds polygon is calculated to create the hexagon grid.

[5]:
```
outline.total_bounds
```

[5]:
```
array([3726110.8907332 , 2663054.98848197, 3742096.48479688,
       2684380.29935271])
```

[6]:
```
outline_total_bounds_polygon = Polygon([(outline.total_bounds[0],
                                         outline.total_bounds[1]),
                                        (outline.total_bounds[0],
                                         outline.total_bounds[3]),
                                        (outline.total_bounds[2],
                                         outline.total_bounds[3]),
                                        (outline.total_bounds[2],
                                         outline.total_bounds[1])])

outline_total_bounds_polygon
```

[6]:

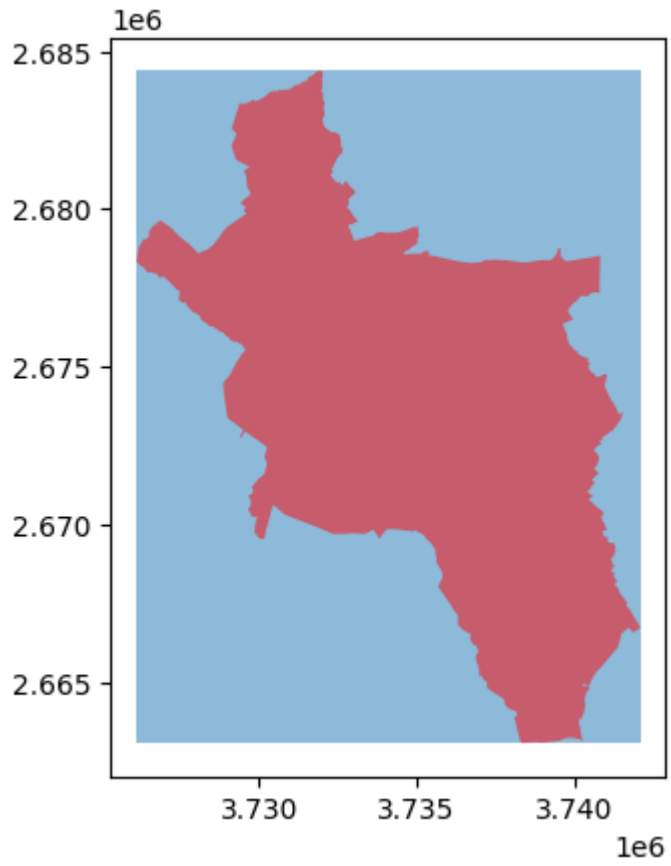
[7]:
```
polygon_gdf = gpd.GeoDataFrame(geometry=[outline_total_bounds_polygon],
                               crs=outline.crs)
polygon_gdf
```

[7]:
```
                                            geometry
0  POLYGON ((3726110.891 2663054.988, 3726110.891...
```

6.59.5 Plotting the outline and the total bounds polygon

[8]:
```
fix, ax = plt.subplots(1)

polygon_gdf.plot(ax=ax, alpha=0.5)
outline.plot(color='red', ax=ax, alpha=0.5)
```

[8]:
```
<Axes: >
```



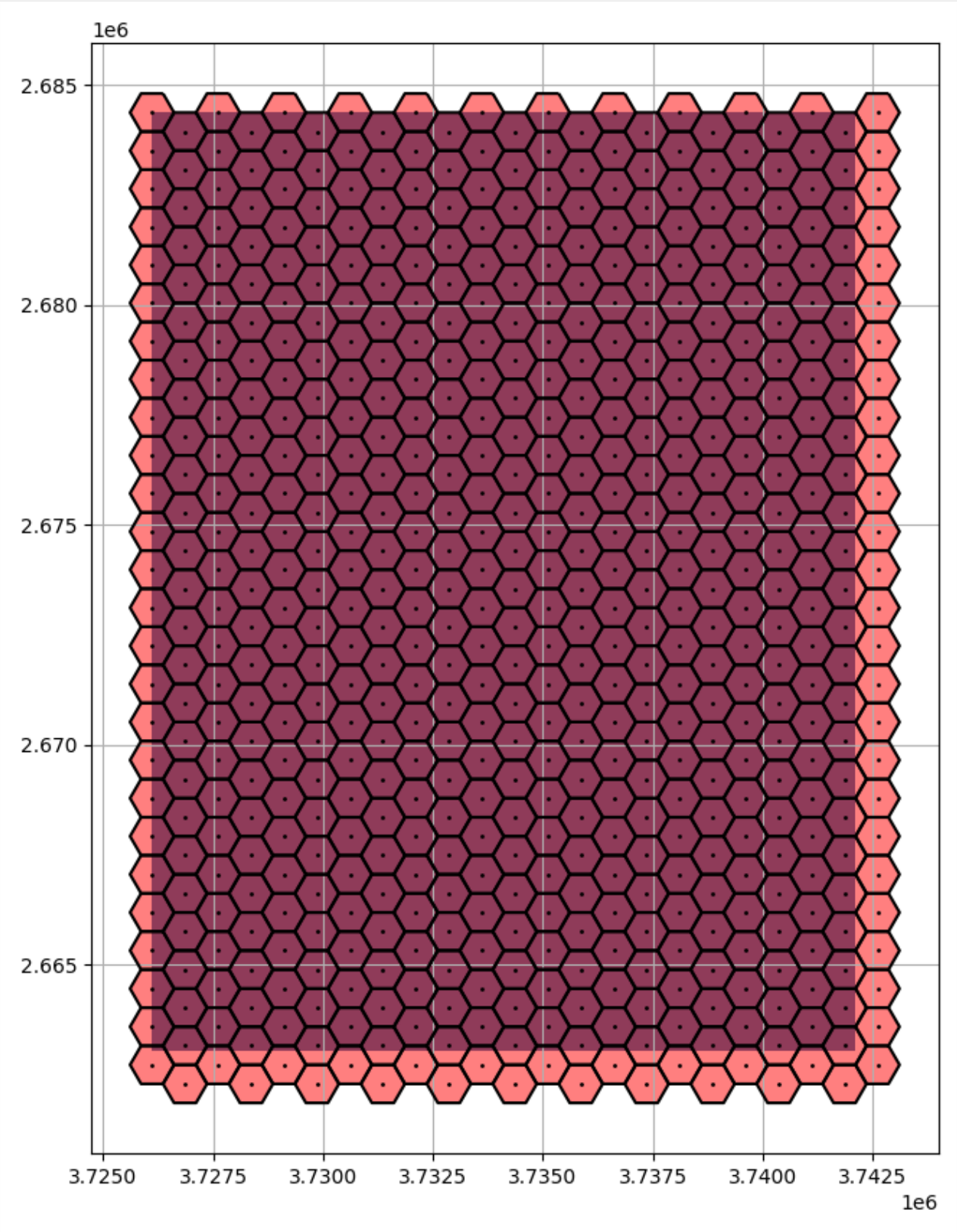
6.59.6 Creating hexagon grid

The function `gg.vector.create_hexagon_grid(...)` creates a hexagon grid based on the total bounds of the provided outline. Setting `crop_gdf` to `False` will return the entire GeoDataFrame with the hexagon grid.

[9]:
```
hex_gdf = gg.vector.create_hexagon_grid(gdf=outline,
                                        radius=500,
                                        crop_gdf=False)
hex_gdf.head()
```

[9]:
```
                                            geometry
0  POLYGON ((3726360.891 2684813.312, 3725860.891...
1  POLYGON ((3726360.891 2683947.287, 3725860.891...
2  POLYGON ((3726360.891 2683081.261, 3725860.891...
3  POLYGON ((3726360.891 2682215.236, 3725860.891...
4  POLYGON ((3726360.891 2681349.210, 3725860.891...
```

[10]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

polygon_gdf.plot(ax=ax)

hex_gdf.plot(ax=ax,
             color='red',
             alpha=0.5)

hex_gdf.exterior.plot(ax=ax,
                      color='black')

hex_gdf.centroid.plot(ax=ax,
                      color='black',
                      markersize=1)
plt.grid()
```

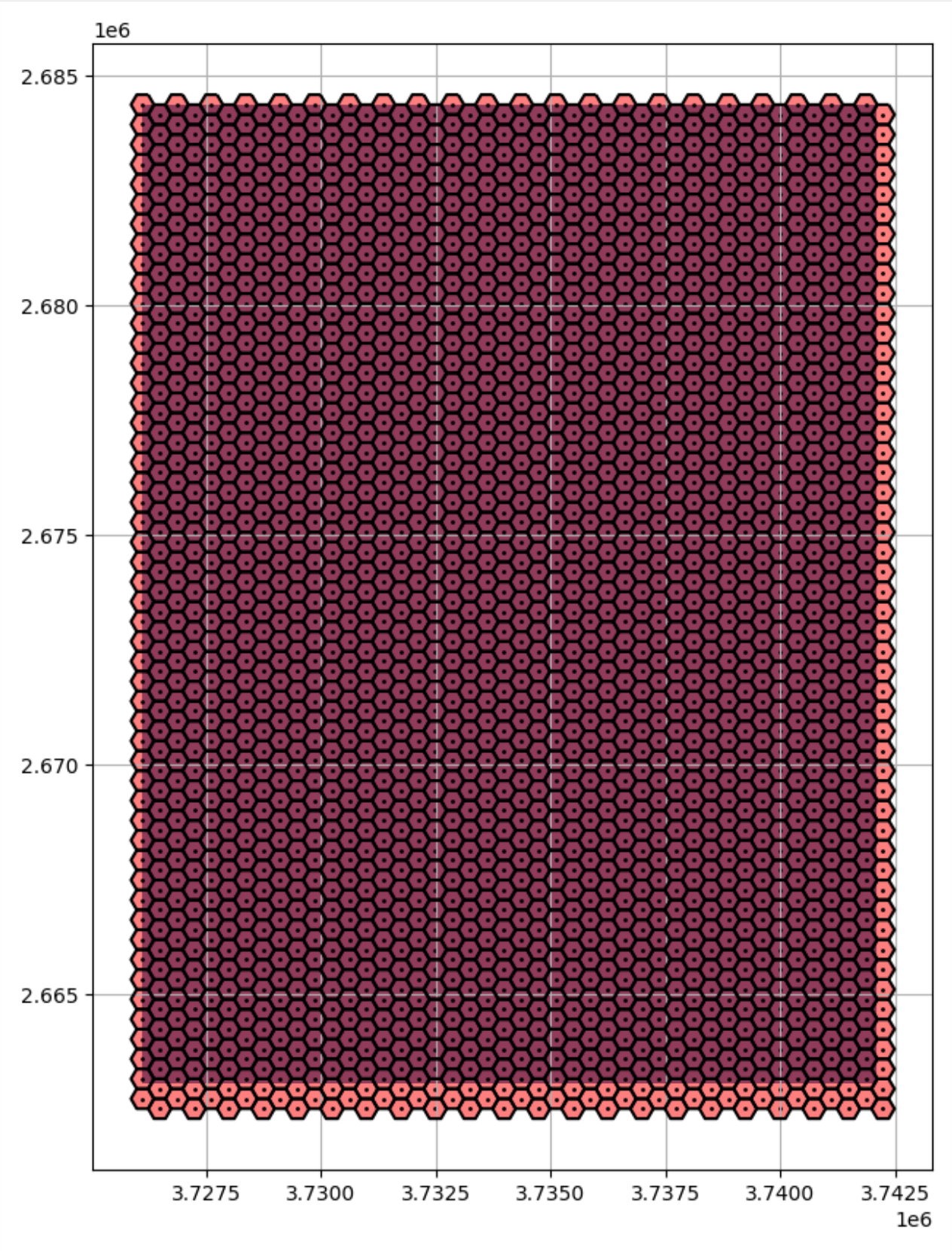


Using a smaller radius will lead to a finer grid with a higher number of hexagons.

[11]:
```
hex_gdf250 = gg.vector.create_hexagon_grid(gdf=outline,
                                           radius=250,
                                           crop_gdf=False)
hex_gdf250.head()
```

[11]:
```
                                            geometry
0  POLYGON ((3726235.891 2684596.806, 3725985.891...
1  POLYGON ((3726235.891 2684163.793, 3725985.891...
2  POLYGON ((3726235.891 2683730.780, 3725985.891...
3  POLYGON ((3726235.891 2683297.768, 3725985.891...
4  POLYGON ((3726235.891 2682864.755, 3725985.891...
```

[12]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

polygon_gdf.plot(ax=ax)

hex_gdf250.plot(ax=ax,
                color='red',
                alpha=0.5)

hex_gdf250.exterior.plot(ax=ax,
                         color='black')

hex_gdf250.centroid.plot(ax=ax,
                         color='black',
                         markersize=1)
plt.grid()
```


1e6
2.685
2.680
2.675
2.670
2.665
3.7275
3.7300
3.7325
3.7350
3.7375
3.7400
3.7425
1e6

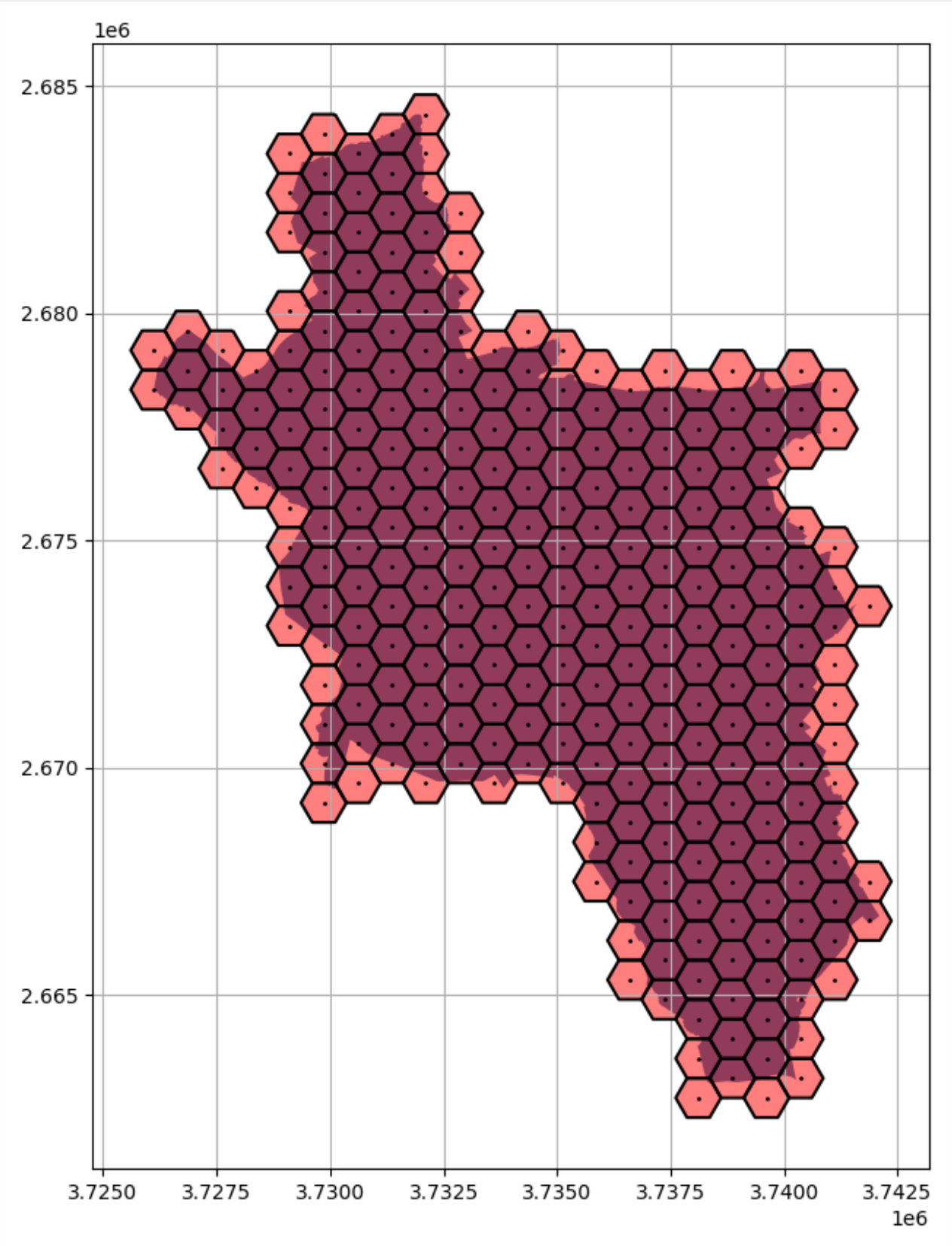
6.59.7 Creating cropped hexagon grid

Setting `crop_gdf` to `True` will automatically crop the resulting GeoDataFrame to the outline of the provided GeoDataFrame.

[13]:
```
hex_gdf = gg.vector.create_hexagon_grid(gdf=outline,
                                        radius=500,
                                        crop_gdf=True)
hex_gdf.head()
```

[13]:
```
                                            geometry
0  POLYGON ((3726360.891 2679617.160, 3725860.891...
1  POLYGON ((3726360.891 2678751.134, 3725860.891...
2  POLYGON ((3727110.891 2680050.172, 3726610.891...
3  POLYGON ((3727110.891 2679184.147, 3726610.891...
4  POLYGON ((3727110.891 2678318.122, 3726610.891...
```

[14]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

outline.plot(ax=ax)

hex_gdf.plot(ax=ax,
             color='red',
             alpha=0.5)

hex_gdf.exterior.plot(ax=ax,
                      color='black')

hex_gdf.centroid.plot(ax=ax,
                      color='black',
                      markersize=1)

plt.grid()
```

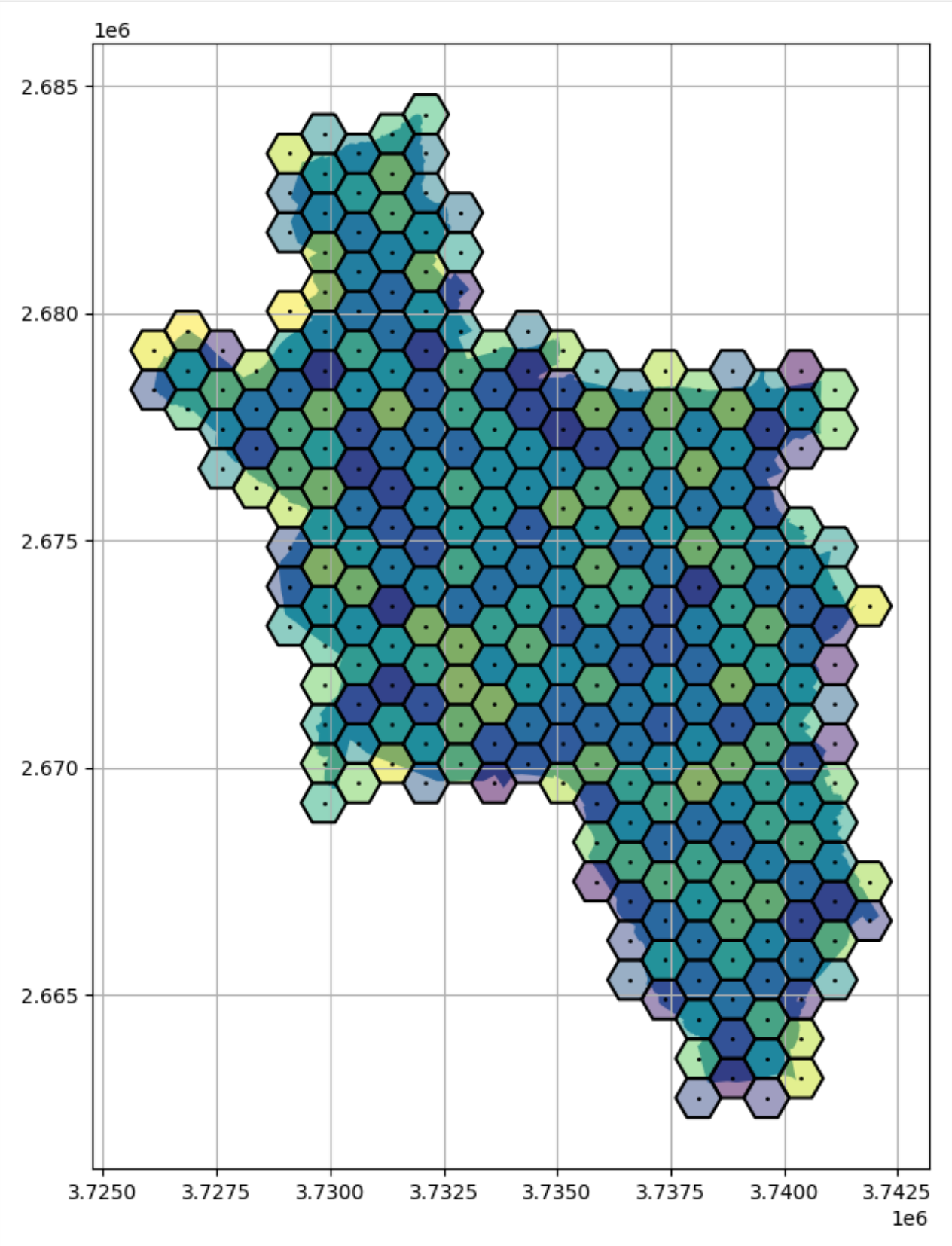

1e6
2.685
2.680
2.675
2.670
2.665
3.7250
3.7275
3.7300
3.7325
3.7350
3.7375
3.7400
3.7425
1e6

6.59.8 Displaying random values in hexagon

The main purpose of creating the hexagon grid is to display certain values of the underlying map (e.g. population density, heat demand, etc.). The following simply illustrates how to assign random values to the hexagons and how to plot them.

[15]:
```
values = np.random.randint(0,
                           200,
                           len(hex_gdf))

hex_gdf['values'] = values
```

[16]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

outline.plot(ax=ax)

hex_gdf.plot(ax=ax,
             column='values',
             alpha=0.5)

hex_gdf.exterior.plot(ax=ax,
                      color='black')

hex_gdf.centroid.plot(ax=ax,
                      color='black',
                      markersize=1)

plt.grid()
```


1e6
2.685
2.680
2.675
2.670
2.665
3.7250
3.7275
3.7300
3.7325
3.7350
3.7375
3.7400
3.7425
1e6

6.60 59 Visualizing DoubletCalc Results

This notebook illustrates how to create the output plots provided by DoubletCalc with matplotlib.

6.60.1 Importing Libraries

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

6.61 Reading DoubletCalc Results CSV File

[2]:
```
data = pd.read_csv('data/export_output.csv',
                   skiprows = 2,
                   delimiter = ',',
                   encoding = "ISO-8859-1",
                   skipfooter = 1)
data
```

```
C:\Users\ale93371\AppData\Local\Temp\ipykernel_15364\1359617900.py:1: ParserWarning:␣
↪Falling back to the 'python' engine because the 'c' engine does not support skipfooter;
↪ you can avoid this warning by specifying engine='python'.
  data = pd.read_csv('data/export_output.csv',
```

[2]:
```
       iS   aquifer permeability (mD)   aquifer gross thickness (m)
0       1                   254.29453                      98.67665  \
1       2                   232.21005                     101.61317
2       3                   347.02588                     110.62248
3       4                   416.37143                     110.71101
4       5                   385.43112                     106.67485
..    ...                         ...                           ...
995   996                   208.79338                     107.78278
996   997                   335.04092                     107.57787
997   998                   230.71222                     100.96610
998   999                   246.51558                     102.81991
999  1000                   294.76205                     105.02041

      aquifer net to gross (-)   aquifer top at injector (m TVD)
0                     0.809259                         2458.3457  \
1                     0.787313                         2643.6982
2                     0.783280                         2322.7173
3                     0.793836                         2409.1120
4                     0.803228                         2553.3394
..                         ...                               ...
995                   0.796793                         2371.8330
996                   0.803419                         2431.6885
997                   0.813266                         2666.8223
998                   0.757966                         2573.6740
999                   0.792474                         2348.7234
```

(continues on next page)

(continued from previous page)

```
      aquifer top at producer (m TVD)   aquifer water salinity (ppm)
0                           2457.9678                       0.119523  \
1                           2521.1965                       0.123078
2                           2336.0005                       0.102066
3                           2465.4832                       0.126697
4                           2534.5093                       0.123331
..                                ...                            ...
995                         2634.3254                       0.116407
996                         2622.8328                       0.123286
997                         2487.1226                       0.108363
998                         2542.2840                       0.130512
999                         2447.9807                       0.122900

      aquifer kH net (Dm)   mass flow (kg/s)   delta T (°C)
0               20.306683          40.136295      49.975120  \
1               18.577118          37.513557      51.644176
2               30.069246          51.665833      47.223145
3               36.593388          57.302696      51.169163
4               33.025370          54.664562      53.064050
..                    ...                ...            ...
995             17.931303          38.144660      55.109097
996             28.957619          51.833920      55.581284
997             18.944315          38.278194      50.756508
998             19.211943          38.543340      52.329730
999             24.531853          45.703700      50.095210

      heat capacity (kJ/(kg*K))   av. production density (kg/m³)
0                        3664.4312                        1057.3365  \
1                        3651.1710                        1058.8849
2                        3733.5127                        1046.2684
3                        3636.7332                        1062.0903
4                        3650.8708                        1058.3341
..                             ...                              ...
995                      3679.4836                        1051.8035
996                      3652.3357                        1056.8402
997                      3709.6392                        1048.6069
998                      3622.4026                        1063.8132
999                      3651.1333                        1059.6786

      required pump power (kW)   pump volume flow (m³/h)
0                     248.91680                 136.65536  \
1                     232.31094                 127.53875
2                     323.81015                 177.77185
3                     353.78850                 194.22997
4                     338.69850                 185.94554
..                          ...                       ...
995                   237.80956                 130.55750
996                   321.61395                 176.56612
997                   239.36952                 131.41393
998                   237.58229                 130.43272
999                   282.81820                 155.26726
```

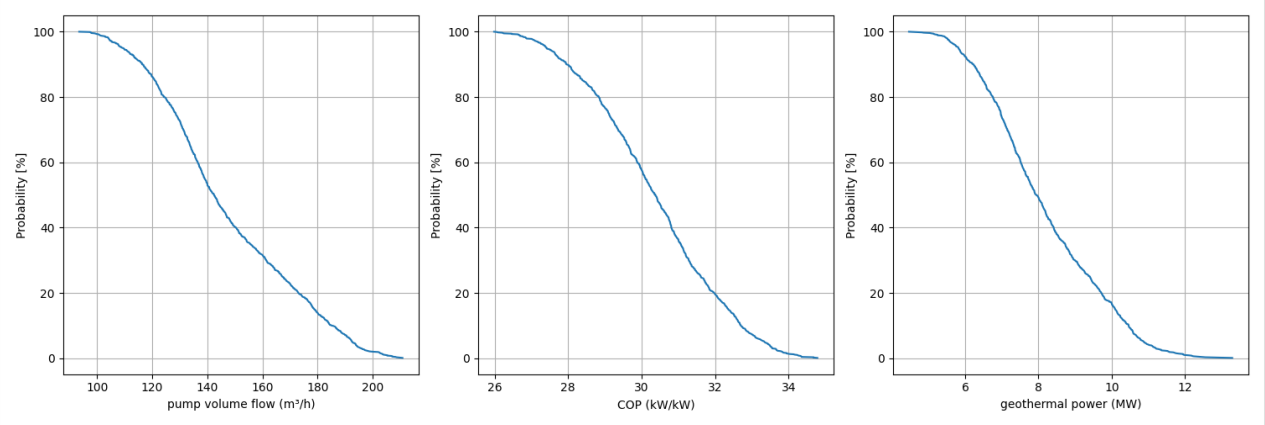
(continues on next page)

(continued from previous page)

```
      geothermal power (MW)   COP (kW/kW)
0                  7.350175     29.528645
1                  7.073621     30.448938
2                  9.109110     28.131021
3                 10.663378     30.140543
4                 10.590165     31.267237
..                      ...           ...
995                7.734708     32.524796
996               10.522364     32.717373
997                7.207337     30.109667
998                7.306250     30.752502
999                8.359403     29.557512

[1000 rows x 16 columns]
```

6.61.1 Plotting the DoubletCalc Results

[3]:
```
fig, axes = plt.subplots(1,3, figsize=(15,5))

axes[0].plot(data[' pump volume flow (m3/h)'].sort_values(ascending=False).values, np.
↪linspace(0.1,100,len(data)))
axes[1].plot(data[' COP (kW/kW)'].sort_values(ascending=False).values, np.linspace(0.1,
↪100,len(data)))
axes[2].plot(data[' geothermal power (MW)'].sort_values(ascending=False).values, np.
↪linspace(0.1,100,len(data)))

axes[0].grid()
axes[1].grid()
axes[2].grid()

#set label text
axes[0].set_ylabel("Probability [%]")
axes[0].set_xlabel("pump volume flow (m3/h)")

axes[1].set_ylabel("Probability [%]")
axes[1].set_xlabel("COP (kW/kW)")

axes[2].set_ylabel("Probability [%]")
axes[2].set_xlabel("geothermal power (MW)")

plt.tight_layout()
```



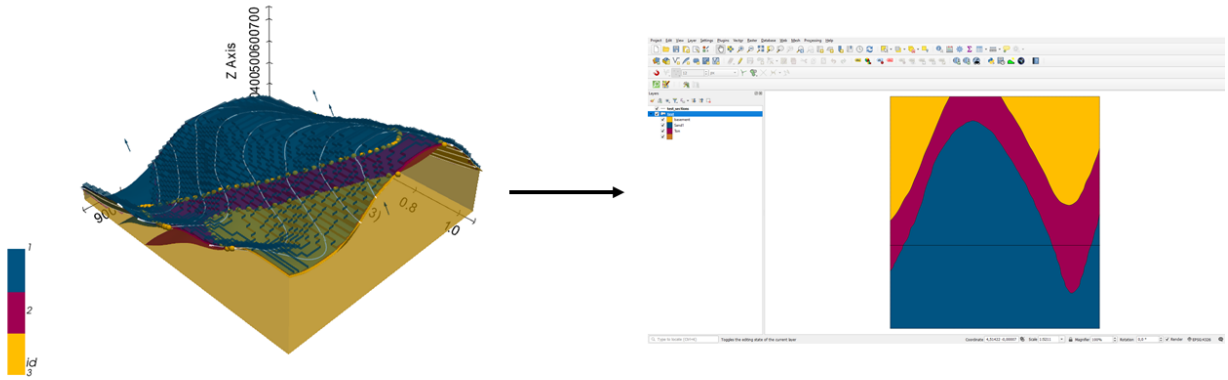
6.62 60 Adding labels to PyVista Contour Lines

[ ]:

6.63 61 Exporting Geological Maps and Custom sections from GemPy

6.63.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.



[22]:
```
import gemgis as gg

file_path ='data/61_Exporting_geological_map_and_custom_sections_from_GemPy/'
gg.download_gemgis_data.download_tutorial_data(filename="61_Exporting_geological_map_and_
↪custom_sections_from_GemPy.zip", dirpath=file_path)
```

6.63.2 Loading the data

[23]:
```
import geopandas as gpd
import rasterio

interfaces = gpd.read_file(file_path + 'interfaces.shp')
orientations = gpd.read_file(file_path + 'orientations.shp')
extent = [0,972,0,1069, 300, 800]
resolution = [50, 50, 50]
```

[24]:
```
interfaces.head()
```

[24]:
```
   level_0  level_1 formation     X      Y      Z                    geometry
0        0        0     Sand1  0.26 264.86 353.97   POINT (0.25633 264.86215)
1        0        0     Sand1 10.59 276.73 359.04  POINT (10.59347 276.73371)
2        0        0     Sand1 17.13 289.09 364.28  POINT (17.13494 289.08982)
3        0        0     Sand1 19.15 293.31 364.99  POINT (19.15013 293.31349)
4        0        0     Sand1 27.80 310.57 372.81  POINT (27.79512 310.57169)
```

[25]:
```
orientations['polarity'] = 1
orientations.head()
```

[25]:
```
  formation   dip  azimuth      X      Y      Z                    geometry  \
0       Ton 30.50   180.00  96.47 451.56 477.73   POINT (96.47104 451.56362)
1       Ton 30.50   180.00 172.76 661.88 481.73  POINT (172.76101 661.87650)
2       Ton 30.50   180.00 383.07 957.76 444.45  POINT (383.07389 957.75787)
3       Ton 30.50   180.00 592.36 722.70 480.57  POINT (592.35583 722.70229)
4       Ton 30.50   180.00 766.59 348.47 498.96  POINT (766.58562 348.46907)

   polarity
0         1
1         1
2         1
3         1
4         1
```

6.63.3 Creating the GemPy Model

[26]:
```
import sys
sys.path.append('../../../../gempy-master')
import gempy as gp
```

[27]:
```
geo_model = gp.create_model('Model1')
geo_model
```

[27]:
```
Model1  2023-05-08 12:52
```

Initiating the Model

[28]:
```
import pandas as pd

gp.init_data(geo_model, extent, resolution,
             surface_points_df = interfaces,
             orientations_df = orientations,
             default_values=True)
geo_model.surfaces
```

```
Active grids: ['regular']
```

[28]:

	surface	series	order_surfaces	color	id
0	Sand1	Default series	1	#015482	1
1	Ton	Default series	2	#9f0052	2

The vertices and edges are currently NaN values, so no model has been computed so far.

[29]:
```
geo_model.surfaces.df
```

[29]:

	surface	series	order_surfaces	isBasement	isFault	isActive	hasData	color	vertices	edges	sfai	id
0	Sand1	Default series	1	False	False	True	True	#015482	NaN	NaN	NaN	1
1	Ton	Default series	2	True	False	True	True	#9f0052	NaN	NaN	NaN	2

Mapping Stack to Surfaces

[30]:
```
gp.map_stack_to_surfaces(geo_model,
                         {"Strat_Series": ('Sand1', 'Ton')},
                         remove_unused_series=True)
geo_model.add_surfaces('basement')
```

[30]:

	surface	series	order_surfaces	color	id
0	Sand1	Strat_Series	1	#015482	1
1	Ton	Strat_Series	2	#9f0052	2
2	basement	Strat_Series	3	#ffbe00	3

Loading the Topography

[31]:
```
geo_model.set_topography(
    source='gdal', filepath='data/61_Exporting_geological_map_and_custom_sections_from_
↪GemPy/raster1.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[31]:
```
Grid Object. Values:
array([[   9.72      ,   10.69      ,  305.        ],
       [   9.72      ,   10.69      ,  315.        ],
       [   9.72      ,   10.69      ,  325.        ],
       ...,
       [ 970.056     , 1059.28181818,  622.0892334 ],
       [ 970.056     , 1063.16909091,  622.06713867],
       [ 970.056     , 1067.05636364,  622.05786133]])
```

Defining Custom Section

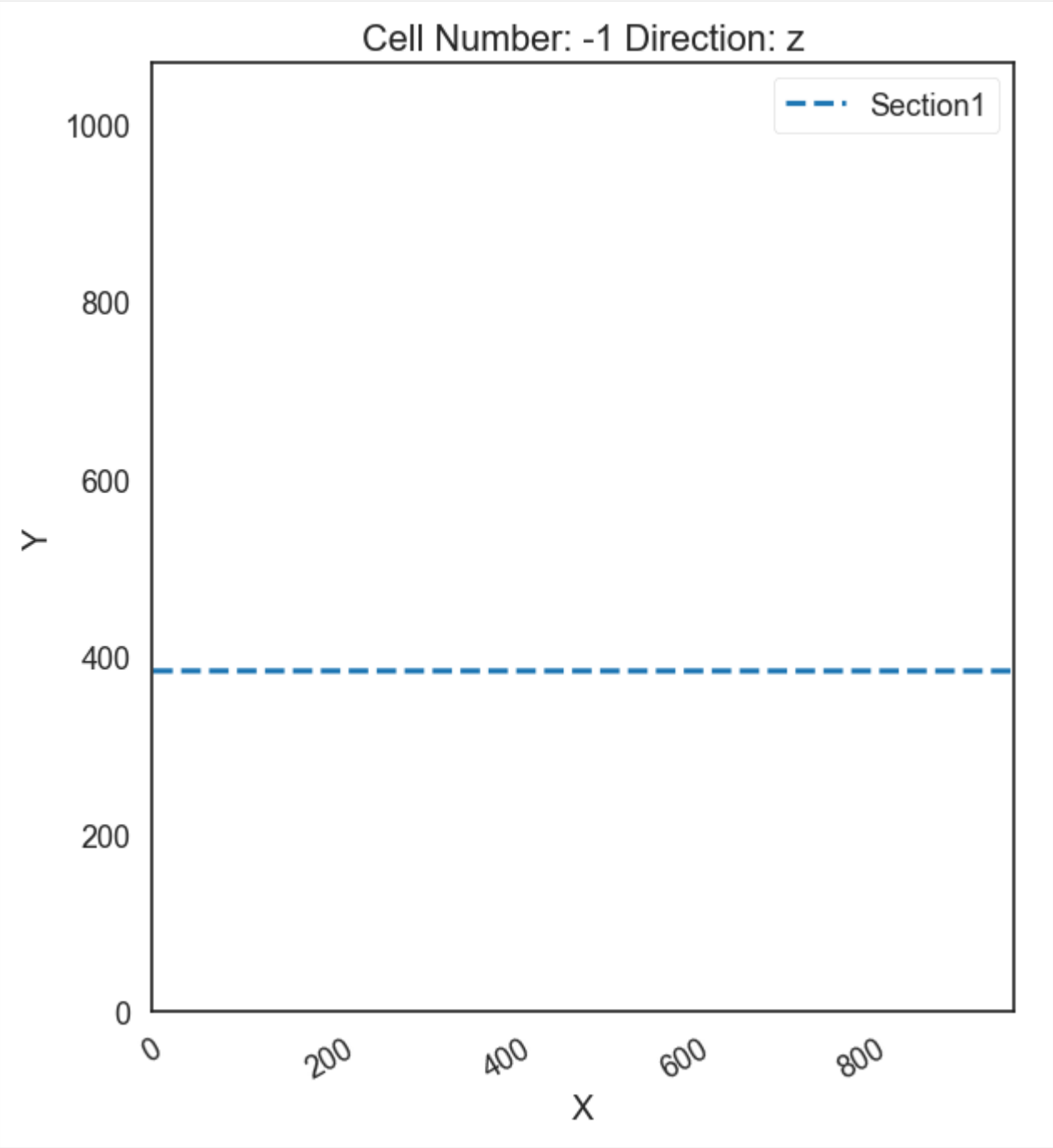
[32]:
```
custom_section = gpd.read_file(file_path + 'customsections1.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='section')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[32]:
```
                                          start                                       stop␣
↪resolution   dist
Section1  [1.372395262185787, 383.9794474025771]  [970.9954955186289, 383.8831909730347]␣
↪ [100, 80] 969.62
```

[33]:
```
gp.plot.plot_section_traces(geo_model)
```

[33]:
```
<gempy.plot.visualization_2d.Plot2D at 0x2b0f5226bb0>
```


Cell Number: -1 Direction: z
Section1
1000
800
600
400
200
0
Y
0
200
400
600
800
X

Setting Interpolator

[34]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range            1528.90
$C_o$           55655.83
drift equations      [3]
```

[34]:
```
<gempy.core.interpolator.InterpolatorModel at 0x2b0f52261c0>
```

Computing Model

[35]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

The surfaces DataFrame now contains values for vertices and edges.

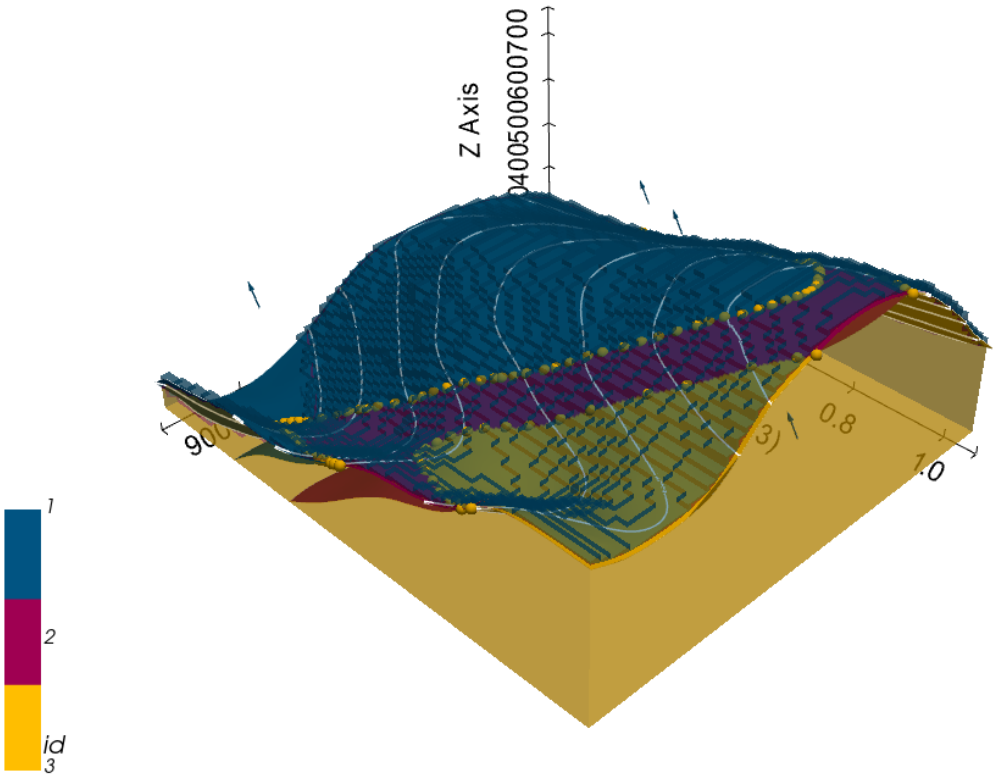
[36]:
```
geo_model.surfaces.df
```

[36]:
```
    surface        series  order_surfaces  isBasement isFault  isActive  \
0     Sand1  Strat_Series               1       False   False      True
1       Ton  Strat_Series               2       False   False      True
2  basement  Strat_Series               3        True   False      True

   hasData    color                                              vertices  \
0     True  #015482  [[29.160000000000004, 194.27877317428587, 305...
1     True  #9f0052  [[29.160000000000004, 365.78652999877926, 305...
2     True  #ffbe00                                                NaN

                                              edges sfai  id
0  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.26   1
1  [[2, 1, 0], [2, 0, 3], [3, 4, 2], [2, 4, 5], [... 0.21   2
2                                                NaN  NaN   3
```

6.63.4 Plotting the 3D Model

[37]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```

```
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\pyvista\plotting\tools.py:571:␣
↪PyvistaDeprecationWarning: The usage of `parse_color` is deprecated in favor of the␣
↪new `Color` class.
  warnings.warn(
C:\Users\jan13846\.conda\envs\gemgis\lib\site-packages\pyvista\jupyter\notebook.py:60:␣
↪UserWarning: Failed to use notebook backend:

Please install `ipyvtklink` to use this feature: https://github.com/Kitware/ipyvtklink

Falling back to a static output.
  warnings.warn(
```



6.63.5 Extracting Shapefiles from Gempy model

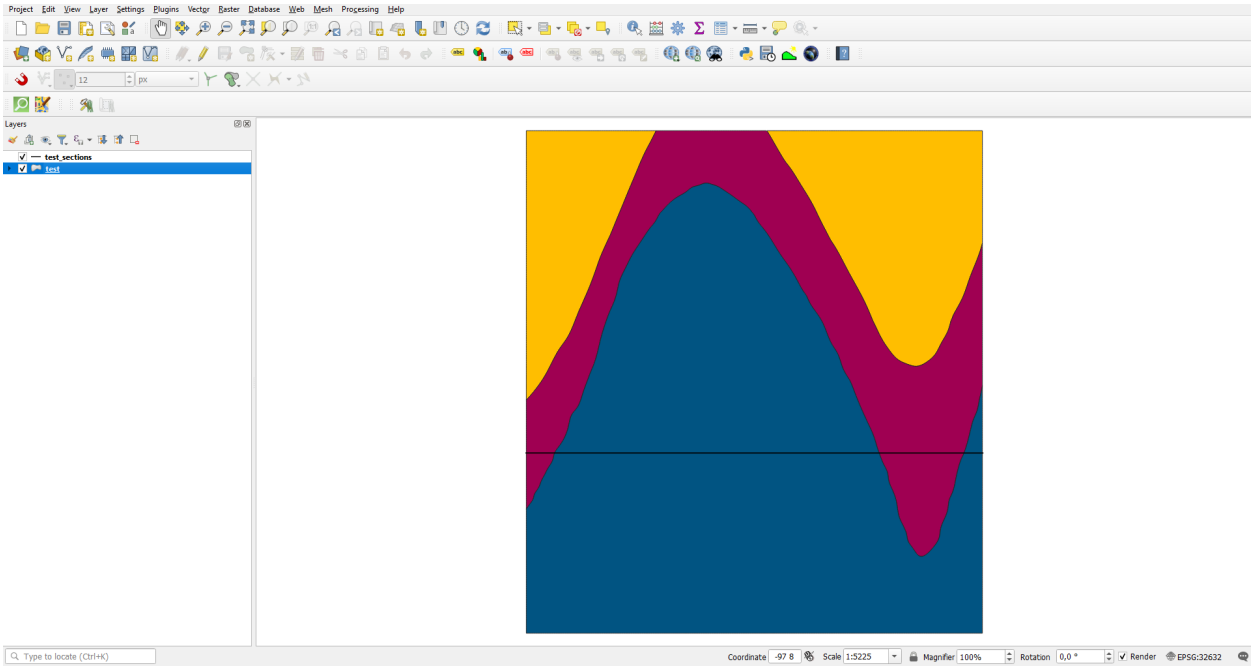
Using the following code we are able to download the custom sections and the geological map as shapefiles

Extracting GemPy Section Lines as Shapefiles

[38]:
```
from shapely.geometry import LineString
gdf_sections = geo_model.grid.sections.df.copy(deep=True)
gdf_sections.reset_index()
linestrings = [LineString((gdf_sections.iloc[i]['start'],gdf_sections.iloc[i]['stop'] ))␣
↪for i in range(len(gdf_sections))]
gdf_sections_new = gpd.GeoDataFrame(geometry=linestrings, data= gdf_sections.reset_
↪index()[['index']], crs='EPSG:32632')
gdf_sections_new.to_file(file_path + 'test_sections.shp')
```

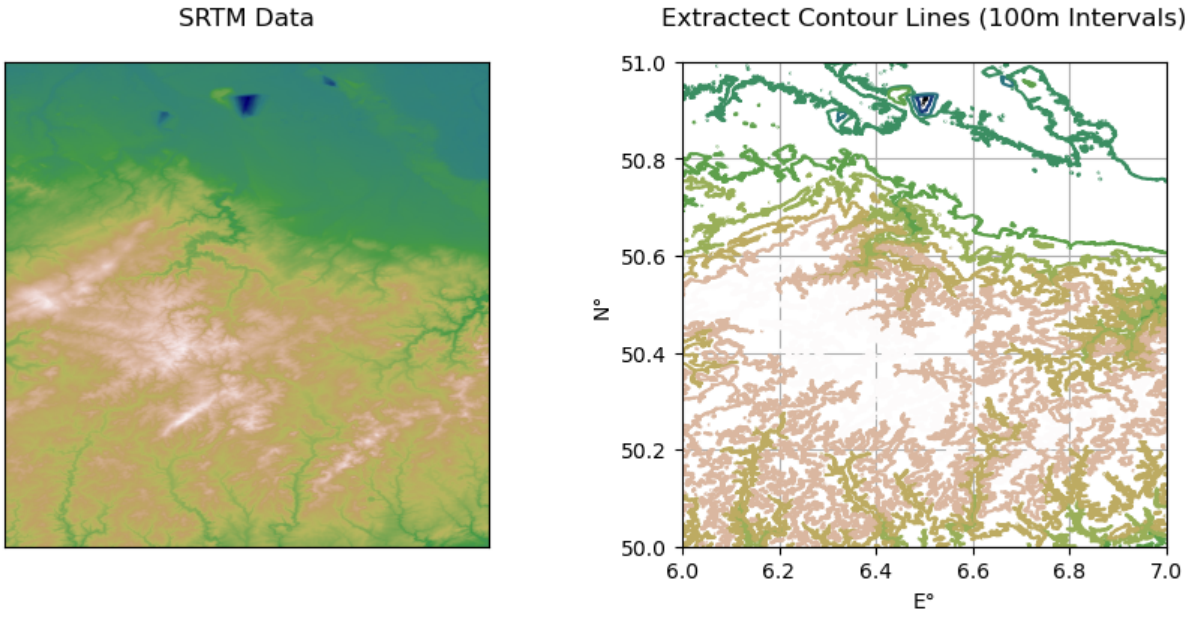
Extracting GemPy Geological Map Polygons as Shapefiles

[39]:
```
gdf = gg.postprocessing.extract_lithologies(geo_model, geo_model.grid.regular_grid.
↪extent[:4], crs='EPSG:32632')
gdf.to_file(file_path + 'test.shp')
```



6.64 62 Extracting contour lines from raster

The following notebook illustrates how to extract contour lines from a raster utilizing `Rasterio`, `Shapely` and `Geopandas`.



6.64.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[12]:
```
import gemgis as gg

file_path = 'data/62_extracting_contour_lines_from_raster/'
```

[13]:
```
gg.download_gemgis_data.download_tutorial_data(filename="62_extracting_contour_lines_
↪from_raster.zip", dirpath=file_path)
```

6.64.2 Loading the data

For this tutorial a SRTM DEM with a resolution of 30 m showing the area around Aachen and the High Fens - Eifel is used. SRTM data is provided by the LP DAAC (https://lpdaac.usgs.gov/product_search/?collections=MEaSUREs+SRTM&view=list&sort=title), which is responsible for the distribution of NASA MEaSUREs data, and can be downloaded for free from different website or through different GIS plugins.

As the `GemGIS` function `extract_contour_lines_from_raster` is able to work with `.tif` files, `raster.io` objects and `np.ndarrays`, multiple ways of loading the data will be shown below.

1) Loading the data as a raster.io object

[30]:
```
import rasterio

raster =  rasterio.open(file_path + 'N50E006.tif')
raster.read(1)
```

[30]:
```
array([[ 80,  81,  82, ...,  46,  46,  45],
       [ 81,  81,  81, ...,  45,  45,  46],
       [ 80,  80,  81, ...,  47,  47,  47],
       ...,
       [399, 400, 402, ..., 197, 196, 194],
       [395, 395, 396, ..., 197, 196, 194],
       [391, 391, 390, ..., 196, 195, 194]], dtype=int16)
```

If the raster is loaded as a raster.io object, the extent and CRS can be extractet easily.

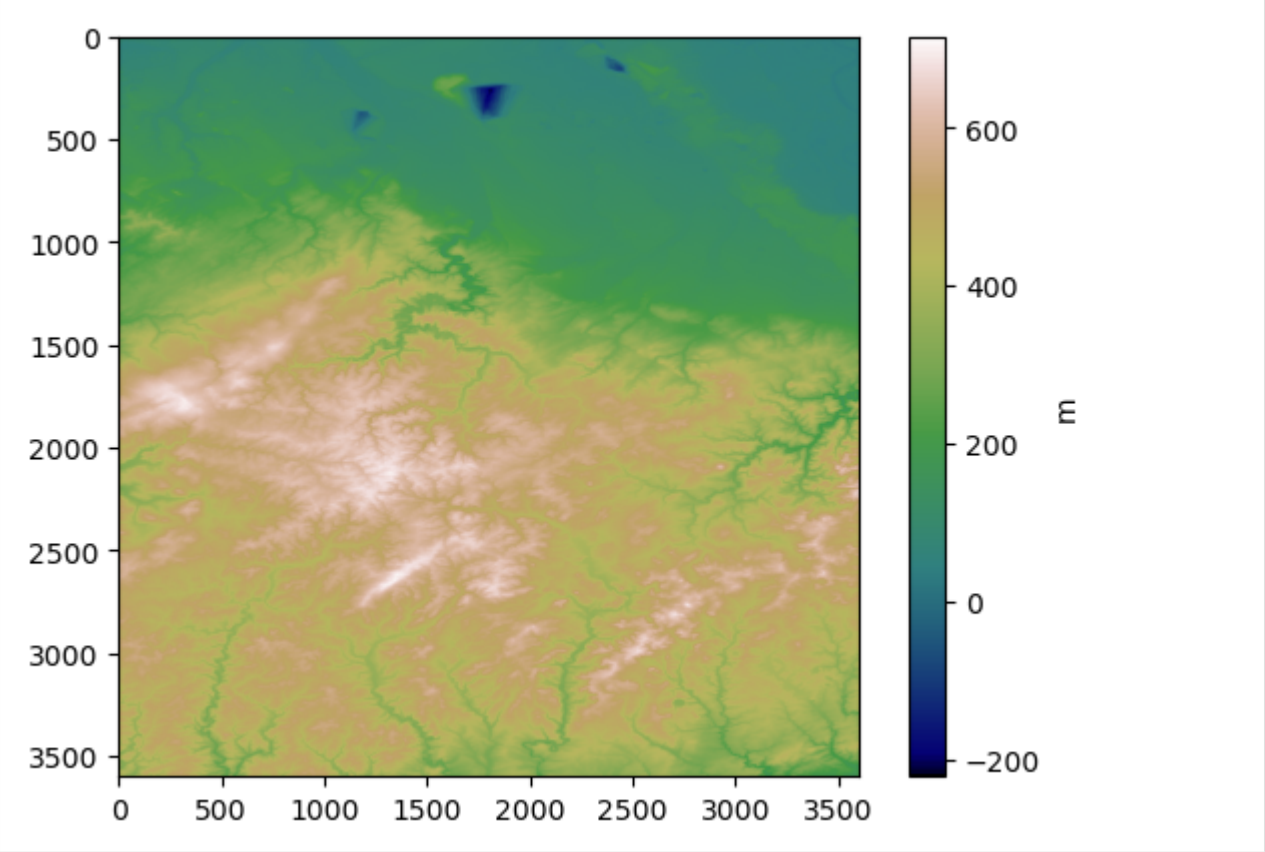
[31]:
```
raster_extent = raster.bounds
raster_extent
```

[31]:
```
BoundingBox(left=5.999861111, bottom=49.999861111, right=7.000138889, top=51.000138889)
```

[32]:
```
raster_crs = raster.crs
raster_crs
```

[32]:
```
CRS.from_epsg(4326)
```

[33]:
```
import matplotlib.pyplot as plt

im = plt.imshow(raster.read(1), cmap ='gist_earth')
cbar = plt.colorbar(im)
cbar.set_label('m')
```

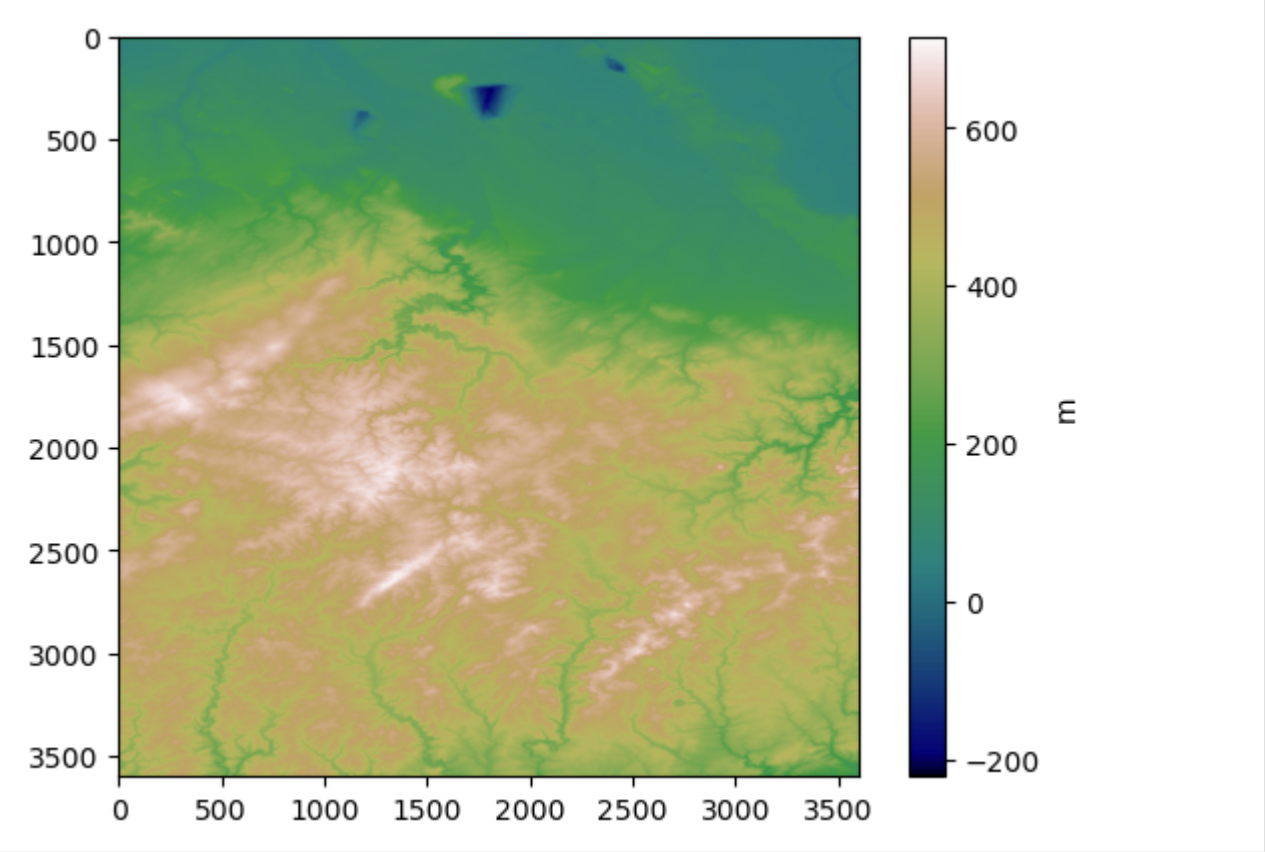


2) Loading the data as an np.ndarray

[27]:
```
import numpy as np

raster = np.load(file_path + 'raster_array.npy')
```

Plotting the raster

[19]:
```
import matplotlib.pyplot as plt

im = plt.imshow(raster, cmap ='gist_earth')
cbar = plt.colorbar(im)
cbar.set_label('m')
```



6.64.3 Extracting Contour Lines

For extracting the contour lines from the raster, it is necessary to provide the raster data as well as the extent from which the contour lines are supposed to extractet from, the crs of the raster and the interval with which contour lines are supposed to be extractet.

For the chosen example for this tutorial, the extent will be set to the SRTM DEM extent, CRS to EPSG:4326 and the interval will be set to 100 meters.

If raster was loaded as raster.io object

[37]:
```
gdf_lines = gg.raster.extract_contour_lines_from_raster(raster= raster,
                                        interval = 100)
```

If raster is present as .tif file

[39]:
```
gdf_lines = gg.raster.extract_contour_lines_from_raster(raster= file_path + 'N50E006.tif
↪',
                                        interval = 100)
```

If raster is present as ndarray

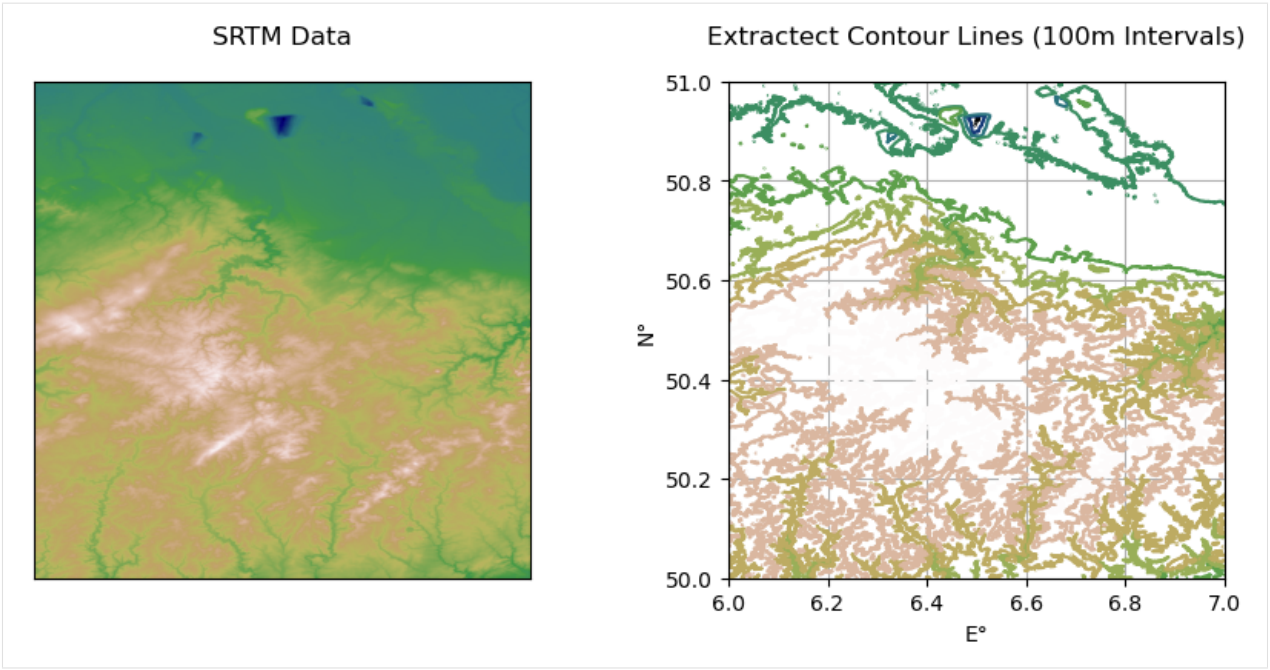
[28]:
```
gdf_lines = gg.raster.extract_contour_lines_from_raster(raster= file_path + 'N50E006.tif
↪',
                                        extent=(5.99, 7.00, 49.99, 51.0),
                                        target_crs='EPSG:4326',
                                        interval = 100)
```

6.64.4 Plotting Contour Lines

With the contours extractet they can now be easily plotted

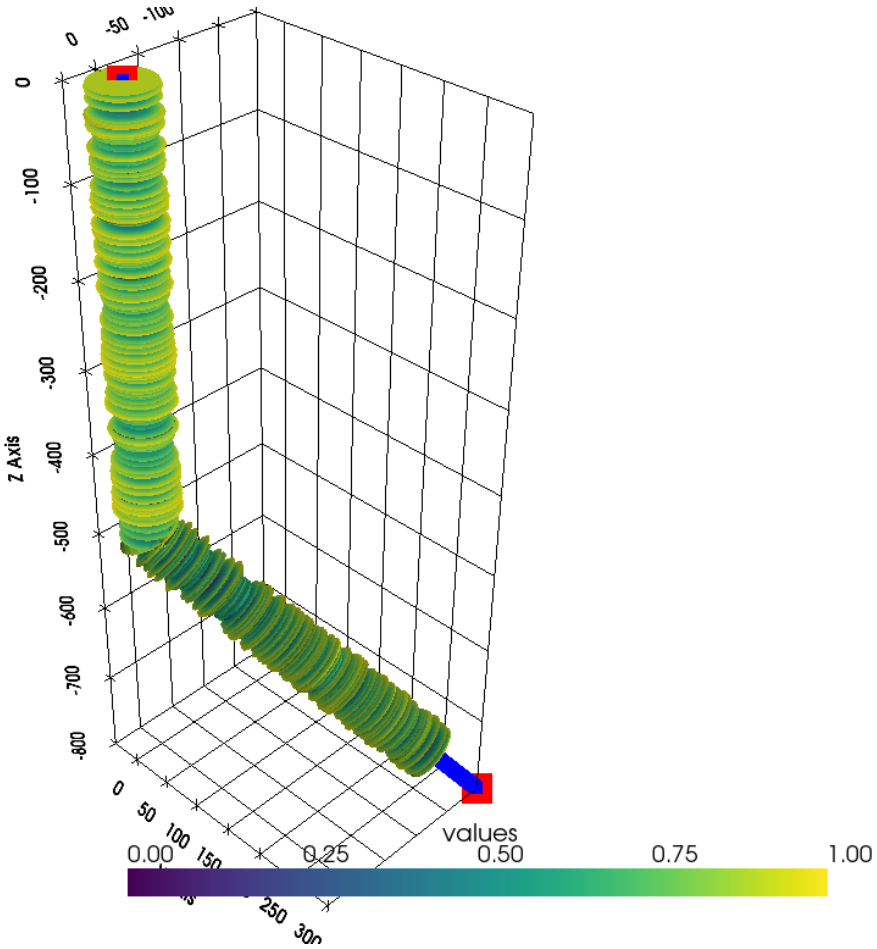
[46]:
```
fig, ax = plt.subplots(1,2, figsize = (10,10))
fig.subplots_adjust(wspace = (0.4))

ax[0].imshow(raster.read(1), cmap ='gist_earth')
ax[0].set_xticks([])
ax[0].set_yticks([])
ax[0].set_title("SRTM Data", y= 1.05)

gdf_lines.plot(column = 'Z', ax = ax[1], cmap ='gist_earth')
ax[1].set_aspect("equal")
ax[1].set_xlabel("E°")
ax[1].set_ylabel("N°")
ax[1].grid()
ax[1].set_xlim(6, 7)
ax[1].set_ylim(50,51)
ax[1].set_title("Extractect Contour Lines (100m Intervals)", y= 1.05)
```

[46]:
```
Text(0.5, 1.05, 'Extractect Contour Lines (100m Intervals)')
```



6.65 63 Displaying Well Log along Well Path

This notebook illustrates how to illustrates how to display a well log consisting of measured depth values and the actual measured values along a well path defined by a well path consisting of x, y, and z coordinates.



6.65.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[2]:
```
import warnings
warnings.filterwarnings("ignore")

import numpy as np
import pyvista as pv
import pandas as pd
from typing import Union
```

6.65.2 Defining well and well log values

First, we define an arbitrary well path using three points as NumPy Array. The measured depths (`dist`) and and some random values are defined as 1D NumPy arrays.

[3]:
```
coordinates = np.array([[0,0,0],
                        [0, 0, -500],
                        [-200, 300,-800]])
pd.DataFrame(coordinates, columns=['X', 'Y', 'Z'])
```

[3]:
```
     X    Y    Z
0    0    0    0
1    0    0 -500
2 -200  300 -800
```

[6]:
```
dist = np.arange(0,901,1)
values = np.random.random(901)
```

[7]:
```
df = pd.DataFrame(dist, columns=['MD'])
df['values'] = values
df
```

[7]:
```
      MD    values
0      0  0.902572
1      1  0.727718
2      2  0.769483
3      3  0.064950
4      4  0.950260
..   ...       ...
896  896  0.416416
897  897  0.417463
898  898  0.848264
899  899  0.881004
900  900  0.732136

[901 rows x 2 columns]
```

6.65.3 Interpolate/Resample between points

The second step is to resample linearly between each provided well path point. A spacing of 5 cm between each point is chosen by default.

[8]:
```
points = gg.visualization.resample_between_well_deviation_points(coordinates)
pd.DataFrame(points, columns=['X', 'Y', 'Z'])
```

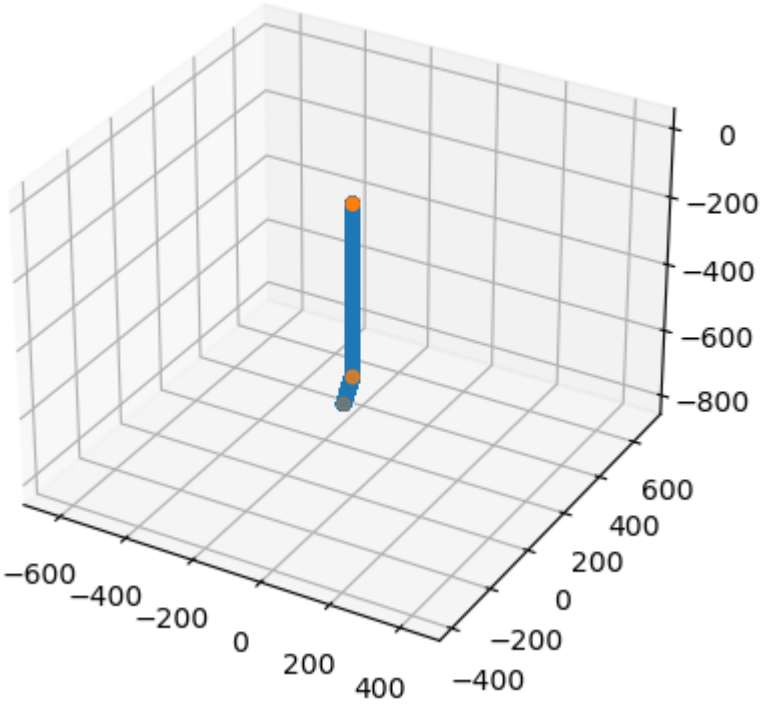
[8]:
```
                X           Y           Z
0        0.000000    0.000000    0.000000
1        0.000000    0.000000   -0.050005
2        0.000000    0.000000   -0.100010
3        0.000000    0.000000   -0.150015
4        0.000000    0.000000   -0.200020
...           ...         ...         ...
19376 -199.914712  299.872068 -799.872068
```

(continues on next page)

(continued from previous page)

```
19377 -199.936034  299.904051 -799.904051
19378 -199.957356  299.936034 -799.936034
19379 -199.978678  299.968017 -799.968017
19380 -200.000000  300.000000 -800.000000

[19381 rows x 3 columns]
```

[9]:
```
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(projection='3d')
ax.scatter(points[:,0], points[:,1], points[:,2])
ax.scatter(coordinates[:,0], coordinates[:,1], coordinates[:,2])
ax.set_aspect('equal')
ax.grid()
```



6.65.4 Create Polyline from well path coordinates

Then, we create a polyline of the original well path for visualization purposes.

[11]:
```
polyline_well_path = gg.visualization.polyline_from_points(coordinates)
polyline_well_path
```

[11]:
```
PolyData (0x298a7635d00)
  N Cells:    1
  N Points:   3
  N Strips:   0
```

(continues on next page)

(continued from previous page)

```
  X Bounds:   -2.000e+02, 0.000e+00
  Y Bounds:   0.000e+00, 3.000e+02
  Z Bounds:   -8.000e+02, 0.000e+00
  N Arrays:   0
```

6.65.5 Creating Spline from resampled well path coordinates

Now, we are creating spline of the resampled points. This automatically calculates the `arc_length` which will be utilized in the next step.

[12]:
```
polyline_well_path_resampled = pv.Spline(points)
polyline_well_path_resampled
```

[12]:
```
PolyData (0x298a7c077c0)
  N Cells:    1
  N Points:   19381
  N Strips:   0
  X Bounds:   -2.000e+02, 1.520e-03
  Y Bounds:   -2.280e-03, 3.000e+02
  Z Bounds:   -8.000e+02, 0.000e+00
  N Arrays:   1
```

6.65.6 Getting the Points along the resampled Spline

The main step is to assign a resampled value to a measured value using `get_points_along_spline`.

[13]:
```
points_along_spline = gg.visualization.get_points_along_spline(polyline_well_path_
↪resampled, dist)
pd.DataFrame(points_along_spline, columns=['X', 'Y', 'Z'])
```

[13]:

	X	Y	Z
0	0.000000	0.000000	0.000000
1	0.000000	0.000000	-1.000043
2	0.000000	0.000000	-2.000086
3	0.000000	0.000000	-3.000129
4	0.000000	0.000000	-4.000172
..	...	...	...
896	-168.850037	253.275055	-753.275085
897	-169.276459	253.914688	-753.914673
898	-169.702881	254.554321	-754.554321
899	-170.129303	255.193954	-755.193970
900	-170.555725	255.833572	-755.833557

[901 rows x 3 columns]

6.65.7 Creating Polyline from Points, assigning values and creating Tube

Once we have extracted the points, we again create a PolyLine from it. We then assign the measured well log values as data array to the newly created PolyLine and create a tube from it using the `values` to define the radius of the tube. The `radius_factor` is a scaling factor that needs to changed according to the length of the well.

[14]:
```
polyline_along_spline = gg.visualization.polyline_from_points(points_along_spline)
polyline_along_spline['values'] = values
tube_along_spline = polyline_along_spline.tube(scalars='values', radius_factor=75)
tube_along_spline
```

[14]:
```
PolyData (0x298a8c46e80)
  N Cells:    22
  N Points:   18060
  N Strips:   22
  X Bounds:   -2.000e+02, 3.747e+01
  Y Bounds:   -3.747e+01, 2.805e+02
  Z Bounds:   -7.807e+02, 0.000e+00
  N Arrays:   2
```

6.65.8 Combined Function

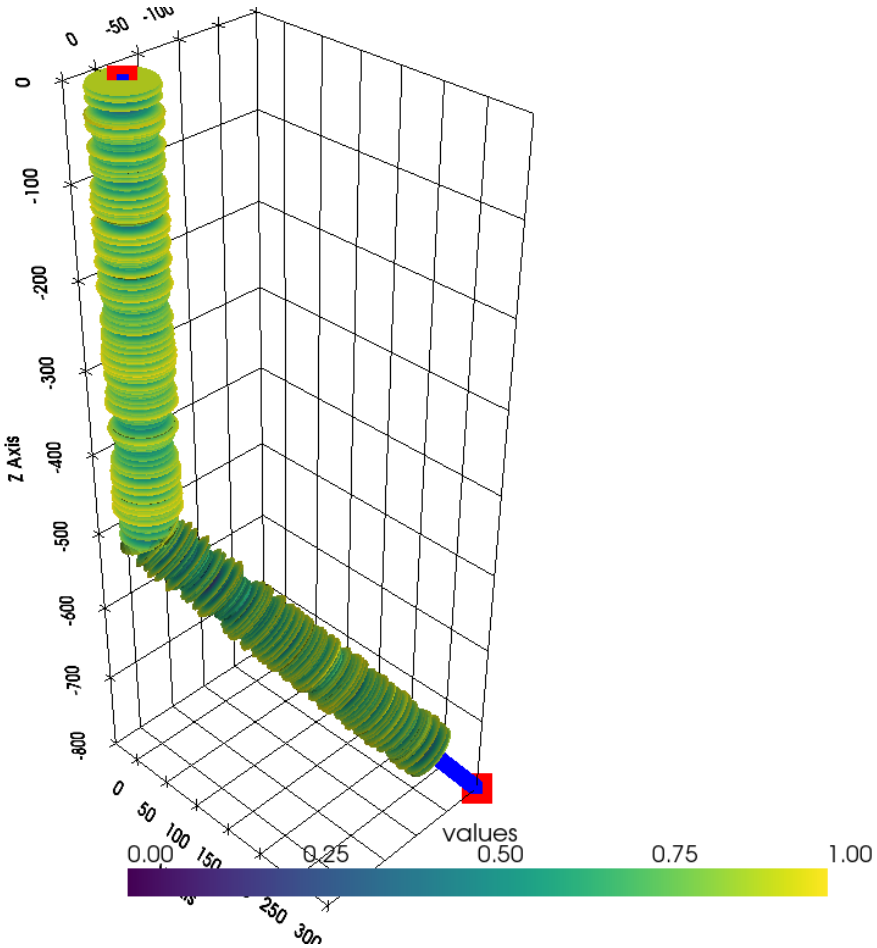
[23]:
```
tube_along_spline = show_well_log_along_well(coordinates, dist, values, radius_factor=75)
tube_along_spline
```

[23]:
```
PolyData (0x298a9080fa0)
  N Cells:    22
  N Points:   18060
  N Strips:   22
  X Bounds:   -2.000e+02, 3.747e+01
  Y Bounds:   -3.747e+01, 2.805e+02
  Z Bounds:   -7.807e+02, 0.000e+00
  N Arrays:   2
```

6.65.9 Plotting the result

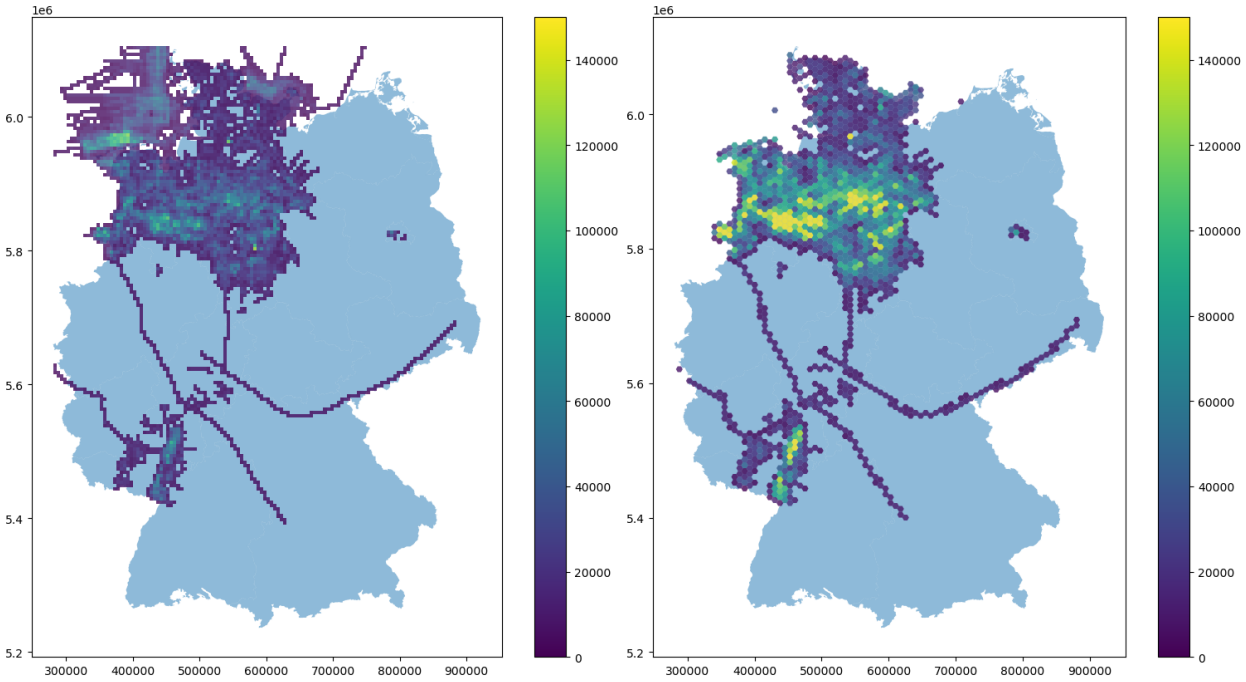
[24]:
```
sargs = dict(fmt="%.2f", color='black')

p = pv.Plotter(notebook=True)

p.add_mesh(tube_along_spline, scalar_bar_args=sargs, clim=[0,1])
p.add_mesh(pv.PolyData(coordinates), color='red', point_size=25)
p.add_mesh(pv.PolyData(points), color='blue', point_size=10)

p.set_background('white')
p.show_grid(color='black')
p.show()
```

```
C:\Users\ale93371\Anaconda3\envs\gempy_new8\lib\site-packages\pyvista\utilities\helpers.
↪py:507: UserWarning: Points is not a float type. This can cause issues when␣
↪transforming or applying filters. Casting to ``np.float32``. Disable this by passing␣
↪``force_float=False``.
  warnings.warn(
C:\Users\ale93371\Anaconda3\envs\gempy_new8\lib\site-packages\pyvista\jupyter\notebook.
↪py:58: UserWarning: Failed to use notebook backend:

No module named 'trame'

Falling back to a static output.
  warnings.warn(
```



6.66 64 Creating Seismic Line Density Maps

This notebook illustrates how to calculate the density of seismic lines (km length per defined area) for a data set provided by the LBEG in Hanover. This workflow can be applied to any other line data to calculate the density of the data.



6.66.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import gemgis as gg
import pandas as pd
import geopandas as gpd
import numpy as np
from shapely.geometry import Polygon
import matplotlib.pyplot as plt
import shapely
from typing import Union
```

[2]:
```
file_path ='data/64_displaying_seismic_line_density/'
gg.download_gemgis_data.download_tutorial_data(filename="64_displaying_seismic_line_
↪density.zip", dirpath=file_path)
```

6.66.2 Loading Outline of Germany

First, we are loading an outline of Germany as reference.

[3]:
```
outline_germany = gpd.read_file(file_path + '250_NUTS1.shp')
print(outline_germany.crs)
outline_germany
```

```
epsg:25832
```

[3]:

	GF	NUTS_LEVEL	NUTS_CODE	NUTS_NAME	geometry
0	4	1	DE1	Baden-Württemberg	MULTIPOLYGON (((579209.636 5345138.889, 579352...
1	4	1	DE2	Bayern	POLYGON ((797853.293 5352195.002, 797799.497 5...
2	4	1	DE3	Berlin	POLYGON ((802831.687 5845501.783, 802754.455 5...
3	4	1	DE4	Brandenburg	MULTIPOLYGON (((819325.396 5702557.885, 819202...
4	4	1	DE5	Bremen	MULTIPOLYGON (((471136.959 5933521.448, 470997...
5	4	1	DE6	Hamburg	MULTIPOLYGON (((467342.107 5975636.297, 467364...
6	4	1	DE7	Hessen	MULTIPOLYGON (((492728.845 5483452.099, 492466...
7	4	1	DE8	Mecklenburg-Vorpommern	MULTIPOLYGON (((790065.392 6006141.945, 790110...
8	4	1	DE9	Niedersachsen	MULTIPOLYGON (((541001.526 5719889.433, 541005...
9	4	1	DEA	Nordrhein-Westfalen	MULTIPOLYGON (((301890.303 5600399.203, 301901...
10	4	1	DEB	Rheinland-Pfalz	POLYGON ((398205.424 5437424.732, 398065.565 5...
11	4	1	DEC	Saarland	POLYGON ((356382.638 5448971.303, 356487.438 5...
12	4	1	DED	Sachsen	POLYGON ((839823.634 5702877.598, 839884.204 5...
13	4	1	DEE	Sachsen-Anhalt	MULTIPOLYGON (((682097.567 5665706.575, 681973...
14	4	1	DEF	Schleswig-Holstein	MULTIPOLYGON (((550941.703 5942349.973, 550503...
15	4	1	DEG	Thüringen	POLYGON ((728949.772 5634682.439, 729000.333 5...
16	2	1	DE1	Baden-Württemberg	MULTIPOLYGON (((507883.567 5282624.777, 507923...
17	2	1	DE5	Bremen	POLYGON ((468447.156 5938186.211, 469137.167 5...
18	2	1	DE6	Hamburg	MULTIPOLYGON (((551291.556 5934133.677, 551330...

(continues on next page)

(continued from previous page)

```
19  MULTIPOLYGON (((647883.205 5982996.453, 647855...
20  MULTIPOLYGON (((379503.515 5909127.653, 383098...
21  MULTIPOLYGON (((524404.744 5961855.658, 524207...
```

6.66.3 Loading Seismic Data

The seismic data provided by the LBEG is also loaded as shape file

[4]:
```
seismic_data = gpd.read_file(file_path + '2D-Seismik_ETRS89.shp').to_crs('EPSG:25832')
seismic_data['length'] = seismic_data.length
seismic_data.head()
```

[4]:
```
    ID  ID_SURVEY                    SURVEYNAME  S_KURZNAME  \
0  182       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
1  185       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
2  187       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
3  188       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
4  189       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91

           LBEG_ARCHI OPERATOR            OP_LANG         OP_NACHFOL  \
0  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
1  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
2  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
3  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
4  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum

   MESSJAHR  MESSBEGINN    MESSENDE     PROFILNAME         KOORDINATE  \
0   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-13  Schusskoordinaten
1   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-16  Schusskoordinaten
2   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-18  Schusskoordinaten
3   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-19  Schusskoordinaten
4   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-21  Schusskoordinaten

  ANREGUNG  RECDAUER  SAMPLING  FOLD  \
0   Airgun      7.00      2.00 81.00
1   Airgun      7.00      2.00 81.00
2   Airgun      7.00      2.00 81.00
3   Airgun      7.00      2.00 81.00
4   Airgun      7.00      2.00 81.00

                                    EINSICHT  \
0  Keine Einsicht ohne Erlaubnis des Eigentümers
1  Keine Einsicht ohne Erlaubnis des Eigentümers
2  Keine Einsicht ohne Erlaubnis des Eigentümers
3  Keine Einsicht ohne Erlaubnis des Eigentümers
4  Keine Einsicht ohne Erlaubnis des Eigentümers

                                        geometry    length
0  LINESTRING (410508.201 6012745.422, 411210.339...  51178.55
1  LINESTRING (428189.885 6100261.568, 428149.814...  82392.83
2  LINESTRING (439890.118 6079493.876, 439879.345...  66871.14
```

(continues on next page)

(continued from previous page)

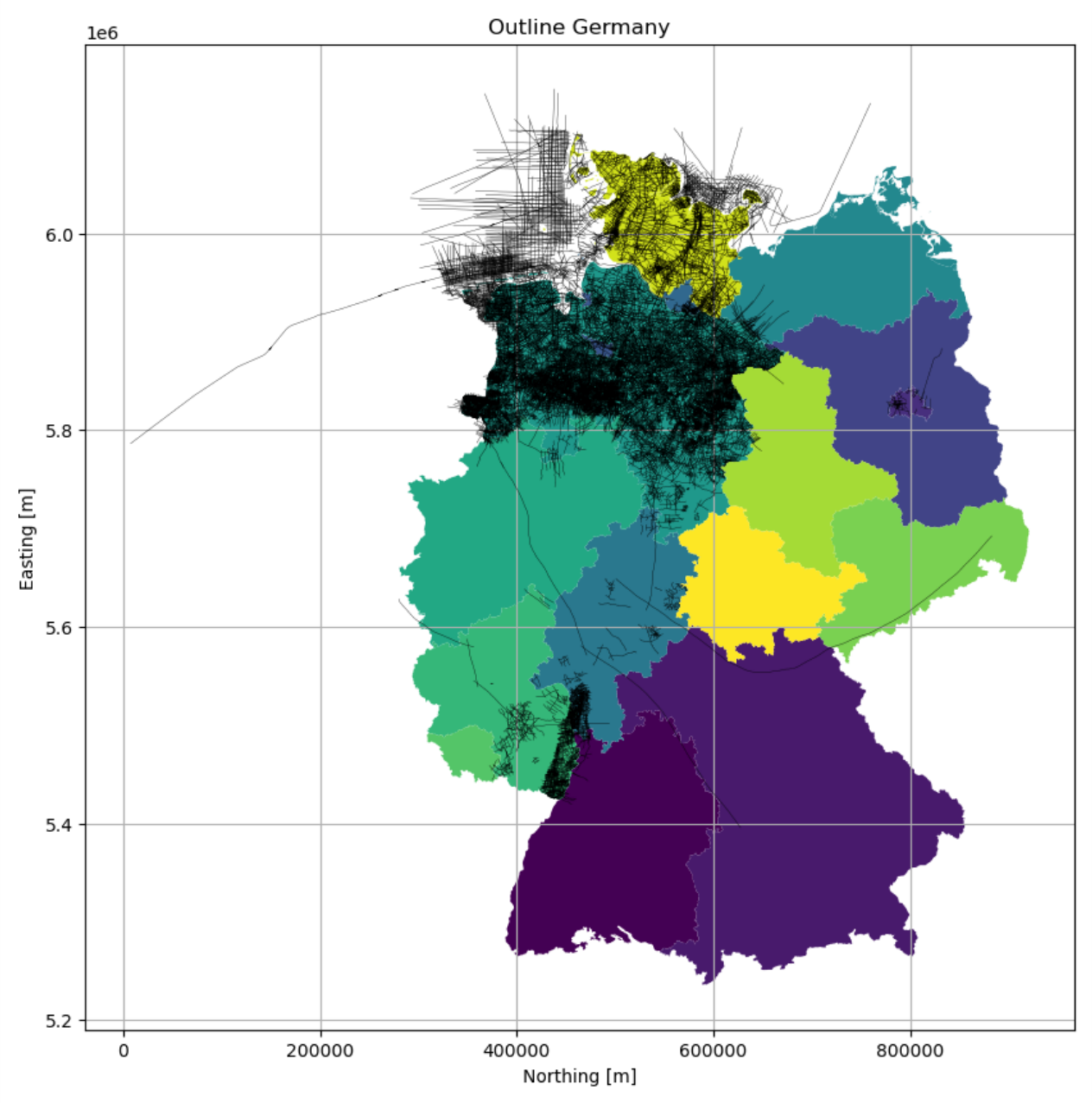
```
3  LINESTRING (302758.022 6025567.558, 303708.941... 151224.04
4  LINESTRING (299479.888 6029068.599, 300252.382... 155473.12
```

6.66.4 Plotting Outline of Germany

The seismic data and the outline of Germany are plotted.

[5]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

outline_germany.plot(ax=ax, column='NUTS_CODE', cmap='viridis')
seismic_data.plot(ax=ax, color='black', linewidth=0.25)
plt.grid()
plt.title('Outline Germany')
plt.xlabel('Northing [m]')
plt.ylabel('Easting [m]')
```

[5]:
```
Text(88.88530637344071, 0.5, 'Easting [m]')
```



6.66.5 Filter settings

The data is filtered for data that was acquired after 1972.

[6]:
```
cell_size = 5000 # m each side
year = 1972
length = 5000 # m
```

[7]:
```
seismic_data_filtered = seismic_data[seismic_data['MESSJAHR']>year]
print(len(seismic_data_filtered))
seismic_data_filtered.head()
```

```
5601
```

[7]:

```
    ID  ID_SURVEY                    SURVEYNAME  S_KURZNAME  \
0  182       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
1  185       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
2  187       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
3  188       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91
4  189       8.00  ARCO 1991, Blocks E, H, J, K  AR-EHJK-91

            LBEG_ARCHI OPERATOR            OP_LANG         OP_NACHFOL  \
0  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
1  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
2  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
3  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum
4  OASYS204070,0123954     ARCO  Arco GmbH Germany  British Petroleum

   MESSJAHR  MESSBEGINN    MESSENDE     PROFILNAME         KOORDINATE  \
0   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-13  Schusskoordinaten
1   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-16  Schusskoordinaten
2   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-18  Schusskoordinaten
3   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-19  Schusskoordinaten
4   1991.00  1991-06-01  1991-06-30  AR-EHJK-91-21  Schusskoordinaten

  ANREGUNG  RECDAUER  SAMPLING  FOLD  \
0   Airgun      7.00      2.00 81.00
1   Airgun      7.00      2.00 81.00
2   Airgun      7.00      2.00 81.00
3   Airgun      7.00      2.00 81.00
4   Airgun      7.00      2.00 81.00

                                    EINSICHT  \
0  Keine Einsicht ohne Erlaubnis des Eigentümers
1  Keine Einsicht ohne Erlaubnis des Eigentümers
2  Keine Einsicht ohne Erlaubnis des Eigentümers
3  Keine Einsicht ohne Erlaubnis des Eigentümers
4  Keine Einsicht ohne Erlaubnis des Eigentümers

                                        geometry    length
0  LINESTRING (410508.201 6012745.422, 411210.339...  51178.55
1  LINESTRING (428189.885 6100261.568, 428149.814...  82392.83
2  LINESTRING (439890.118 6079493.876, 439879.345...  66871.14
3  LINESTRING (302758.022 6025567.558, 303708.941... 151224.04
4  LINESTRING (299479.888 6029068.599, 300252.382... 155473.12
```

6.66.6 Creating Polygon Mask for Germany

Creating a polygon mask for Germany

[8]:
```
from itertools import product

def create_polygon_mask(gdf: gpd.GeoDataFrame,
                        stepsize: int,
                        crs: str = 'EPSG:3034'):
    """Creating a mask GeoDataFrame consisting of squares with a defined stepsize
    Parameters:
    ----------
        gdf: gpd.GeoDataFrame
            GeoDataFrame over which a mask is created
        stepsize: int
            Size of the rasterized squares in meters.
    Returns:
    --------
        gdf_mask: gpd.GeoDataFrame
            GeoDataFrame containing the masked polygons
    """

    # Creating arrays
    x = np.arange(gdf.total_bounds[0], gdf.total_bounds[2], stepsize)
    y = np.arange(gdf.total_bounds[1], gdf.total_bounds[3], stepsize)

    # Creating polygons
    polygons = [Polygon([(a, b), (a + stepsize, b), (a + stepsize, b + stepsize), (a, b␣
↪+ stepsize)]) for a, b in
                product(x, y)]

    # Converting polygons to GeoDataFrame
    gdf_mask = gpd.GeoDataFrame(geometry=polygons,
                                crs=crs)

    return gdf_mask
```

[9]:
```
outline_germany_mask = create_polygon_mask(gdf=outline_germany,
                                           stepsize=cell_size,
                                           crs=outline_germany.crs)
#outline_germany_mask.to_file('outline_germany_mask.shp')
outline_germany_mask
```

[9]:
```
                                                geometry
0      POLYGON ((280371.059 5235855.977, 285371.059 5...
1      POLYGON ((280371.059 5240855.977, 285371.059 5...
2      POLYGON ((280371.059 5245855.977, 285371.059 5...
3      POLYGON ((280371.059 5250855.977, 285371.059 5...
4      POLYGON ((280371.059 5255855.977, 285371.059 5...
...                                                  ...
22441  POLYGON ((920371.059 6080855.977, 925371.059 6...
22442  POLYGON ((920371.059 6085855.977, 925371.059 6...
22443  POLYGON ((920371.059 6090855.977, 925371.059 6...
22444  POLYGON ((920371.059 6095855.977, 925371.059 6...
```

(continues on next page)

(continued from previous page)

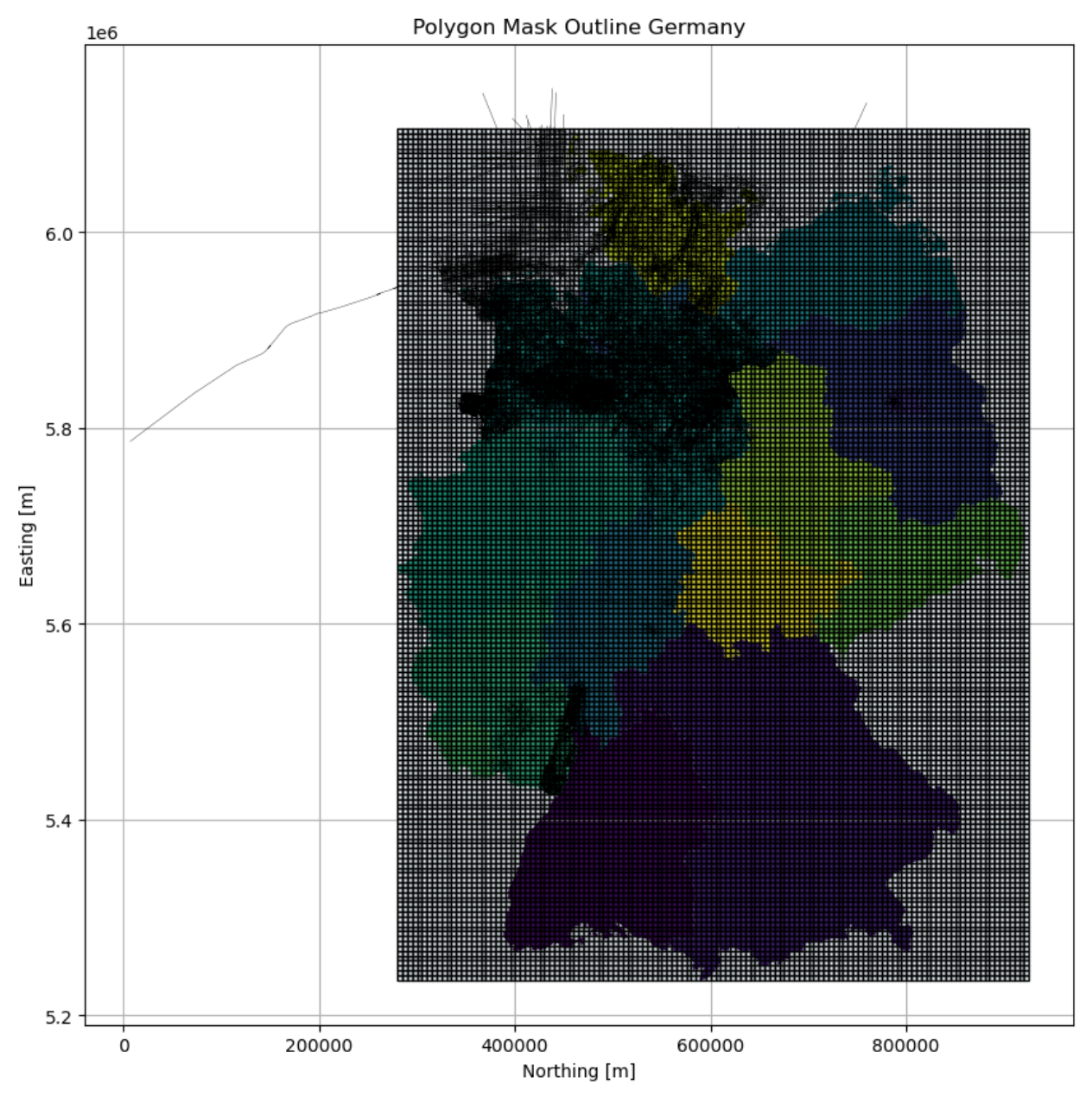
```
22445  POLYGON ((920371.059 6100855.977, 925371.059 6...

[22446 rows x 1 columns]
```

6.66.7 Plotting Polygon Mask for Germany

Plotting the polygon mask, seismic data and the outline of Germany.

[10]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

outline_germany.plot(ax=ax, column='NUTS_CODE', cmap='viridis')
outline_germany_mask.plot(ax=ax, alpha=0.1)
outline_germany_mask.boundary.plot(ax=ax, color='black', linewidth=1)
seismic_data.plot(ax=ax, color='black', linewidth=0.25)
plt.grid()
plt.title('Polygon Mask Outline Germany')
plt.xlabel('Northing [m]')
plt.ylabel('Easting [m]')
```

[10]:
```
Text(87.59722222222221, 0.5, 'Easting [m]')
```



6.66.8 Intersect Seismic Data with Mask

Performing spatial join.

[11]:
```
joined = gpd.sjoin(left_df=outline_germany_mask,
                   right_df=seismic_data_filtered).join(seismic_data_filtered.geometry.
↪rename('line'),
                                                        on='index_right')
joined.head()
```

[11]:
```
                                                geometry  index_right     ID  \
76   POLYGON ((280371.059 5615855.977, 285371.059 5...         3371  10958
```

(continues on next page)

(continued from previous page)

```
77   POLYGON ((280371.059 5620855.977, 285371.059 5...         3371  10958
78   POLYGON ((280371.059 5625855.977, 285371.059 5...         3371  10958
249  POLYGON ((285371.059 5610855.977, 290371.059 5...         3371  10958
250  POLYGON ((285371.059 5615855.977, 290371.059 5...         3371  10958

     ID_SURVEY SURVEYNAME S_KURZNAME LBEG_ARCHI OPERATOR  \
76     1308.00  DEKORP-87  DEKORP-87       None      GFZ
77     1308.00  DEKORP-87  DEKORP-87       None      GFZ
78     1308.00  DEKORP-87  DEKORP-87       None      GFZ
249    1308.00  DEKORP-87  DEKORP-87       None      GFZ
250    1308.00  DEKORP-87  DEKORP-87       None      GFZ

                         OP_LANG OP_NACHFOL  ...    MESSENDE    PROFILNAME  \
76   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
77   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
78   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
249  GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
250  GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A

    KOORDINATE  ANREGUNG RECDAUER SAMPLING   FOLD  \
76        None  Vibrator    16.00     4.00 200.00
77        None  Vibrator    16.00     4.00 200.00
78        None  Vibrator    16.00     4.00 200.00
249       None  Vibrator    16.00     4.00 200.00
250       None  Vibrator    16.00     4.00 200.00

                                       EINSICHT   length  \
76   Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
77   Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
78   Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
249  Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
250  Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75

                                              line
76   LINESTRING (279995.529 5628459.873, 280200.709...
77   LINESTRING (279995.529 5628459.873, 280200.709...
78   LINESTRING (279995.529 5628459.873, 280200.709...
249  LINESTRING (279995.529 5628459.873, 280200.709...
250  LINESTRING (279995.529 5628459.873, 280200.709...

[5 rows x 22 columns]
```

Assigning the length of the seismic lines to the DataFrame.

[12]:
```
joined['length'] = joined.geometry.intersection(joined.line).length
joined.head()
```

[12]:
```
                                          geometry  index_right     ID  \
76   POLYGON ((280371.059 5615855.977, 285371.059 5...         3371  10958
77   POLYGON ((280371.059 5620855.977, 285371.059 5...         3371  10958
78   POLYGON ((280371.059 5625855.977, 285371.059 5...         3371  10958
249  POLYGON ((285371.059 5610855.977, 290371.059 5...         3371  10958
250  POLYGON ((285371.059 5615855.977, 290371.059 5...         3371  10958
```

(continues on next page)

(continued from previous page)

```
     ID_SURVEY SURVEYNAME S_KURZNAME LBEG_ARCHI OPERATOR  \
76     1308.00  DEKORP-87  DEKORP-87       None      GFZ
77     1308.00  DEKORP-87  DEKORP-87       None      GFZ
78     1308.00  DEKORP-87  DEKORP-87       None      GFZ
249    1308.00  DEKORP-87  DEKORP-87       None      GFZ
250    1308.00  DEKORP-87  DEKORP-87       None      GFZ

                          OP_LANG OP_NACHFOL  ...    MESSENDE    PROFILNAME  \
76   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
77   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
78   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
249  GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
250  GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A

    KOORDINATE  ANREGUNG RECDAUER SAMPLING   FOLD  \
76        None  Vibrator    16.00     4.00 200.00
77        None  Vibrator    16.00     4.00 200.00
78        None  Vibrator    16.00     4.00 200.00
249       None  Vibrator    16.00     4.00 200.00
250       None  Vibrator    16.00     4.00 200.00

                                       EINSICHT  length  \
76   Keine Einsicht ohne Erlaubnis des Eigentümers  843.61
77   Keine Einsicht ohne Erlaubnis des Eigentümers 6446.32
78   Keine Einsicht ohne Erlaubnis des Eigentümers 1293.79
249  Keine Einsicht ohne Erlaubnis des Eigentümers 1480.18
250  Keine Einsicht ohne Erlaubnis des Eigentümers 5957.47

                                                 line
76   LINESTRING (279995.529 5628459.873, 280200.709...
77   LINESTRING (279995.529 5628459.873, 280200.709...
78   LINESTRING (279995.529 5628459.873, 280200.709...
249  LINESTRING (279995.529 5628459.873, 280200.709...
250  LINESTRING (279995.529 5628459.873, 280200.709...

[5 rows x 22 columns]
```

Summing up the lengths and assigning the values to the mask

[13]:
```
total_length_per_box = joined.groupby(level=0).length.sum()
```

[14]:
```
outline_germany_mask['length'] = total_length_per_box
outline_germany_mask
```

[14]:
```
                                             geometry  length
0      POLYGON ((280371.059 5235855.977, 285371.059 5...     NaN
1      POLYGON ((280371.059 5240855.977, 285371.059 5...     NaN
2      POLYGON ((280371.059 5245855.977, 285371.059 5...     NaN
3      POLYGON ((280371.059 5250855.977, 285371.059 5...     NaN
4      POLYGON ((280371.059 5255855.977, 285371.059 5...     NaN
...                                                  ...     ...
```

(continues on next page)

(continued from previous page)

```
22441  POLYGON ((920371.059 6080855.977, 925371.059 6...     NaN
22442  POLYGON ((920371.059 6085855.977, 925371.059 6...     NaN
22443  POLYGON ((920371.059 6090855.977, 925371.059 6...     NaN
22444  POLYGON ((920371.059 6095855.977, 925371.059 6...     NaN
22445  POLYGON ((920371.059 6100855.977, 925371.059 6...     NaN

[22446 rows x 2 columns]
```

Create GeoDataFrame from Data

[15]:
```
gdf_buffered = gpd.GeoDataFrame(geometry=[outline_germany_mask[outline_germany_mask[
↪'length']>0].unary_union.buffer(5000)])
gdf_buffered
```

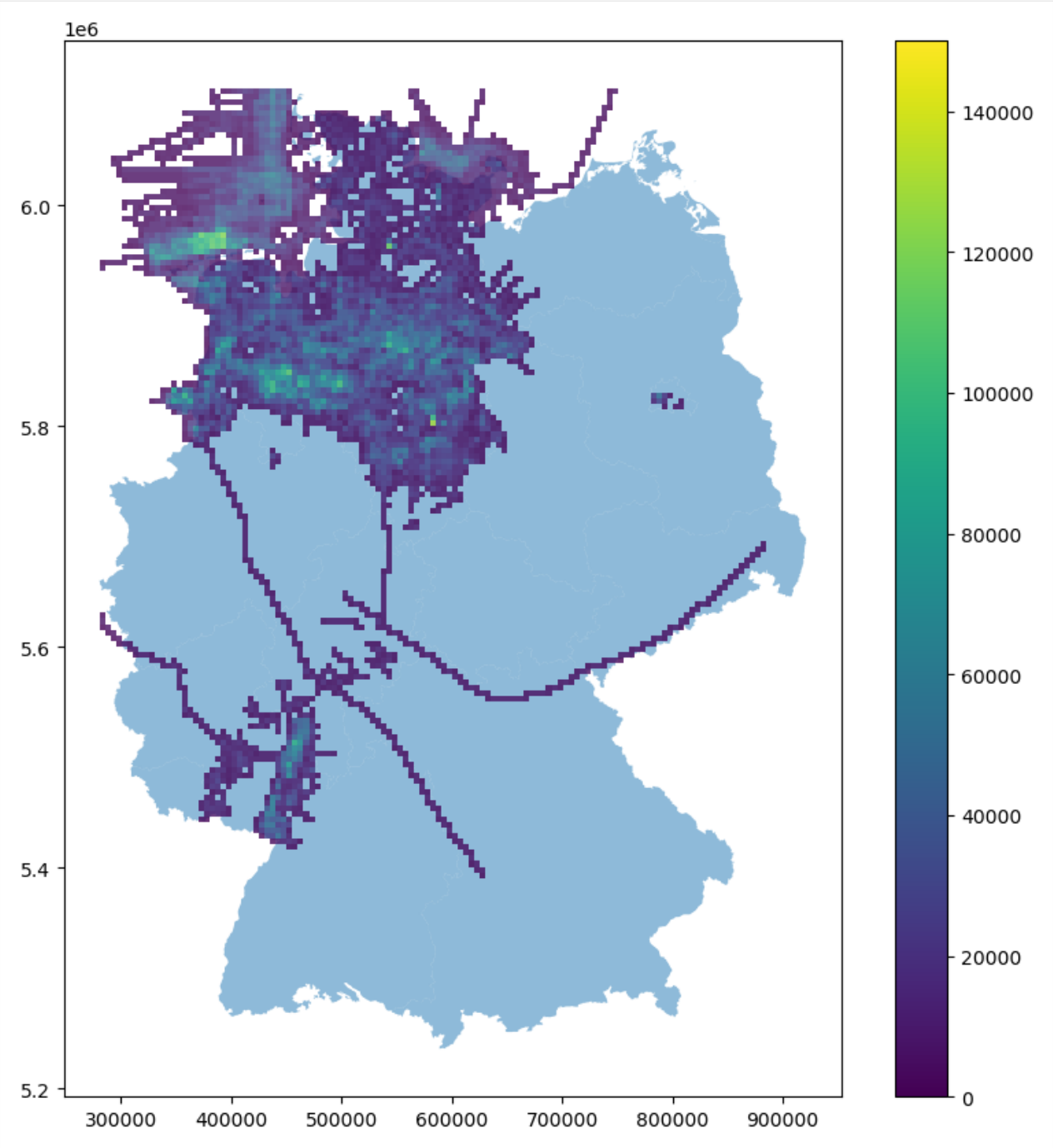
[15]:
```
                                            geometry
0  MULTIPOLYGON (((296047.173 5958355.977, 295961...
```

6.66.9 Plotting results

Plotting the seismic line density

[16]:
```
fig, ax = plt.subplots(1, figsize=(10,10))
outline_germany.plot(ax=ax, alpha=0.5)
# outline_germany_mask.plot(ax=ax, alpha=0.1)
# gdf_buffered.plot(ax=ax)
outline_germany_mask[outline_germany_mask['length']>0].plot(ax=ax, alpha=0.8, column=
↪'length', legend=True, vmin=0, vmax=150000)
# outline_germany_mask.boundary.plot(ax=ax, color='black', linewidth=1)
```

[16]:
```
<AxesSubplot: >
```


1e6
6.0
5.8
5.6
5.4
5.2
300000
400000
500000
600000
700000
800000
900000
140000
120000
100000
80000
60000
40000
20000
0

6.66.10 Merging Outline

[17]:
```
gdf = gpd.GeoDataFrame(geometry=[outline_germany.unary_union], crs='EPSG:25832')
gdf
```

[17]:
```
                                            geometry
0  MULTIPOLYGON (((356678.002 5449479.768, 356695...
```

6.66.11 Creating Hexagon Grid

[18]:
```
hex_gdf = gg.vector.create_hexagon_grid(gdf=outline_germany.explode(),radius=5000, crop_
↪gdf=True).drop_duplicates()
hex_gdf.head()
```

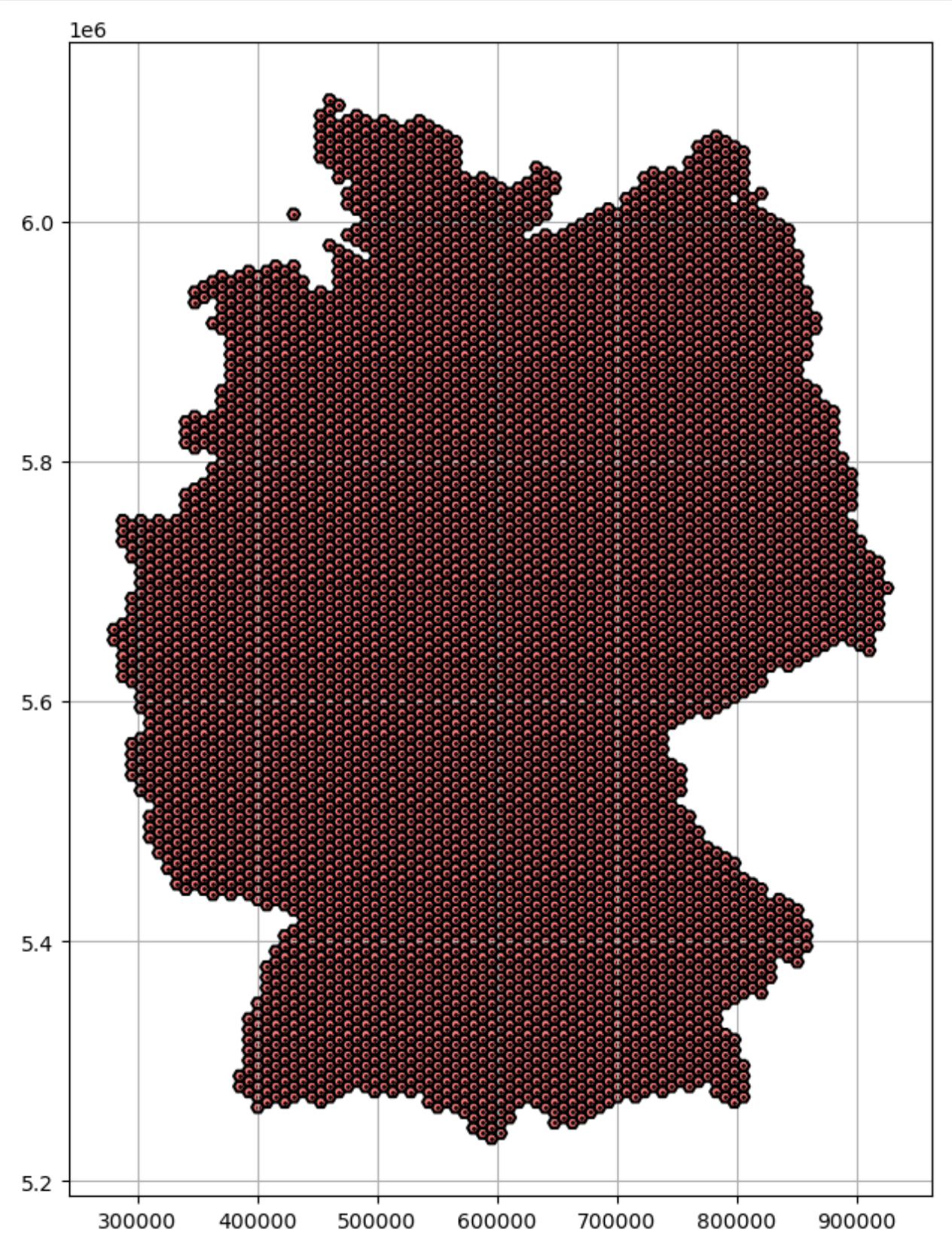
[18]:
```
                                            geometry
0  POLYGON ((282871.059 5664144.049, 277871.059 5...
1  POLYGON ((282871.059 5655483.795, 277871.059 5...
2  POLYGON ((290371.059 5755076.716, 285371.059 5...
3  POLYGON ((290371.059 5746416.462, 285371.059 5...
4  POLYGON ((290371.059 5737756.208, 285371.059 5...
```

[19]:
```
hex_gdf.crs
```

[19]:
```
<Derived Projected CRS: EPSG:25832>
Name: ETRS89 / UTM zone 32N
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Europe between 6°E and 12°E: Austria; Belgium; Denmark - onshore and offshore;␣
↪Germany - onshore and offshore; Norway including - onshore and offshore; Spain -␣
↪offshore.
- bounds: (6.0, 38.76, 12.01, 84.33)
Coordinate Operation:
- name: UTM zone 32N
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989 ensemble
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

6.66.12 Plotting Hexagon Grid

[20]:
```
fig, ax = plt.subplots(1, figsize=(10,10))


# polygon_gdf.plot(ax=ax)
hex_gdf.plot(ax=ax, color='red', alpha=0.5)
hex_gdf.exterior.plot(ax=ax, color='black')
hex_gdf.centroid.plot(ax=ax, color='black', markersize=1)
plt.grid()
```


1e6
6.0
5.8
5.6
5.4
5.2
300000
400000
500000
600000
700000
800000
900000

6.66.13 Calculating seismic line density

[21]:
```
joined = gpd.sjoin(left_df=hex_gdf,
                   right_df=seismic_data_filtered).join(seismic_data_filtered.geometry.
↪rename('line'),
                                                        on='index_right')
joined.head()
```

[21]:
```
                                              geometry  index_right     ID  \
10   POLYGON ((290371.059 5625172.906, 285371.059 5...         3371  10958
41   POLYGON ((305371.059 5607852.397, 300371.059 5...         3371  10958
60   POLYGON ((312871.059 5603522.270, 307871.059 5...         3371  10958
81   POLYGON ((320371.059 5599192.143, 315371.059 5...         3371  10958
102  POLYGON ((327871.059 5594862.016, 322871.059 5...         3371  10958

     ID_SURVEY SURVEYNAME S_KURZNAME LBEG_ARCHI OPERATOR  \
10     1308.00  DEKORP-87  DEKORP-87       None      GFZ
41     1308.00  DEKORP-87  DEKORP-87       None      GFZ
60     1308.00  DEKORP-87  DEKORP-87       None      GFZ
81     1308.00  DEKORP-87  DEKORP-87       None      GFZ
102    1308.00  DEKORP-87  DEKORP-87       None      GFZ

                          OP_LANG OP_NACHFOL  ...    MESSENDE    PROFILNAME  \
10   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
41   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
60   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
81   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
102  GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A

    KOORDINATE  ANREGUNG RECDAUER SAMPLING   FOLD  \
10        None  Vibrator    16.00     4.00 200.00
41        None  Vibrator    16.00     4.00 200.00
60        None  Vibrator    16.00     4.00 200.00
81        None  Vibrator    16.00     4.00 200.00
102       None  Vibrator    16.00     4.00 200.00

                                       EINSICHT   length  \
10   Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
41   Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
60   Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
81   Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75
102  Keine Einsicht ohne Erlaubnis des Eigentümers 93317.75

                                                  line
10   LINESTRING (279995.529 5628459.873, 280200.709...
41   LINESTRING (279995.529 5628459.873, 280200.709...
60   LINESTRING (279995.529 5628459.873, 280200.709...
81   LINESTRING (279995.529 5628459.873, 280200.709...
102  LINESTRING (279995.529 5628459.873, 280200.709...

[5 rows x 22 columns]
```

[22]:
```
joined['length'] = joined.geometry.intersection(joined.line).length
joined.head()
```

[22]:
```
                                              geometry  index_right     ID  \
10   POLYGON ((290371.059 5625172.906, 285371.059 5...         3371  10958
41   POLYGON ((305371.059 5607852.397, 300371.059 5...         3371  10958
60   POLYGON ((312871.059 5603522.270, 307871.059 5...         3371  10958
81   POLYGON ((320371.059 5599192.143, 315371.059 5...         3371  10958
102  POLYGON ((327871.059 5594862.016, 322871.059 5...         3371  10958

     ID_SURVEY SURVEYNAME S_KURZNAME LBEG_ARCHI OPERATOR  \
10     1308.00  DEKORP-87  DEKORP-87       None      GFZ
41     1308.00  DEKORP-87  DEKORP-87       None      GFZ
60     1308.00  DEKORP-87  DEKORP-87       None      GFZ
81     1308.00  DEKORP-87  DEKORP-87       None      GFZ
102    1308.00  DEKORP-87  DEKORP-87       None      GFZ

                          OP_LANG OP_NACHFOL  ...    MESSENDE    PROFILNAME  \
10   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
41   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
60   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
81   GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A
102  GeoForschungsZentrum Potsdam       None  ...  1987-08-31  DEKORP`87-1A

    KOORDINATE  ANREGUNG RECDAUER SAMPLING   FOLD  \
10        None  Vibrator    16.00     4.00 200.00
41        None  Vibrator    16.00     4.00 200.00
60        None  Vibrator    16.00     4.00 200.00
81        None  Vibrator    16.00     4.00 200.00
102       None  Vibrator    16.00     4.00 200.00

                                      EINSICHT  length  \
10   Keine Einsicht ohne Erlaubnis des Eigentümers 7712.61
41   Keine Einsicht ohne Erlaubnis des Eigentümers 8676.04
60   Keine Einsicht ohne Erlaubnis des Eigentümers 8718.85
81   Keine Einsicht ohne Erlaubnis des Eigentümers 8697.56
102  Keine Einsicht ohne Erlaubnis des Eigentümers 8730.73

                                                 line
10   LINESTRING (279995.529 5628459.873, 280200.709...
41   LINESTRING (279995.529 5628459.873, 280200.709...
60   LINESTRING (279995.529 5628459.873, 280200.709...
81   LINESTRING (279995.529 5628459.873, 280200.709...
102  LINESTRING (279995.529 5628459.873, 280200.709...

[5 rows x 22 columns]
```

[23]:
```
total_length_per_box = joined.groupby(level=0).length.sum()
```

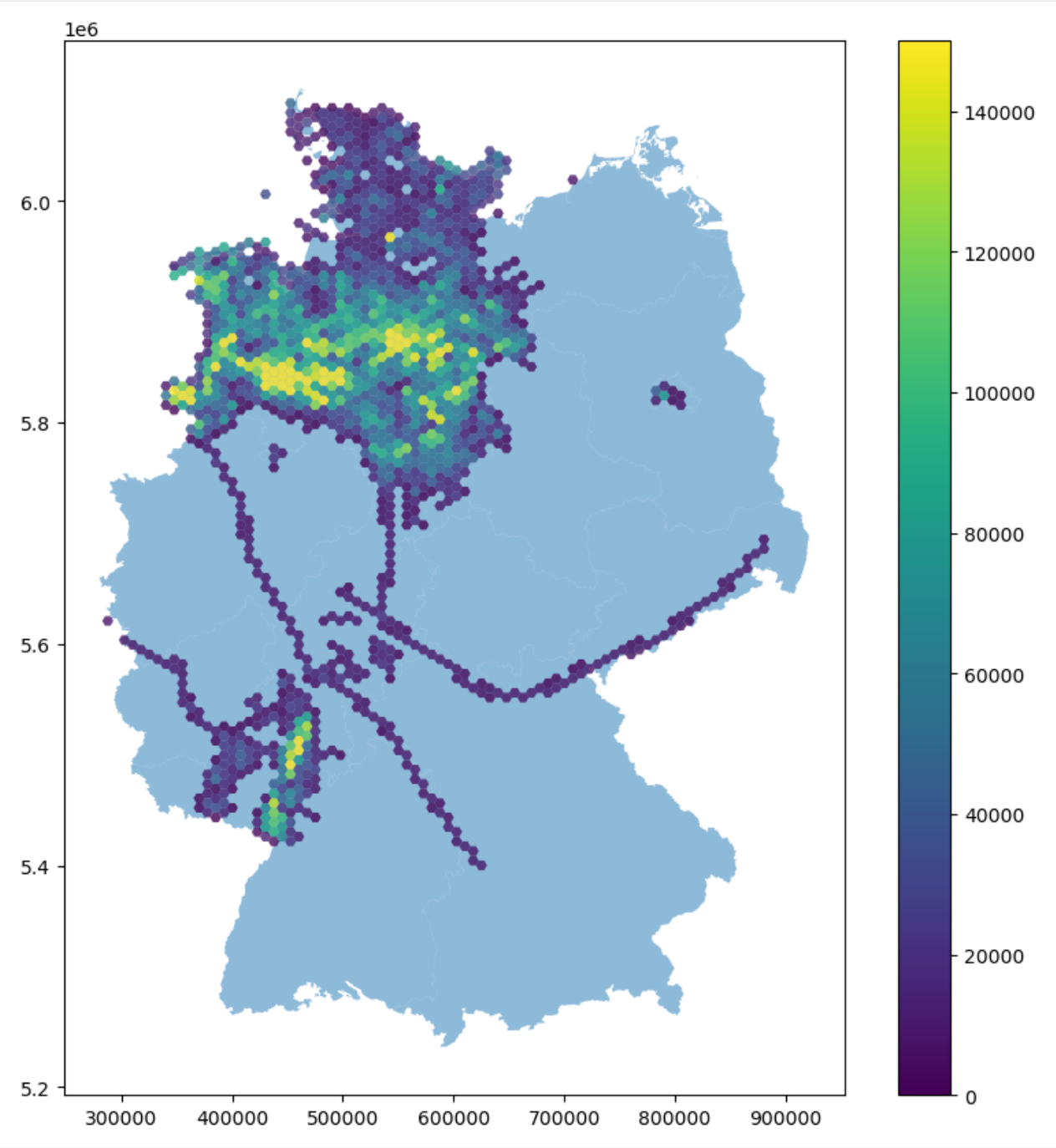
[24]:
```
outline_germany_mask = hex_gdf
outline_germany_mask['length'] = total_length_per_box
outline_germany_mask
```

[24]:
```
                                              geometry  length
0     POLYGON ((282871.059 5664144.049, 277871.059 5...     NaN
1     POLYGON ((282871.059 5655483.795, 277871.059 5...     NaN
2     POLYGON ((290371.059 5755076.716, 285371.059 5...     NaN
3     POLYGON ((290371.059 5746416.462, 285371.059 5...     NaN
4     POLYGON ((290371.059 5737756.208, 285371.059 5...     NaN
...                                                 ...     ...
6873  POLYGON ((845371.059 5997563.829, 840371.059 5...     NaN
6874  POLYGON ((845371.059 5988903.575, 840371.059 5...     NaN
6894  POLYGON ((845371.059 5980243.321, 840371.059 5...     NaN
6897  POLYGON ((852871.059 5975913.194, 847871.059 5...     NaN
6899  POLYGON ((822871.059 6027874.718, 817871.059 6...     NaN

[5872 rows x 2 columns]
```

6.66.14 Plotting seismic line density

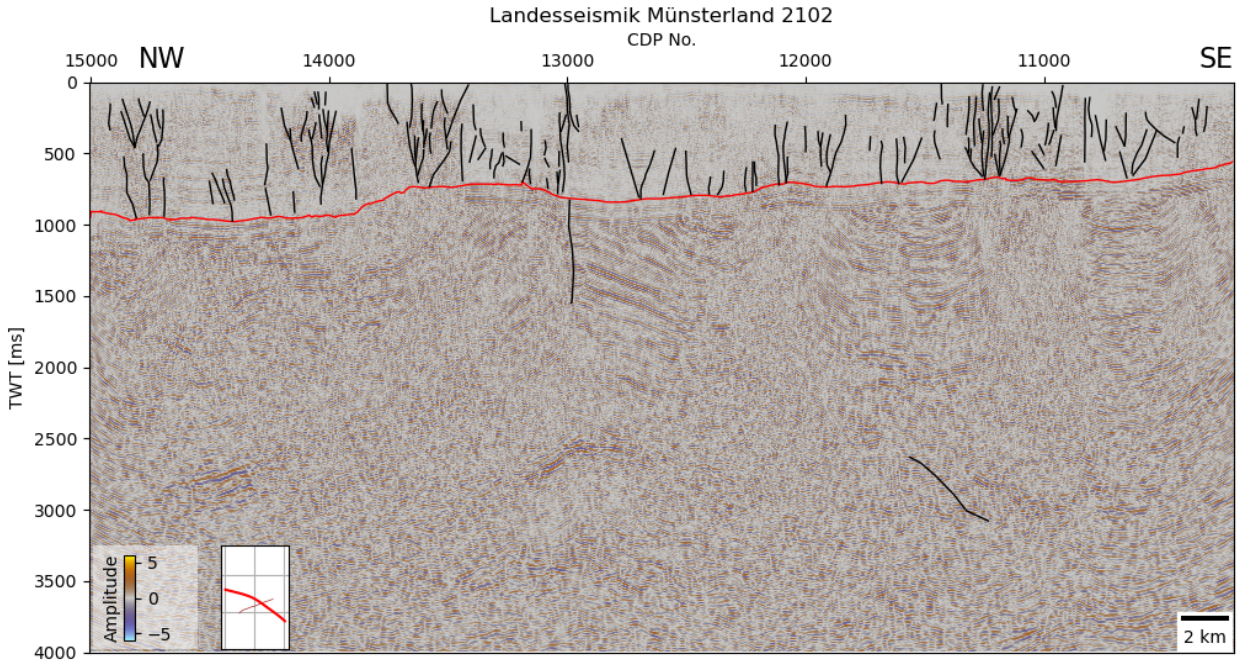
[25]:
```
fig, ax = plt.subplots(1, figsize=(10,10))
outline_germany.plot(ax=ax, alpha=0.5)
# outline_germany_mask.plot(ax=ax, alpha=0.1)
# gdf_buffered.plot(ax=ax)
outline_germany_mask[outline_germany_mask['length']>0].plot(ax=ax, alpha=0.8, column=
↪'length', legend=True, vmin=0, vmax=150000)
# outline_germany_mask.boundary.plot(ax=ax, color='black', linewidth=1)
```

[25]:
```
<AxesSubplot: >
```


1e6
6.0
5.8
5.6
5.4
5.2
300000
400000
500000
600000
700000
800000
900000
140000
120000
100000
80000
60000
40000
20000
0

6.67 65 Displaying Seismic Horizons and Faults

This notebook illustrates how to display seismic interpretations created in Petrel and exported as Shape Files in Python. The seismic data was acquired in 2021 and was obtained from the Geological Survey of NRW.



6.67.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import gemgis as gg
import pandas as pd
import geopandas as gpd
import numpy as np
import matplotlib.pyplot as plt
import shapely
from typing import Union
from segysak.segy import segy_loader
```

[2]:
```
file_path ='data/65_displaying_seismic_horizons_and_faults/'
gg.download_gemgis_data.download_tutorial_data(filename="65_displaying_seismic_horizons_
↪and_faults.zip", dirpath=file_path)
```

6.67.2 Opening traces of seismic lines

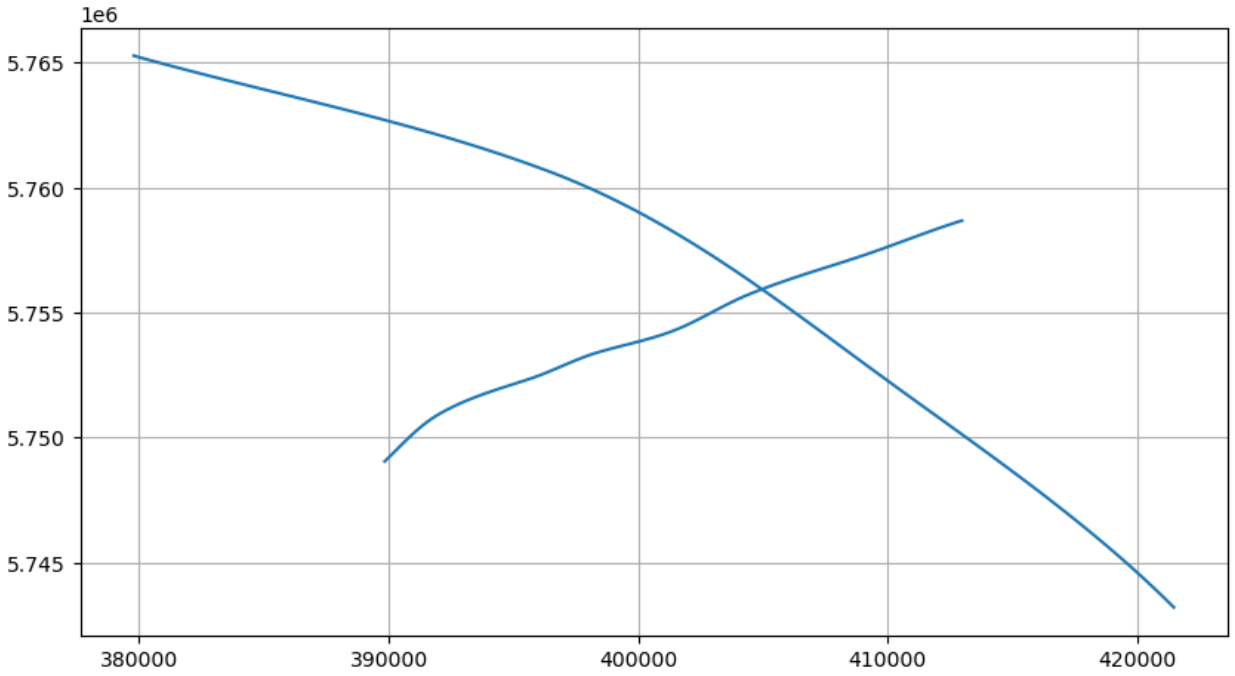
For this tutorial, data of the Landesseismik Münsterland (see https://www.gd.nrw.de/zip/gd_report_2301s.pdf as reference) is used.

[3]:
```
seismic = gpd.read_file(file_path + 'Seismic_Lines_EPSG25832.shp')
seismic
```

[3]:
```
       Profile   length                                           geometry
0  GD_NRW_2101 25436.00  LINESTRING (389796.000 5749009.000, 389810.000...
1  GD_NRW_2102 48120.00  LINESTRING (379724.000 5765301.500, 379734.000...
```

6.67.3 Plotting the traces

[4]:
```
figseismic, ax = plt.subplots(1, figsize=(10,10))

seismic.plot(ax=ax)

plt.grid()
```



6.67.4 Loading the seismic data using segysak

The seismic data is loaded using segysak (https://segysak.readthedocs.io/en/latest/).

[5]:
```
landesseismik2102 = segy_loader(file_path + 'Muenster_2D_PreSTM_Stack_2102_AGC.sgy',␣
↪vert_domain="TWT")
landesseismik2102
```

```
  0%|          | 0.00/4.81k [00:00<?, ? traces/s]
```

```
Loading as 2D
```

```
Converting SEGY:   0%|          | 0.00/4.81k [00:00<?, ? traces/s]
```

[5]:
```
<xarray.Dataset>
Dimensions:  (cdp: 4806, twt: 1751)
Coordinates:
  * cdp      (cdp) uint16 10203 10204 10205 10206 ... 15005 15006 15007 15008
  * twt      (twt) float64 0.0 4.0 8.0 12.0 ... 6.992e+03 6.996e+03 7e+03
    cdp_x    (cdp) float32 4.215e+05 4.215e+05 4.215e+05 ... 3.797e+05 3.797e+05
    cdp_y    (cdp) float32 5.743e+06 5.743e+06 5.743e+06 ... 5.765e+06 5.765e+06
Data variables:
    data     (cdp, twt) float32 0.03581 -0.02099 -0.0793 ... -0.01154 0.04135
Attributes: (12/13)
    ns:                  None
    sample_rate:         4.0
    text:                C 1 Client       Geologischer Dienst NRW\nC 2 Contra...
    measurement_system:  m
    d3_domain:           None
    epsg:                None
    ...                  ...
    corner_points_xy:    None
    source_file:         Muenster_2D_PreSTM_Stack_2102_AGC.sgy
    srd:                 None
    datatype:            None
    percentiles:         [-3.0195783868438255, -2.780390889497134, -1.2279652...
    coord_scalar:        -100.0
```

6.67.5 Converting seismic data to DataFrame

The seismic data can also be converted to a DataFrame to better inspect the data.

[6]:
```
landesseismik2102.to_dataframe()
```

[6]:
```
               data     cdp_x      cdp_y
cdp   twt
10203 0.00     0.04 421512.00 5743166.00
      4.00    -0.02 421512.00 5743166.00
      8.00    -0.08 421512.00 5743166.00
      12.00   -0.12 421512.00 5743166.00
      16.00   -0.12 421512.00 5743166.00
...             ...       ...        ...
15008 6984.00 -0.11 379724.00 5765301.50
      6988.00 -0.03 379724.00 5765301.50
      6992.00 -0.03 379724.00 5765301.50
```

(continues on next page)

(continued from previous page)

```
      6996.00 -0.01 379724.00 5765301.50
      7000.00  0.04 379724.00 5765301.50

[8415306 rows x 3 columns]
```

6.67.6 Getting the seismic colorbar

We can load a color bar for displaying the seismic data. The colorbars were uploaded to the following repository: https://github.com/lperozzi/Seismic_colormaps

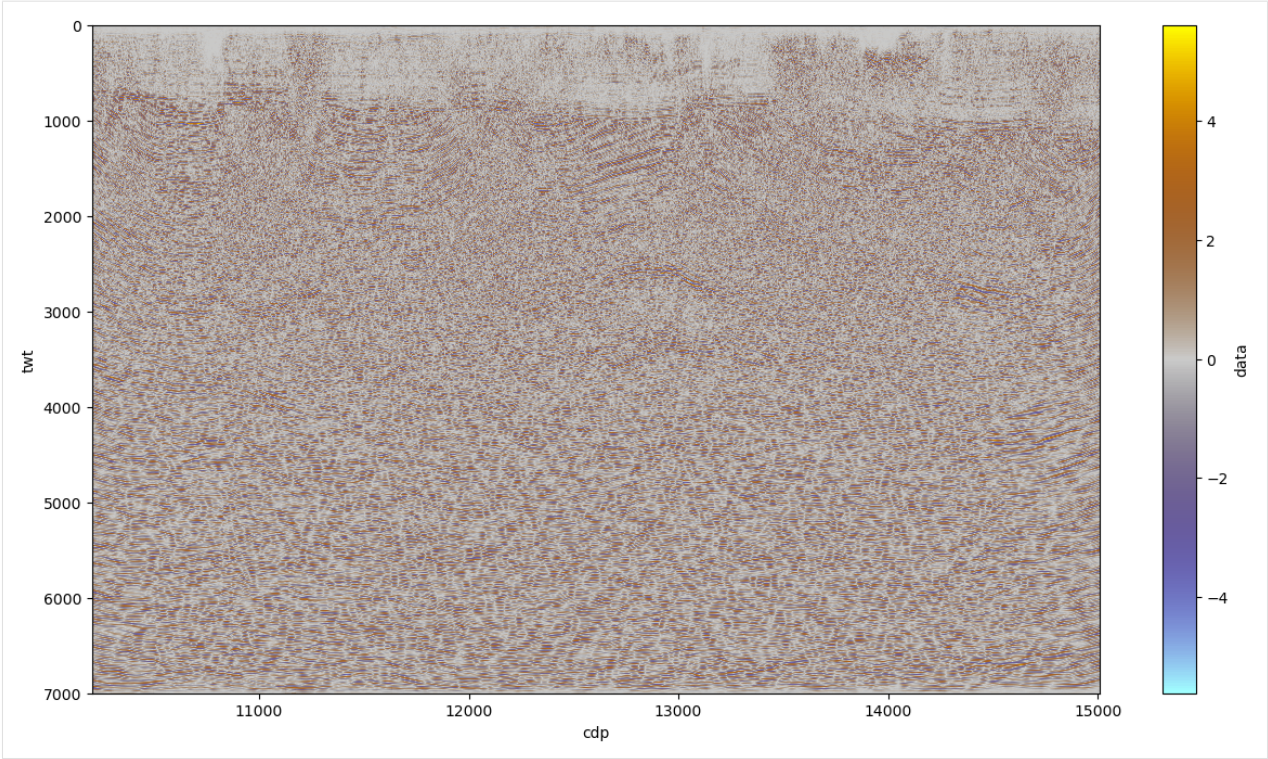
[7]:
```
cmap_seismic = gg.visualization.get_seismic_cmap()
type(cmap_seismic)
```

[7]:
```
matplotlib.colors.ListedColormap
```

6.67.7 Plotting the seismic data

The seismic data is plotted the built-in plotting function

[8]:
```
fig, ax = plt.subplots(ncols=1, figsize=(15, 8))

landesseismik2102.data.transpose().plot(cmap=cmap_seismic)

plt.gca().invert_yaxis()
```



6.67.8 Converting seismic to array

The loaded seismic data is converted into a NumPy array.

[10]:
```
landesseismik2102_array = gg.visualization.seismic_to_array(seismic_
↪data=landesseismik2102, max_depth=4000)
landesseismik2102_array
```

[10]:
```
array([[ 0.03581395, -0.02099377, -0.07930481, ..., -0.3949225 ,
        -0.64953464, -0.88681436],
       [ 0.03755527, -0.02159597, -0.08407128, ..., -0.43278015,
        -0.706827  , -0.9397521 ],
       [ 0.04379281, -0.01925519, -0.09289104, ..., -0.45442533,
        -0.7610789 , -1.0000496 ],
       ...,
       [-0.19376808, -0.18929183, -0.09974575, ..., -1.1590624 ,
        -0.46177298,  0.48989367],
       [-0.16737616, -0.15299934, -0.07530349, ..., -1.0952702 ,
        -0.30956793,  0.64396167],
       [-0.15387064, -0.13157344, -0.06080835, ..., -1.0007935 ,
        -0.14463931,  0.77823424]], dtype=float32)
```

The same dataset can be obtained using the built-in xarray attributes and methods.

[11]:
```
landesseismik2102.data.transpose()
```

[11]:
```
<xarray.DataArray 'data' (twt: 1751, cdp: 4806)>
array([[ 0.03581395,  0.03755527,  0.04379281, ..., -0.19376808,
```

(continues on next page)

(continued from previous page)

```
        -0.16737616, -0.15387064],
       [-0.02099377, -0.02159597, -0.01925519, ..., -0.18929183,
        -0.15299934, -0.13157344],
       [-0.07930481, -0.08407128, -0.09289104, ..., -0.09974575,
        -0.07530349, -0.06080835],
       ...,
       [ 0.02854965,  0.03610878,  0.03544764, ..., -0.03994363,
        -0.0334743 , -0.02788334],
       [-0.3293916 , -0.32478482, -0.32344526, ..., -0.02570425,
        -0.01836406, -0.01154189],
       [-0.3131938 , -0.3141619 , -0.31639624, ...,  0.03036585,
         0.0359443 ,  0.04135381]], dtype=float32)
Coordinates:
  * cdp      (cdp) uint16 10203 10204 10205 10206 ... 15005 15006 15007 15008
  * twt      (twt) float64 0.0 4.0 8.0 12.0 ... 6.992e+03 6.996e+03 7e+03
    cdp_x    (cdp) float32 4.215e+05 4.215e+05 4.215e+05 ... 3.797e+05 3.797e+05
    cdp_y    (cdp) float32 5.743e+06 5.743e+06 5.743e+06 ... 5.765e+06 5.765e+06
```

6.67.9 Loading Base Cretaceous Horizon - Landesseismik 2102

The Base Cretaceous Horizon that was interpreted in Petrel is loaded as Shape File.

[12]:
```
base_cretaceous_2102 = gpd.read_file(file_path + 'U1_2102.shp')
base_cretaceous_2102 = gg.vector.extract_xyz(base_cretaceous_2102)
base_cretaceous_2102['Name'] = '2102'
base_cretaceous_2102['Z'] = base_cretaceous_2102['Z']*(-1)-150
base_cretaceous_2102.head()
```

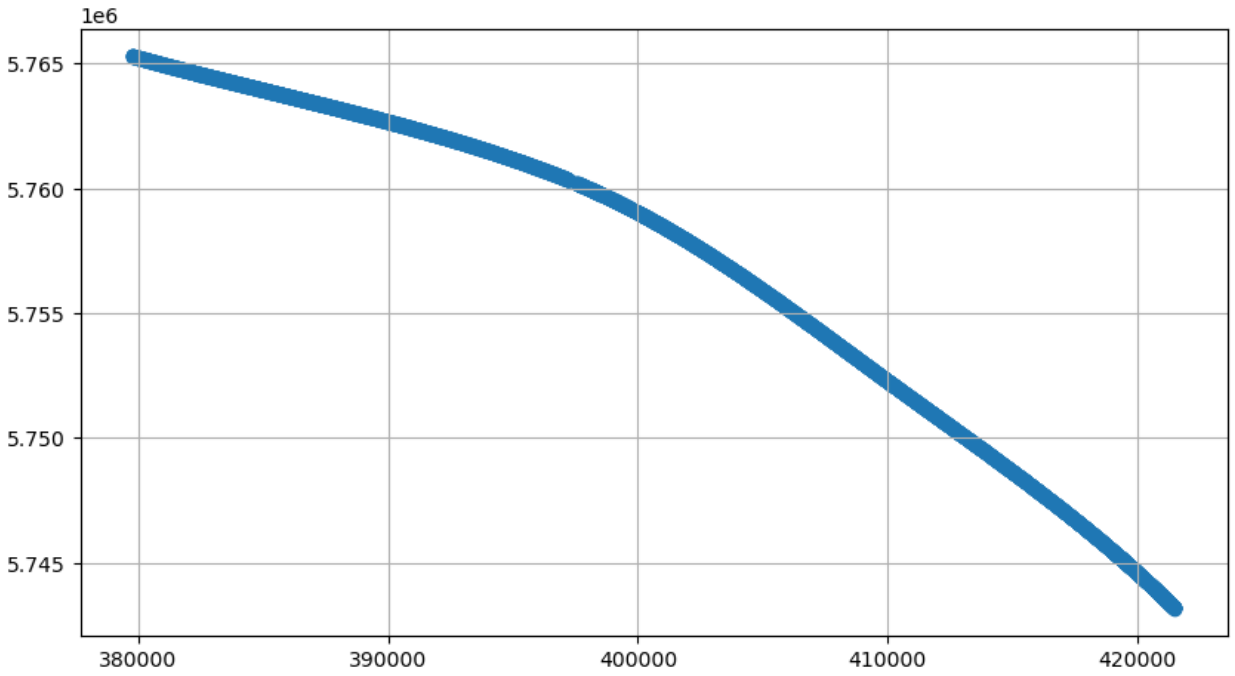
[12]:
```
              Type   Domain
0  Seismic horizon  Unknown  \
1  Seismic horizon  Unknown
2  Seismic horizon  Unknown
3  Seismic horizon  Unknown
4  Seismic horizon  Unknown

                                          Droid  Comment
0  ://1d9a2dd1-dd1d-4676-a92e-6057e64d33c2/cb463e...      NaN  \
1  ://1d9a2dd1-dd1d-4676-a92e-6057e64d33c2/cb463e...      NaN
2  ://1d9a2dd1-dd1d-4676-a92e-6057e64d33c2/cb463e...      NaN
3  ://1d9a2dd1-dd1d-4676-a92e-6057e64d33c2/cb463e...      NaN
4  ://1d9a2dd1-dd1d-4676-a92e-6057e64d33c2/cb463e...      NaN

         ShapeName                                              Project
0  Base Cretaceous  C:\Users\Nicklas.Ackermann\Desktop\Muensterlan...  \
1  Base Cretaceous  C:\Users\Nicklas.Ackermann\Desktop\Muensterlan...
2  Base Cretaceous  C:\Users\Nicklas.Ackermann\Desktop\Muensterlan...
3  Base Cretaceous  C:\Users\Nicklas.Ackermann\Desktop\Muensterlan...
4  Base Cretaceous  C:\Users\Nicklas.Ackermann\Desktop\Muensterlan...

                                  geometry         X          Y      Z  Name
0  POINT Z (421497.819 5743179.962 -708.911) 421497.82 5743179.96 558.91  2102
```

(continues on next page)

(continued from previous page)

```
1  POINT Z (421490.638 5743186.924 -708.911) 421490.64 5743186.92 558.91  2102
2  POINT Z (421483.457 5743193.886 -710.030) 421483.46 5743193.89 560.03  2102
3  POINT Z (421476.276 5743200.848 -711.149) 421476.28 5743200.85 561.15  2102
4  POINT Z (421469.095 5743207.810 -711.894) 421469.10 5743207.81 561.89  2102
```

[13]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

base_cretaceous_2102.plot(ax=ax)
plt.grid()
```



6.67.10 Calculating Position on Seismic Line Base Cretaceous

The x, y, z positions are converted to positions along the seismic line.

[14]:
```
line = seismic.loc[1].geometry
line
```

[14]:

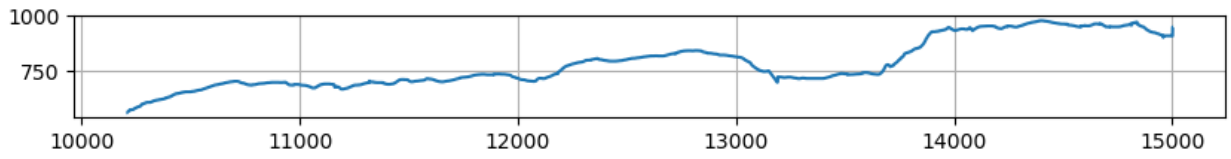
$\text{base}_{c}retaceous_{2}102_{f}eatures = gg.visualization.calculate_{p}osition_{o}n_{s}eismic(line, base_{c}retaceous_{2}102, 15008, 10203)base_{c}reta$

$\text{base}_{c}retaceous_{2}102_{f}eatures.to_{f}ile(file_{p}ath + ' Seismic_{H}orizons_{C}DPs_{L}ines_{B}ase_{C}retaceous.shp')$

[15]:
```
base_cretaceous_2102_features = gpd.read_file(file_path + 'Seismic_Horizons_CDPs_Lines_
↪Base_Cretaceous.shp')
base_cretaceous_2102_features
```

[15]:
```
   Name                                              geometry
0  2102  LINESTRING (10204.968 558.911, 10205.975 558.9...
```

[16]:
```
fig, ax = plt.subplots(1, figsize=(10,10))
base_cretaceous_2102_features.plot(ax=ax)
plt.grid()
```



[18]:
```
base_cretaceous_2102_features_linestrings, change_points = gg.visualization.split_
↪seismic_horizons(features_gdf=base_cretaceous_2102_features,

↪           threshold=20)
base_cretaceous_2102_features_linestrings, change_points
```

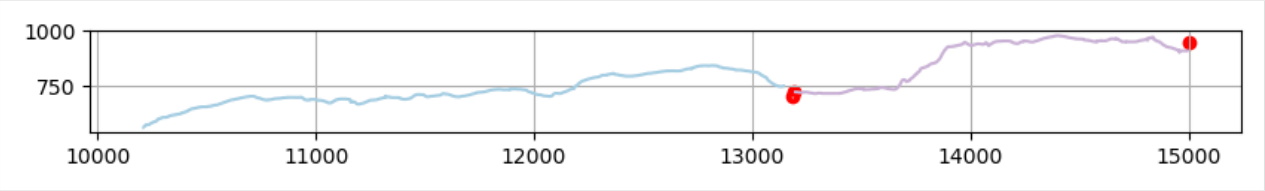
[18]:
```
(                                             geometry  id
 0  LINESTRING (10204.968 558.911, 10205.975 558.9...   0
 1  LINESTRING (13187.289 701.514, 13188.379 698.4...   1
 2  LINESTRING (13191.373 723.164, 13192.368 723.1...   2
 3  LINESTRING (15002.008 950.317, 15003.002 950.094)   3,
 [0, 2947, 2951, 4761, 4763])
```

[19]:
```
base_cretaceous_2102_features_points = gg.vector.extract_xy(base_cretaceous_2102_
↪features)
base_cretaceous_2102_features_points['distance'] = base_cretaceous_2102_features_points.
↪distance(base_cretaceous_2102_features_points.shift(), align=True)

base_cretaceous_2102_features_points
```

[19]:
```
      Name                   geometry        X      Y  distance
0     2102  POINT (10204.968 558.911) 10204.97 558.91       NaN
1     2102  POINT (10205.975 558.911) 10205.97 558.91      1.01
2     2102  POINT (10206.981 560.030) 10206.98 560.03      1.50
3     2102  POINT (10207.985 561.149) 10207.98 561.15      1.50
4     2102  POINT (10208.985 561.894) 10208.99 561.89      1.25
...    ...                        ...      ...    ...       ...
4759  2102  POINT (15000.002 909.671) 15000.00 909.67      0.99
4760  2102  POINT (15000.980 909.075) 15000.98 909.07      1.15
4761  2102  POINT (15002.008 950.317) 15002.01 950.32     41.26
4762  2102  POINT (15003.002 950.094) 15003.00 950.09      1.02
4763  2102  POINT (15004.007 949.935) 15004.01 949.93      1.02

[4764 rows x 5 columns]
```

[20]:
```
threshold = 20
```

[21]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

base_cretaceous_2102_features_linestrings.plot(ax=ax, column='id', cmap='Paired')
base_cretaceous_2102_features_points[base_cretaceous_2102_features_points['distance']>
↪=threshold].plot(ax=ax, color='red')
plt.grid()
```



[22]:
```
base_cretaceous_2102_features_linestrings.to_file(file_path + 'Seismic_Horizons_CDPs_
↪Lines_Base_Cretaceous_split.shp')
```

[23]:
```
faults = gpd.read_file(file_path + 'Faults_CDPs_Lines_split.shp')
faults
```

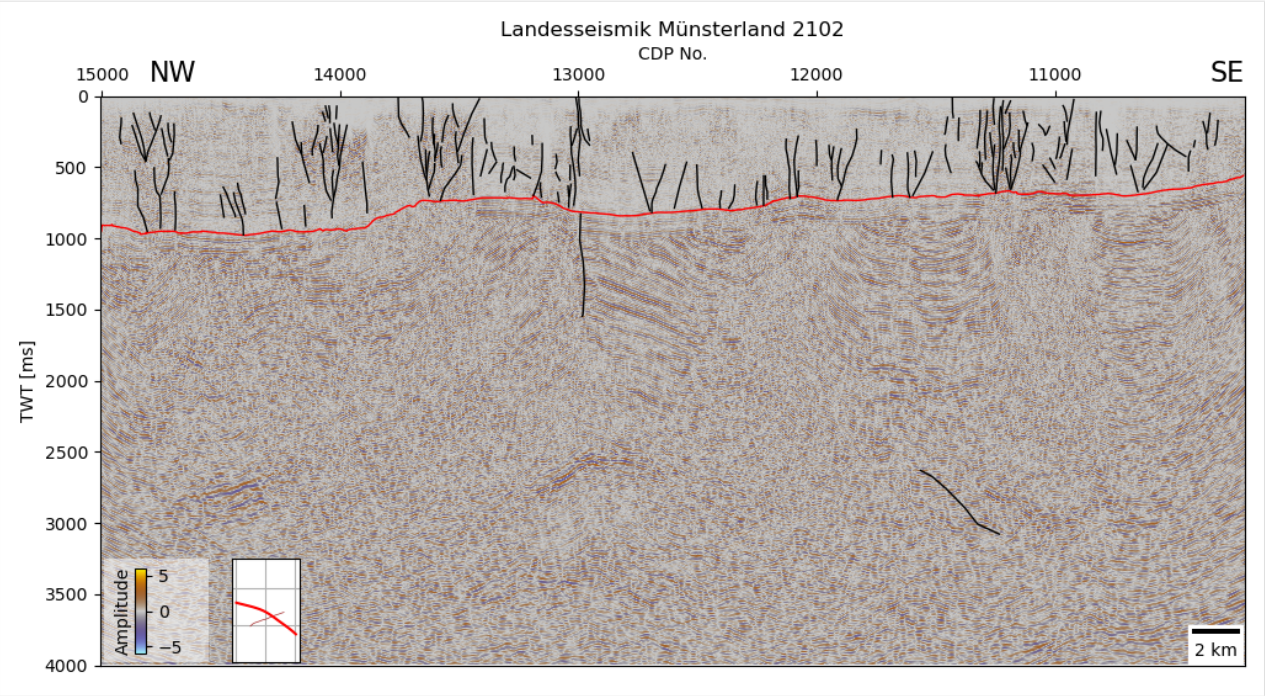
[23]:
```
                          Name
0    Fault interpretation 164  \
1    Fault interpretation 165
2    Fault interpretation 166
3    Fault interpretation 167
4    Fault interpretation 168
..                          ...
140   Fault interpretation 52
141   Fault interpretation 53
142   Fault interpretation 56
143   Fault interpretation 57
144   Fault interpretation 58

                                              geometry
0    LINESTRING (10410.600 309.512, 10417.264 362.367)
1    LINESTRING (10348.015 117.902, 10359.164 262.483)
2    LINESTRING (10825.354 111.007, 10827.121 201.4...
3    LINESTRING (10806.579 103.932, 10812.443 187.4...
4    LINESTRING (10919.610 20.529, 10929.574 120.41...
..                                                 ...
140  LINESTRING (10378.146 176.226, 10363.864 266.9...
141  LINESTRING (10438.399 532.834, 10438.399 430.5...
142  LINESTRING (10661.375 452.519, 10656.012 384.4...
143  LINESTRING (10783.751 231.979, 10763.524 400.9...
144  LINESTRING (10737.597 262.077, 10739.699 399.9...

[145 rows x 2 columns]
```

6.67.11 Plotting the seismic data and interpretations using matplotlib

[24]:
```
minx = landesseismik2102.to_dataframe().reset_index()['cdp'].min()
maxx = landesseismik2102.to_dataframe().reset_index()['cdp'].max()

minx, maxx
```

[24]:
```
(10203, 15008)
```

[25]:
```
from matplotlib_scalebar.scalebar import ScaleBar
fig, ax = plt.subplots(1, figsize=(12,6))
im = ax.imshow(np.fliplr(landesseismik2102_array.T),
           cmap=cmap_seismic,
           vmin=-6,
           vmax=6,
           extent=[maxx,
                   minx,
                   4000,
                   0])
ax.xaxis.set_ticks_position('top')
ax.xaxis.set_label_position('top')

axins = ax.inset_axes(bounds=[0.03, 0.02, 0.01, 0.15])

axseismic = ax.inset_axes(bounds=[0.06, 0.006, 0.17, 0.181])
seismic.plot(ax=axseismic, color='brown', linewidth=0.5)
seismic[1:2].plot(ax=axseismic, color='red', linewidth=1.5)

axseismic.set_ylim(5.725e6, 5.795e6)
axseismic.grid(visible=True, which='both',axis='both')
axseismic.set_yticklabels([])
axseismic.set_xticklabels([])
axseismic.xaxis.set_ticks_position('none')
axseismic.yaxis.set_ticks_position('none')

cbar = fig.colorbar(im, cax=axins)
cbar.set_label('Amplitude' ,labelpad=-40)

ax.set_ylabel('TWT [ms]')
ax.set_xlabel('CDP No.')
ax.set_title('Landesseismik Münsterland 2102')

ax.text(10350, -100, 'SE', fontsize=16)
ax.text(14800, -100, 'NW', fontsize=16)


pp1 = plt.Rectangle((14550, 3250), 440, 725, zorder=1, facecolor='white', alpha=0.5)
ax.add_patch(pp1)

scalebar = ScaleBar(0.01, "km", length_fraction=0.1, location='lower right')
ax.add_artist(scalebar)

base_cretaceous_2102_features.plot(ax=ax, color='red', linewidth=1, marker='-')
faults.plot(ax=ax, color='black', linewidth=1, marker='-')
plt.ylim(4000,0)
plt.gca().set_aspect('auto')
```



6.68 66 Generating Voronoi Polygons

This notebook illustrates the creation of Thiessen Polygons to illustrate the the data quantity of borehole data according to page 236 of the following report of the BGE: https://www.bge.de/fileadmin/user_upload/Standortsuche/Wesentliche_Unterlagen/Methodik/Phase_I_Schritt_2/rvSU-Methodik/20220328_Anlage_zu_rvSU_Konzept_Methodenbeschreibung_barrierefrei.pdf

6.68.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import gemgis as gg
import geopandas as gpd
import matplotlib.pyplot as plt
```

[2]:
```
file_path ='data/66_generating_voronoi_polygons/'
# gg.download_gemgis_data.download_tutorial_data(filename="66_generating_voronoi_
↪polygons.zip", dirpath=file_path)
```

6.68.2 Loading Borehole Data

The borehole data was extracted from the borehole database of the Geological Survey of NRW (https://www.gd.nrw.de/gd_archive_dabo.htm).

[3]:
```
data = gpd.read_file(file_path + 'Bohrungen_groesser_1000.shp')
data.head()
```

[3]:
```
     Index     DABO No.                   Name  Number   Depth           X  \
0  GD10092  DABO_102029     Klingerhuf1(1982)     NaN 1118.00 32332544.77  \
1  GD50035    DABO_1521           Salzbergen2     NaN 1875.70 32388961.53
2  GD50073    DABO_1564         B.Salzbergen3       3 1908.90 32390628.91
3  GD50261    DABO_2319  Ibbenbüren-Westfeld1  UB1121 1352.00 32409115.65
4  GD51096    DABO_7331           GildehausZ1     NaN 1683.10 32370464.85

           Y      Z       X_GK       Y_GK  ...     Kind
0 5701554.56  30.80 2541103.04 5700806.87  ...  Bohrung  \
1 5798219.66  30.00 2593500.00 5799735.00  ...  Bohrung
2 5799824.32  36.00 2595100.00 5801408.00  ...  Bohrung
3 5798184.48 118.20 3409152.00 5800063.00  ...  Bohrung
4 5790221.74  40.00 2575345.00 5790975.00  ...  Bohrung

                      Procedure
0        gemischtes Bohrverfahren  \
1        gemischtes Bohrverfahren
2  Spülbohrung (Rotary-Verfahren)
3        gemischtes Bohrverfahren
4        gemischtes Bohrverfahren

                                      Confidenti
0  offen; Bohrung mit dokumentiertem Freigabeverm...  \
1  vertraulich, offen nur mit Einwilligung; Eigen...
2  vertraulich, offen nur mit Einwilligung; Eigen...
3  offen; Bohrung mit dokumentiertem Freigabeverm...
4  vertraulich, offen nur mit Einwilligung; Eigen...

                                      Record Typ Lithlog Ve
0           Übertragung eines alten Archivbestandes          1  \
1  Aufnahme durch Sachbearbeiter; überarbeitet na...          1
2           Übertragung eines alten Archivbestandes          1
3           Übertragung eines alten Archivbestandes          1
4  Aufnahme durch Sachbearbeiter; überarbeitet na...          1

                                         Quality               Drilling P
0  Schichtdaten von guter Qualität; genaue strati...   30.9.1982 - 1.12.1982  \
1  Schichtdaten von guter Qualität; genaue strati...   15.2.1949 - 27.5.1950
2  Schichtdaten von guter Qualität; genaue strati...  19.10.1950 - 25.5.1951
3  Schichtdaten von guter Qualität; genaue strati...  11.11.1974 - 14.1.1975
4  Schichtdaten von guter Qualität; genaue strati...                  ..1956

                                    Remarks
0                              Karbon Cumulus  \
1  paläontologisch untersucht; Ausführliche Unter...
2               siehe Original-Schichtenverzeichnis
```

(continues on next page)

(continued from previous page)

```
3                                    Karbon Cumulus
4                Ausführliche Unterlagen im Original

                                Availabili                              geometry
0  Original-Schichtenverzeichnis liegt vor  POINT (32332544.770 5701554.560)
1  Original-Schichtenverzeichnis liegt vor  POINT (32388961.530 5798219.660)
2  Original-Schichtenverzeichnis liegt vor  POINT (32390628.910 5799824.320)
3  Original-Schichtenverzeichnis liegt vor  POINT (32409115.650 5798184.480)
4  Original-Schichtenverzeichnis liegt vor  POINT (32370464.850 5790221.740)

[5 rows x 26 columns]
```

6.68.3 Loading the Outline of NRW

[4]:
```
outline = gpd.read_file(file_path + 'outline.shp')
outline
```

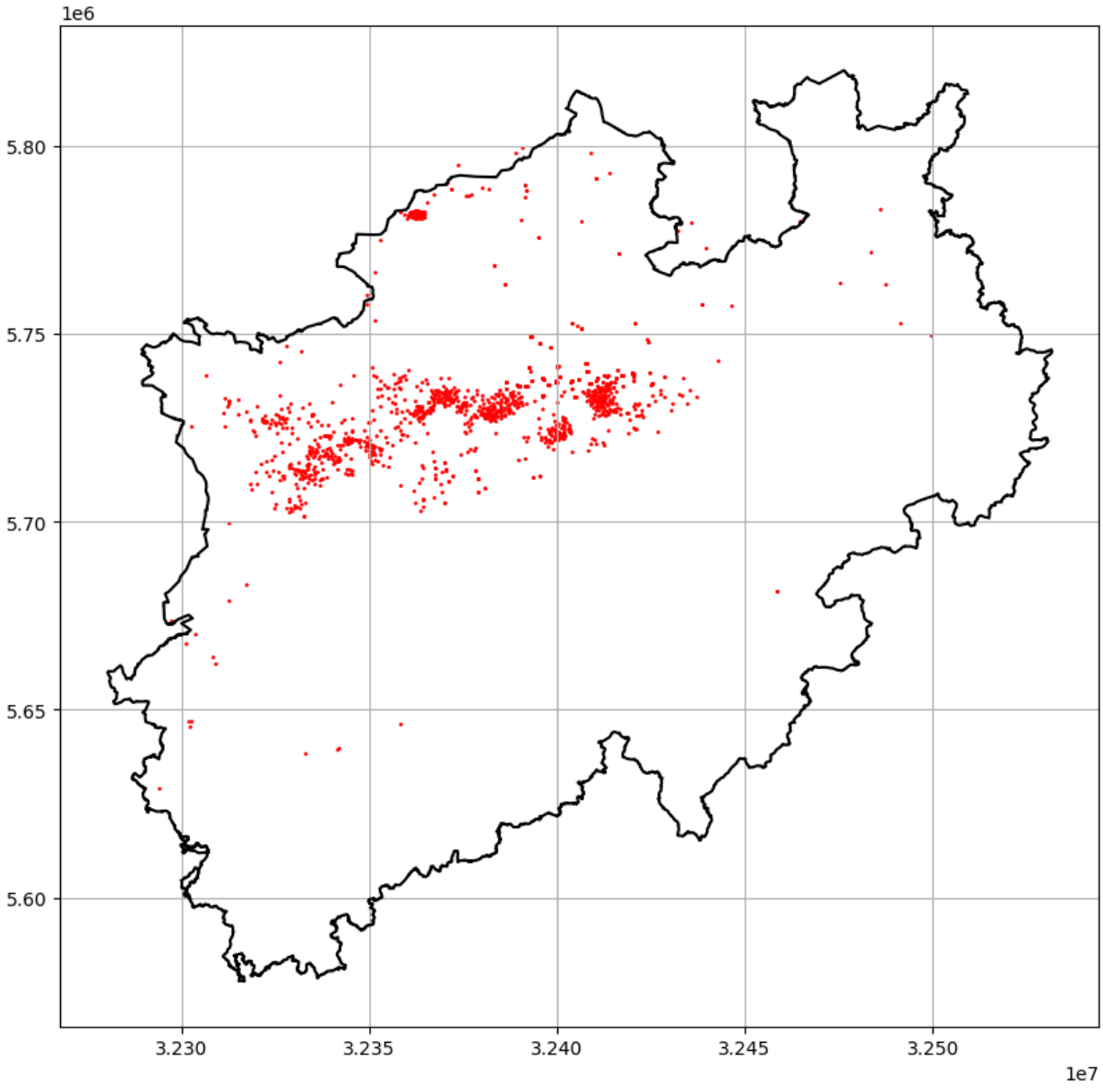
[4]:
```
    osm_id     class            type                              name
0 62761.00  boundary  administrative  North Rhine-Westphalia, Germany  \

                                          address
0  "state"=>"North Rhine-Westphalia","ISO3166-2-l...  \

                                         extratags
0  "ref:nuts"=>"DEA","wikidata"=>"Q1198","wikiped...  \

                                          geometry
0  MULTIPOLYGON (((300636.227 5600913.724, 300639...
```

[5]:
```
outline.crs
```

[5]:
```
<Projected CRS: EPSG:25832>
Name: ETRS89 / UTM zone 32N
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Europe between 6°E and 12°E: Austria; Belgium; Denmark - onshore and offshore;␣
↪Germany - onshore and offshore; Norway including - onshore and offshore; Spain -␣
↪offshore.
- bounds: (6.0, 38.76, 12.01, 84.33)
Coordinate Operation:
- name: UTM zone 32N
- method: Transverse Mercator
Datum: European Terrestrial Reference System 1989 ensemble
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
```

6.68.4 Plotting the Borehole Data

[6]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

outline.to_crs('EPSG:4647').boundary.plot(ax=ax, color='black')
data.plot(ax=ax, markersize=1, color='red')

plt.grid()
```



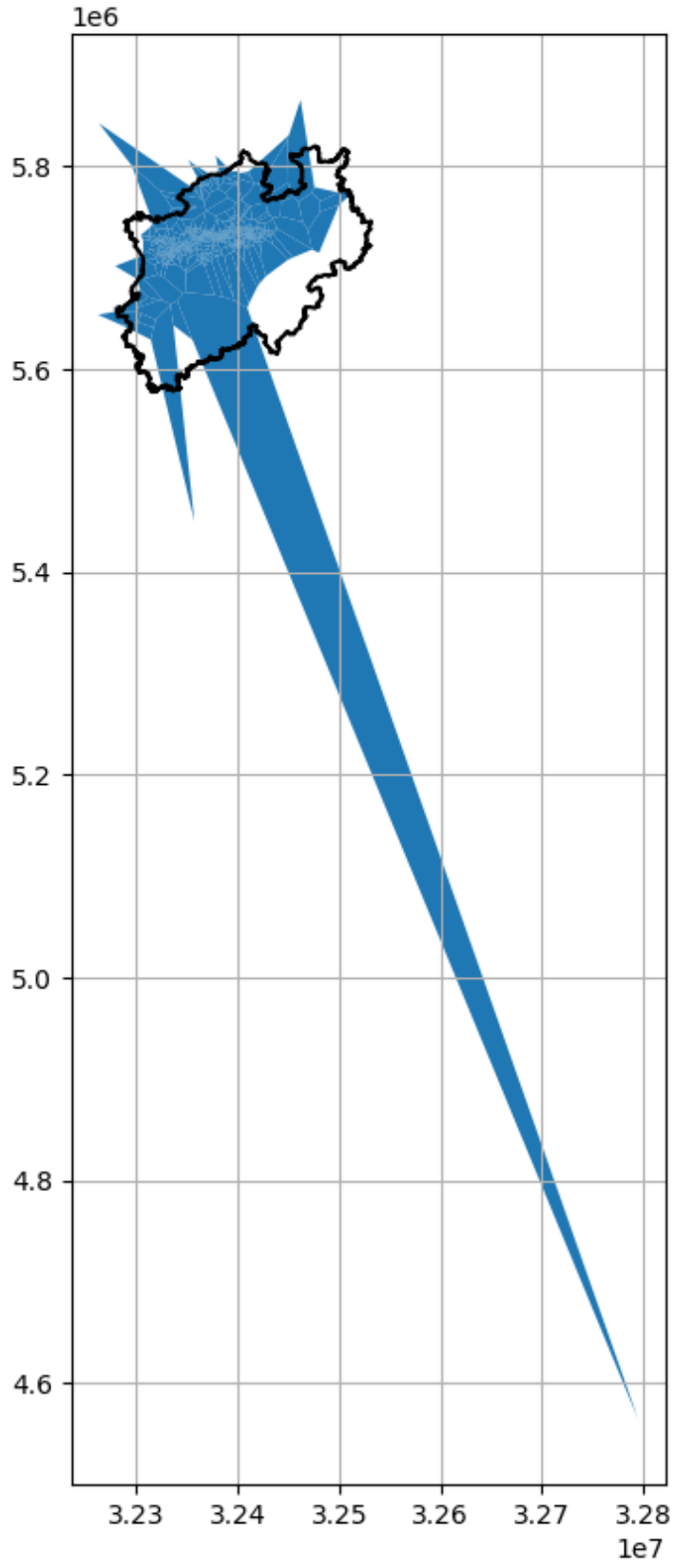
6.68.5 Voronoi Tessalation

[8]:
```
gdf = gg.vector.create_voronoi_polygons(data)
gdf
```

[8]:
```
                                              geometry         area
0     POLYGON ((32473194.081 5718253.741, 32469287.8... 665698318.29
1     POLYGON ((32473175.498 5773862.463, 32481354.2... 184545066.31
2     POLYGON ((32481354.247 5765667.587, 32481234.9... 304148453.73
3     POLYGON ((32459819.403 5765274.413, 32473175.4... 490884327.21
4     POLYGON ((32317260.014 5640927.969, 32314568.7... 250801576.02
...                                                 ...          ...
1131  POLYGON ((32326629.143 5727191.374, 32326669.3...    922649.29
1132  POLYGON ((32326895.182 5726701.188, 32327147.6...   1407420.98
1133  POLYGON ((32327531.454 5727716.570, 32327549.9...    349756.27
1134  POLYGON ((32327677.535 5727156.396, 32328944.7...   1725230.07
1135  POLYGON ((32327531.454 5727716.570, 32327549.9...   3340904.68

[1135 rows x 2 columns]
```

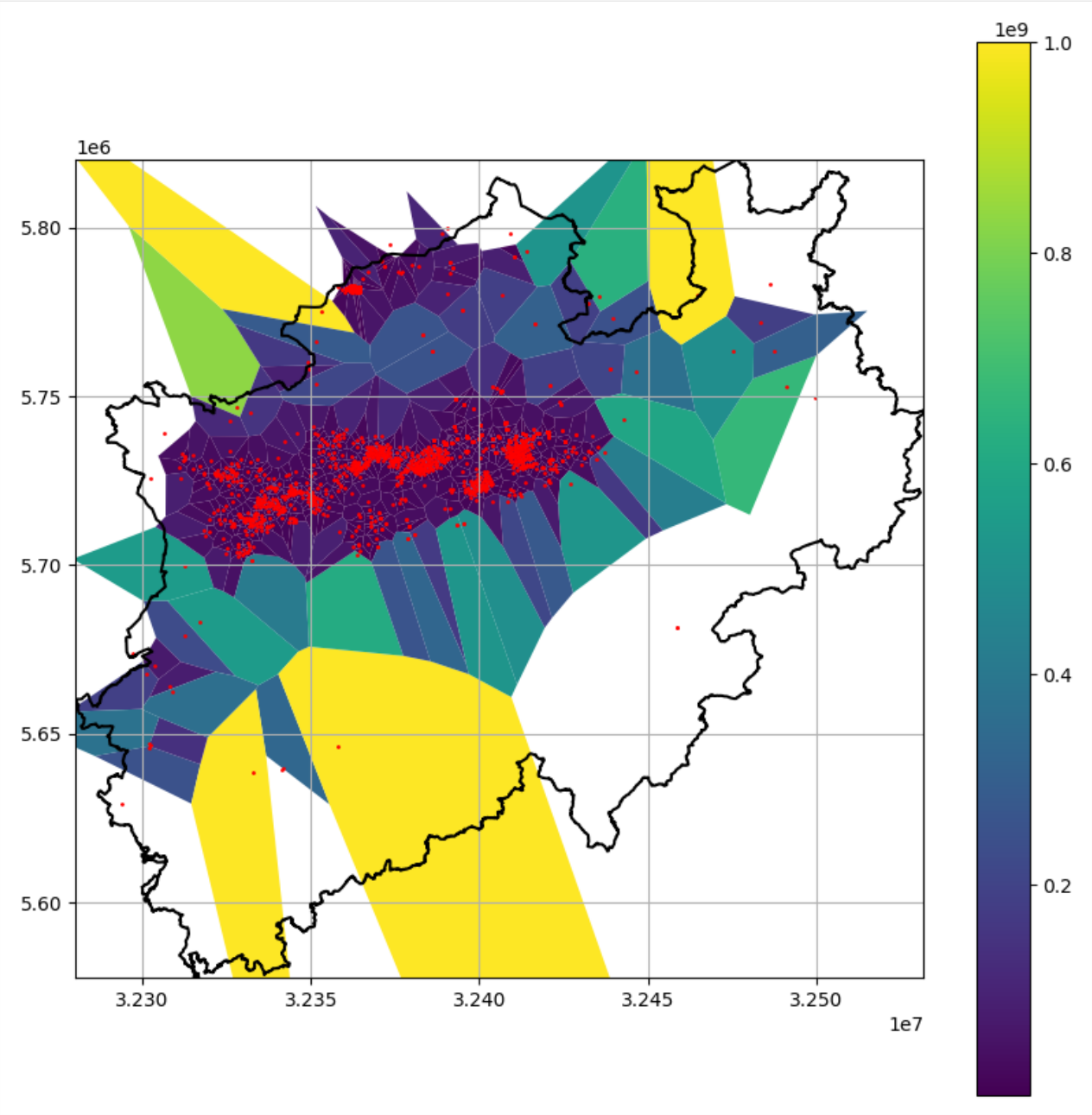
6.68.6 Plotting Valid Voronoi Polygons

[9]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

gdf.plot(ax=ax)
outline.to_crs('EPSG:4647').boundary.plot(ax=ax, color='black')

plt.grid()
plt.show()
```


1e6
5.8
5.6
5.4
5.2
5.0
4.8
4.6
3.23
3.24
3.25
3.26
3.27
3.28
1e7

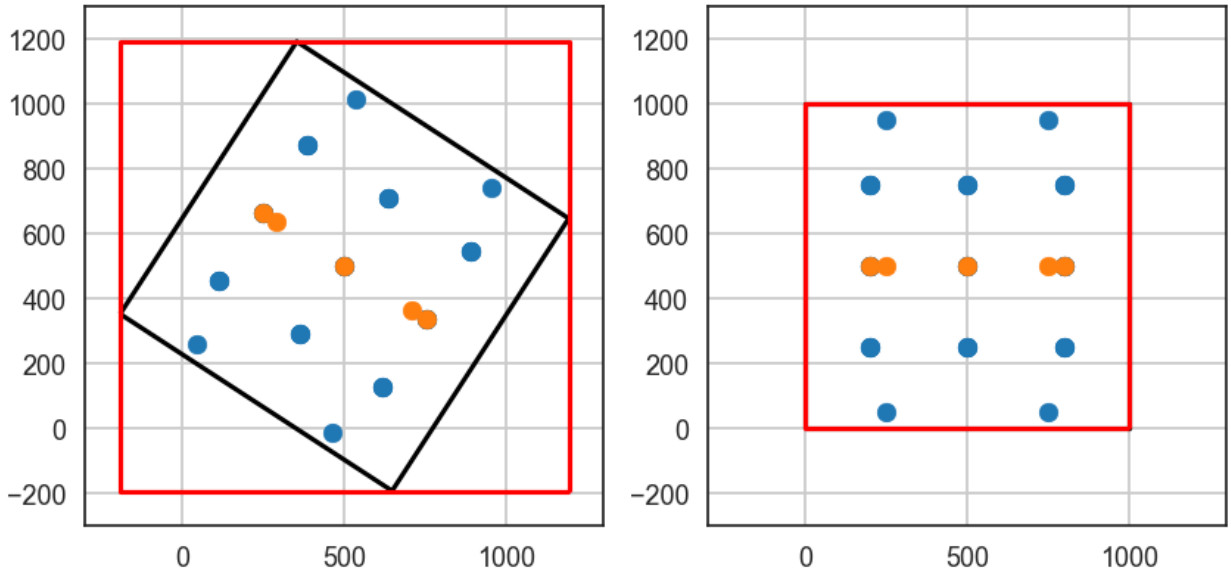
6.68.7 Plotting GeoDataFrame

[10]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

outline.to_crs('EPSG:4647').boundary.plot(ax=ax, color='black')#
gdf.plot(ax=ax, column='area', vmax=1e9, legend=True)
data.plot(ax=ax, markersize=1, color='red')

plt.xlim(outline.to_crs('EPSG:4647').total_bounds[0], outline.to_crs('EPSG:4647').total_
↪bounds[2])
plt.ylim(outline.to_crs('EPSG:4647').total_bounds[1], outline.to_crs('EPSG:4647').total_
↪bounds[3])
plt.grid()
plt.show()
```



6.69 67 Rotating GemPy Input Data

This notebook illustrates how to rotate GemPy Input data. Input data may be rotated because subsurface structures do not trend West-East or North-South but rather oblique. As GemPy only takes extents that are oriented N-S and W-E, there would be a large area that would be discarded after the modeling process and does not need to be modeled in the first place. Rotating the data will therefore reduce the area that needs to be calculated and therefore computing times which may allow for a higher resolution or the integration of more input data. The rotation implemented in GemGIS is performed around the center point of the extent of the model. This does not only allow for rotating the data at its place (instead of rotating it around (0,0) which may cause a translation of the model to extreme coordinates, but also for rotating the resulting meshes back to their original orientation.



6.69.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import gempy as gp
import gemgis as gg
import pandas as pd
from shapely.geometry import Polygon
import matplotlib.pyplot as plt

import pyvista as pv
pv.set_jupyter_backend('client')
```

```
No module named 'osgeo'
```

```
WARNING (aesara.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (aesara.configdefaults): g++ not detected!  Aesara will be unable to compile C-
↪implementations and will default to Python. Performance may be severely degraded. To␣
↪remove this warning, set Aesara flags cxx to an empty string.
WARNING (aesara.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
file_path ='data/67_rotating_gempy_input_data/'
gg.download_gemgis_data.download_tutorial_data(filename="67_rotation_gempy_input_data.zip
↪", dirpath=file_path)
```

6.69.2 Loading Input Data

[3]:
```
interfaces = pd.read_csv(file_path + 'interfaces.csv', delimiter=';')
interfaces.head()
```

[3]:
```
     X    Y    Z formation
0  200  250 -100    Layer1
1  200  500 -100    Layer1
2  200  750 -100    Layer1
3  200  250 -200    Layer2
4  200  500 -200    Layer2
```

[4]:
```
orientations = pd.read_csv(file_path + 'orientations.csv', delimiter=';')
orientations.head()
```

[4]:
```
     X    Y    Z formation  dip  azimuth  polarity
0  200  500 -100    Layer1    0        0         1
1  800  500 -100    Layer1    0        0         1
2  500  500 -300    Layer1    0        0         1
3  250  500 -100    Fault1   60       90         1
4  750  500 -100    Fault2   60      270         1
```

6.69.3 Creating Polygon Extent

The original extent of the model will be translated into a Shapely Polygon.

[5]:
```
extent = Polygon([(0,0), (0,1000), (1000, 1000), (1000,0)])
extent
```

[5]:

6.69.4 Performing initial rotation for Tutorial

The input data will be rotated a first time to create a non-N-S-W-E oriented dataset.

[6]:
```
extent_rotated, interfaces_rotated, orientations_rotated, extent_gdf = gg.utils.rotate_
↪gempy_input_data(extent=extent,
↪                 interfaces=interfaces,
↪                 orientations=orientations,
↪                 zmin=-600,
↪                 zmax=0,
↪                 return_extent_gdf=True,
↪                 manual_rotation_angle=33)
extent_rotated, interfaces_rotated.head(), orientations_rotated.head(), extent_gdf
```

[6]:
```
([-191.65480148022556,
  1191.6548014802256,
```

(continues on next page)

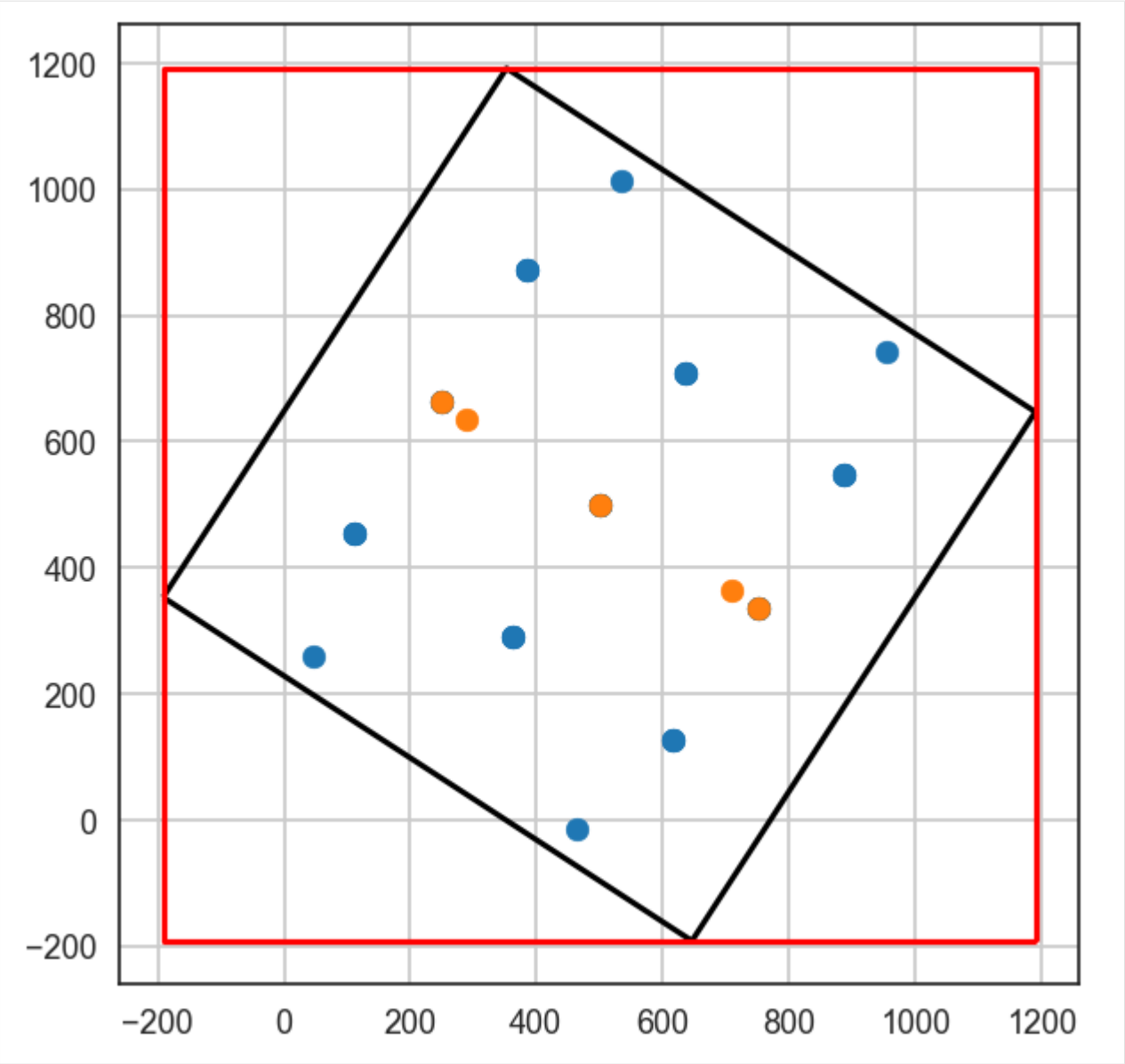
(continued from previous page)

```
 -191.65480148022556,
 1191.6548014802256,
 -600,
 0],
       X      Y       Z formation                 geometry
0 112.24 453.72 -100.00    Layer1  POINT (112.239 453.724)
1 248.40 663.39 -100.00    Layer1  POINT (248.399 663.392)
2 384.56 873.06 -100.00    Layer1  POINT (384.559 873.059)
3 112.24 453.72 -200.00    Layer2  POINT (112.239 453.724)
4 248.40 663.39 -200.00    Layer2  POINT (248.399 663.392),
       X      Y       Z formation   dip  azimuth  polarity  \
0 248.40 663.39 -100.00    Layer1  0.00     0.00      1.00  \
1 751.60 336.61 -100.00    Layer1  0.00     0.00      1.00
2 500.00 500.00 -300.00    Layer1  0.00     0.00      1.00
3 290.33 636.16 -100.00    Fault1 60.00    90.00      1.00
4 709.67 363.84 -100.00    Fault2 60.00   270.00      1.00

                  geometry
0  POINT (248.399 663.392)
1  POINT (751.601 336.608)
2  POINT (500.000 500.000)
3  POINT (290.332 636.160)
4  POINT (709.668 363.840)  ,
                                             geometry
0  POLYGON ((-191.655 352.984, 352.984 1191.655, ...)
```

6.69.5 Plotting the non-N-S-W-E Model Data

Plotting the input data shows that the graben structure is striking NW-SE. This is not ideal for modeling in GemPy. In order to model the entire black extent that is desired, the red extent would have to be modeled which results in a much larger model than necessary.

[7]:
```
fig, ax = plt.subplots(1, figsize=(7,7))

extent_gdf.boundary.plot(ax=ax, color='black')
interfaces_rotated.plot(ax=ax)
orientations_rotated.plot(ax=ax)
gpd.GeoDataFrame(geometry=[box(*extent_gdf.total_bounds)]).boundary.plot(ax=ax, color=
↪'red')
plt.grid()
```



6.69.6 Rotating the Model back to a N-S-W-E-oriented model

The non-N-S-W-E-oriented model can be rotated to a N-S-W-E-oriented model using the GemGIS function `gg.utils.rotate_gempy_input_data(...)`. The rotation angle is either calculated automatically or can be set manually.

[8]:
```
extent_straight, interfaces_straight, orientations_straight, extent_gdf_straight = gg.
↪utils.rotate_gempy_input_data(extent=extent_gdf,
                               interfaces=interfaces_rotated,
                               orientations=orientations_rotated,
                               zmin=-600,
```

(continues on next page)

(continued from previous page)

```
                                                                                ␣
↪                           zmax=0,
                                                                                ␣
↪                           rotate_reverse_direction=True,
                                                                                ␣
↪                           return_extent_gdf=True)
extent_straight, interfaces_straight.head(), orientations_straight.head(), extent_gdf_
↪straight
```

[8]:
```
([-1.1368683772161603e-13, 1000.0, -2.842170943040401e-14, 1000.0, -600, 0],
       X      Y       Z formation                 geometry
 0 200.00 250.00 -100.00    Layer1  POINT (200.000 250.000)
 1 200.00 500.00 -100.00    Layer1  POINT (200.000 500.000)
 2 200.00 750.00 -100.00    Layer1  POINT (200.000 750.000)
 3 200.00 250.00 -200.00    Layer2  POINT (200.000 250.000)
 4 200.00 500.00 -200.00    Layer2  POINT (200.000 500.000),
       X      Y       Z formation   dip  azimuth  polarity
 0 200.00 500.00 -100.00    Layer1  0.00     0.00      1.00  \
 1 800.00 500.00 -100.00    Layer1  0.00     0.00      1.00
 2 500.00 500.00 -300.00    Layer1  0.00     0.00      1.00
 3 250.00 500.00 -100.00    Fault1 60.00    90.00      1.00
 4 750.00 500.00 -100.00    Fault2 60.00   270.00      1.00

                   geometry
 0  POINT (200.000 500.000)
 1  POINT (800.000 500.000)
 2  POINT (500.000 500.000)
 3  POINT (250.000 500.000)
 4  POINT (750.000 500.000)  ,
                                              geometry
 0  POLYGON ((-0.000 -0.000, -0.000 1000.000, 1000...)
```

6.69.7 Plotting the N-S-W-E Model Data

The rotated data can be plotted as before. We can see that the data was rotated around the center of the extent and that the size of the red frame is now equal to the size of the black frame as desired. Now, the modeling can be performed.

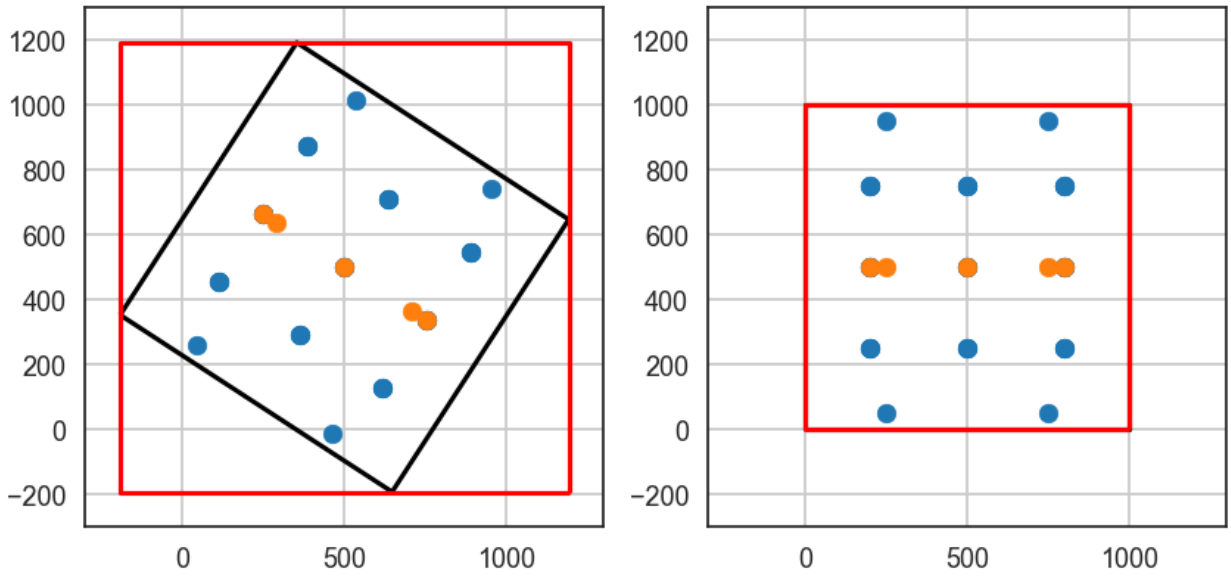
[9]:
```
fig, (ax1, ax) = plt.subplots(1,2, figsize=(10,7))

extent_gdf_straight.boundary.plot(ax=ax, color='black')
interfaces_straight.plot(ax=ax)
orientations_straight.plot(ax=ax)
gpd.GeoDataFrame(geometry=[box(*extent_gdf_straight.total_bounds)]).boundary.plot(ax=ax,␣
↪color='red')
ax.grid()
ax.set_ylim(-300,1300)
ax.set_xlim(-300,1300)


extent_gdf.boundary.plot(ax=ax1, color='black')
interfaces_rotated.plot(ax=ax1)
```

(continues on next page)

(continued from previous page)

```
orientations_rotated.plot(ax=ax1)
gpd.GeoDataFrame(geometry=[box(*extent_gdf.total_bounds)]).boundary.plot(ax=ax1, color=
↪'red')
ax1.grid()
ax1.set_ylim(-300,1300)
ax1.set_xlim(-300,1300)
```

[9]:
```
(-300.0, 1300.0)
```



6.69.8 Creating the GemPy Model

[10]:
```
geo_model = gp.create_model('Graben_Model')
geo_model
```

[10]:
```
Graben_Model  2023-08-01 14:29
```

[11]:
```
gp.init_data(geo_model, [0, 1000, 0, 1000, -600, 0], [50,50,50],
             surface_points_df=interfaces_straight,
             orientations_df=orientations_straight,
             default_values=True)
```

```
Active grids: ['regular']
```

[11]:
```
Graben_Model  2023-08-01 14:29
```

[12]:
```
geo_model.surfaces
```

[12]:

	surface	series	order_surfaces	color	id
0	Layer1	Default series	1	#015482	1
1	Layer2	Default series	2	#9f0052	2
2	Layer3	Default series	3	#ffbe00	3
3	Fault1	Default series	4	#728f02	4
4	Fault2	Default series	5	#443988	5

[13]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('Fault1'),
                          'Fault2': ('Fault2'),
                          'Strata1': ('Layer1', 'Layer2', 'Layer3'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_is_fault(['Fault1', 'Fault2'])
geo_model.surfaces
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[13]:

	surface	series	order_surfaces	color	id
3	Fault1	Fault1	1	#527682	1
4	Fault2	Fault2	1	#527682	2
0	Layer1	Strata1	1	#ffbe00	3
1	Layer2	Strata1	2	#728f02	4
2	Layer3	Strata1	3	#443988	5
5	Basement	Strata1	4	#ff3f20	6

[14]:
```
geo_model.set_topography(source='random')
```

```
[-120.    0.]
Active grids: ['regular' 'topography']
```

[14]:
```
Grid Object. Values:
array([[  10.        ,   10.        , -594.        ],
       [  10.        ,   10.        , -582.        ],
       [  10.        ,   10.        , -570.        ],
       ...,
       [1000.        ,  959.18367347,  -48.98763452],
       [1000.        ,  979.59183673,  -56.49305639],
       [1000.        , 1000.        ,  -61.62003486]])
```

[15]:
```
geo_model.surfaces
```

[15]:

	surface	series	order_surfaces	color	id
3	Fault1	Fault1	1	#527682	1
4	Fault2	Fault2	1	#527682	2
0	Layer1	Strata1	1	#ffbe00	3
1	Layer2	Strata1	2	#728f02	4
2	Layer3	Strata1	3	#443988	5
5	Basement	Strata1	4	#ff3f20	6

[16]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```

```
Widget(value="<iframe src='http://localhost:57216/index.html?ui=P_0x1b00e903100_0&
↪reconnect=auto' style='width...
```

[16]:
```
<gempy.plot.vista.GemPyToVista at 0x1b0112cb5e0>
```

[17]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
```

(continues on next page)

(continued from previous page)

```
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling aesara function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  2
Compilation Done!
Kriging values:
                     values
range              1536.23
$C_o$             56190.48
drift equations  [3, 3, 3]
```

[17]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1b0112cbe50>
```

[18]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

[19]:
```
gp.plot_3d(geo_model, notebook=True)
```

```
Widget(value="<iframe src='http://localhost:57216/index.html?ui=P_0x1b024d4c880_1&
↪reconnect=auto' style='width...
```

[19]:
```
<gempy.plot.vista.GemPyToVista at 0x1b024d4c130>
```

6.69.9 Extracting Depth Maps from GemPy Model

We now extract the depth maps from the GemPy Model which will then be rotated using Py

[20]:
```
meshes = gg.visualization.create_depth_maps_from_gempy(geo_model, surfaces=['Layer2',
↪'Fault1', 'Fault2'])
meshes
```

```
{'Layer2': [PolyData (0x1b024dab9a0)
    N Cells:    7938
    N Points:   4100
    N Strips:   0
    X Bounds:   1.000e+01, 9.900e+02
    Y Bounds:   1.000e+01, 9.900e+02
    Z Bounds:   -4.000e+02, -2.000e+02
    N Arrays:   1,
  '#728f02'],
 'Fault1': [PolyData (0x1b024da8160)
    N Cells:    5820
    N Points:   3038
    N Strips:   0
    X Bounds:   2.096e+02, 5.352e+02
    Y Bounds:   1.000e+01, 9.900e+02
    Z Bounds:   -5.940e+02, -3.000e+01
    N Arrays:   1,
```

(continues on next page)

(continued from previous page)

```
  '#527682'],
 'Fault2': [PolyData (0x1b024da9240)
   N Cells:    5720
   N Points:   2990
   N Strips:   0
   X Bounds:   4.648e+02, 7.904e+02
   Y Bounds:   1.000e+01, 9.900e+02
   Z Bounds:   -5.940e+02, -3.000e+01
   N Arrays:   1,
  '#527682']}
```

6.69.10 Rotating Meshes

The meshes will be rotated around the center of the extent to reach their original position.

[22]:
```
x,y = list(extent_gdf.iloc[0]['geometry'].centroid.coords)[0][0], list(extent_gdf.
↪iloc[0]['geometry'].centroid.coords)[0][1]
x,y
```

[22]:
```
(500.00000000000006, 500.00000000000006)
```

[23]:
```
mesh_layer2 = meshes['Layer2'][0].rotate_z(angle=33, point=(x,y,0))
mesh_fault1 = meshes['Fault1'][0].rotate_z(angle=33, point=(x,y,0))
mesh_fault2 = meshes['Fault2'][0].rotate_z(angle=33, point=(x,y,0))
```

6.69.11 Plotting the rotates Meshes

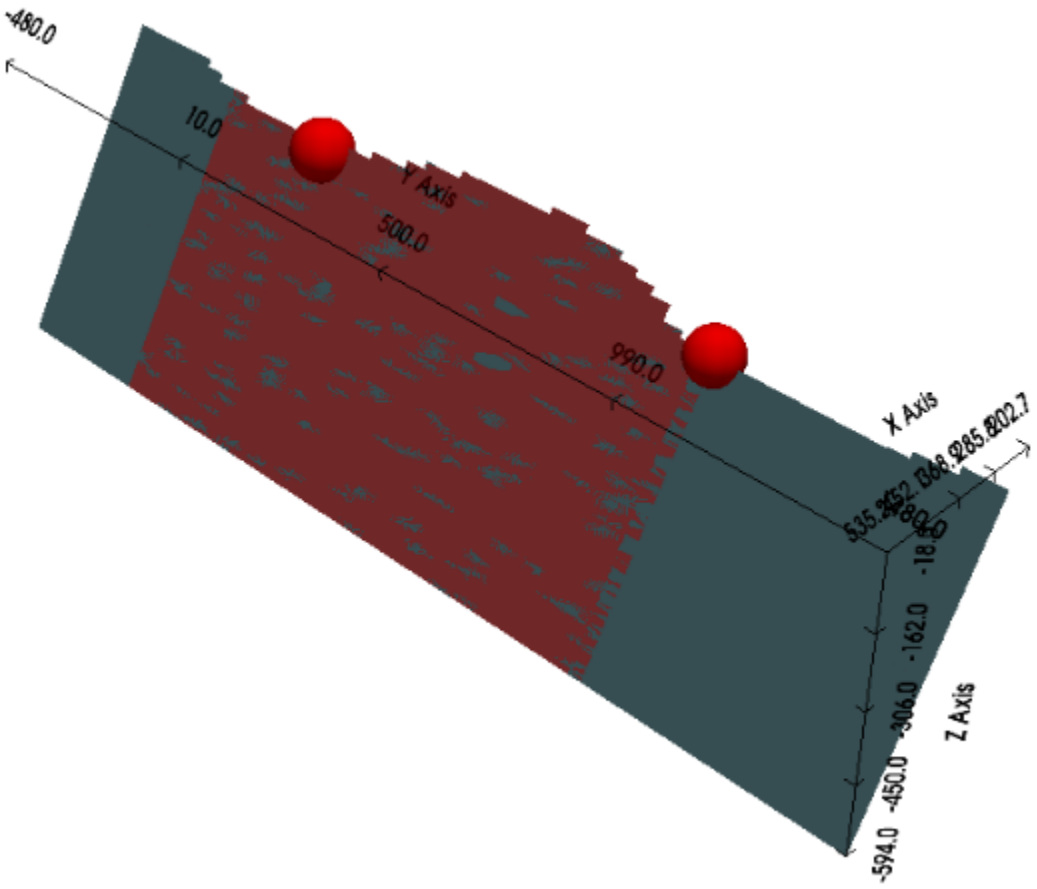
Rotating the meshes will result in the original position of the layers. The meshes/surfaces can now be used for further calculations and computations or exported to a GIS System.

[24]:
```
p = pv.Plotter(notebook=True)

p.add_mesh(mesh_layer2, color= meshes['Layer2'][1])

p.add_mesh(mesh_fault1, color= meshes['Fault1'][1])

p.add_mesh(mesh_fault2, color= meshes['Fault2'][1])

p.set_background('white')
p.show_grid(color='black')
p.show()
```

```
Widget(value="<iframe src='http://localhost:57216/index.html?ui=P_0x1b02de3ac20_2&
↪reconnect=auto' style='width...
```

6.70 68 Creating Finite Faults with GemGIS

This notebook illustrates how to create finite faults from a GemPy Model. Here, we will use an elongated version of the Graben Model to later create the finite faults.



6.70.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

import gempy as gp
import gemgis as gg
import pandas as pd
import geopandas as gpd
```

(continues on next page)

(continued from previous page)

```
import pyvista as pv
```

```
WARNING (aesara.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (aesara.configdefaults): g++ not detected!  Aesara will be unable to compile C-
↪implementations and will default to Python. Performance may be severely degraded. To␣
↪remove this warning, set Aesara flags cxx to an empty string.
WARNING (aesara.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
file_path ='data/68_Creating_Finite_Faults_with_GemGIS/'
# gg.download_gemgis_data.download_tutorial_data(filename="68_creating_finite_faults_
↪with_gemgis.zip", dirpath=file_path)
```

6.70.2 Loading Input Data

[3]:
```
interfaces = pd.read_csv(file_path + 'interfaces.csv', delimiter=';')
interfaces.head()
```

[3]:

	X	Y	Z	formation
0	200	250	-100	Layer1
1	200	500	-100	Layer1
2	200	750	-100	Layer1
3	200	250	-200	Layer2
4	200	500	-200	Layer2

[4]:
```
orientations = pd.read_csv(file_path + 'orientations.csv', delimiter=';')
orientations.head()
```

[4]:

	X	Y	Z	formation	dip	azimuth	polarity
0	200	500	-100	Layer1	0	0	1
1	800	500	-100	Layer1	0	0	1
2	500	500	-300	Layer1	0	0	1
3	250	500	-100	Fault1	60	90	1
4	750	500	-100	Fault2	60	270	1

6.70.3 Creating the GemPy Model

[5]:
```
geo_model = gp.create_model('Graben_Model')
geo_model
```

[5]:
```
Graben_Model  2023-10-10 16:03
```

[6]:
```
gp.init_data(geo_model, [0, 1000, -500, 1500, -600, 0], [50,50,50],
             surface_points_df=interfaces,
             orientations_df=orientations,
             default_values=True)

gp.map_stack_to_surfaces(geo_model,
                         {
```

(continues on next page)

(continued from previous page)

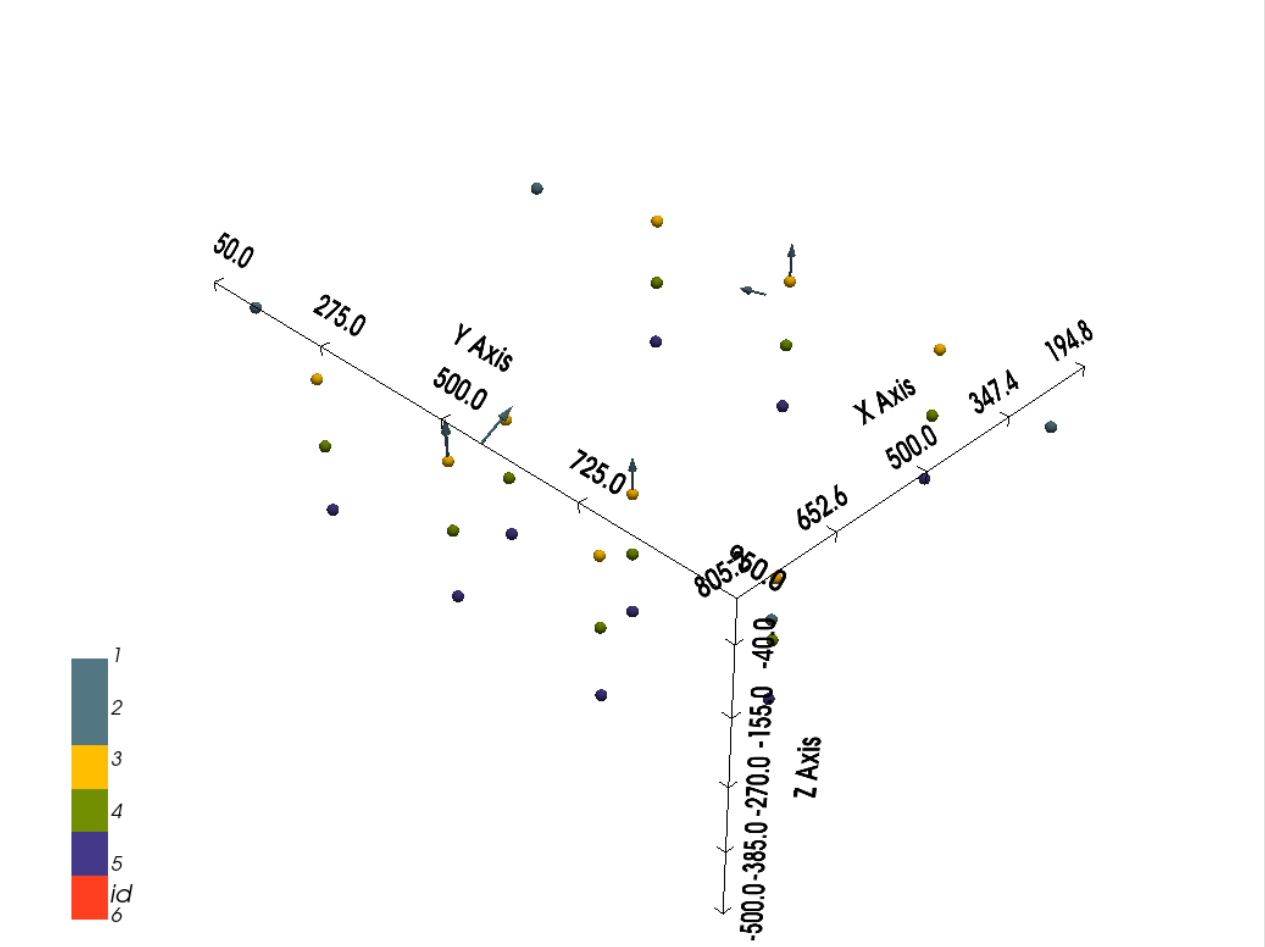
```
                          'Fault1': ('Fault1'),
                          'Fault2': ('Fault2'),
                          'Strata1': ('Layer1', 'Layer2', 'Layer3'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_is_fault(['Fault1', 'Fault2'])
geo_model.set_topography(source='random')
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
geo_model.surfaces
```

```
Active grids: ['regular']
Fault colors changed. If you do not like this behavior, set change_color to False.
Fault colors changed. If you do not like this behavior, set change_color to False.
[-120.    0.]
Active grids: ['regular' 'topography']
Compiling aesara function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  2
Compilation Done!
Kriging values:
                     values
range              2315.17
$C_o$            127619.05
drift equations  [3, 3, 3]
```

[6]:

	surface	series	order_surfaces	color	id
3	Fault1	Fault1	1	#527682	1
4	Fault2	Fault2	1	#527682	2
0	Layer1	Strata1	1	#ffbe00	3
1	Layer2	Strata1	2	#728f02	4
2	Layer3	Strata1	3	#443988	5
5	Basement	Strata1	4	#ff3f20	6

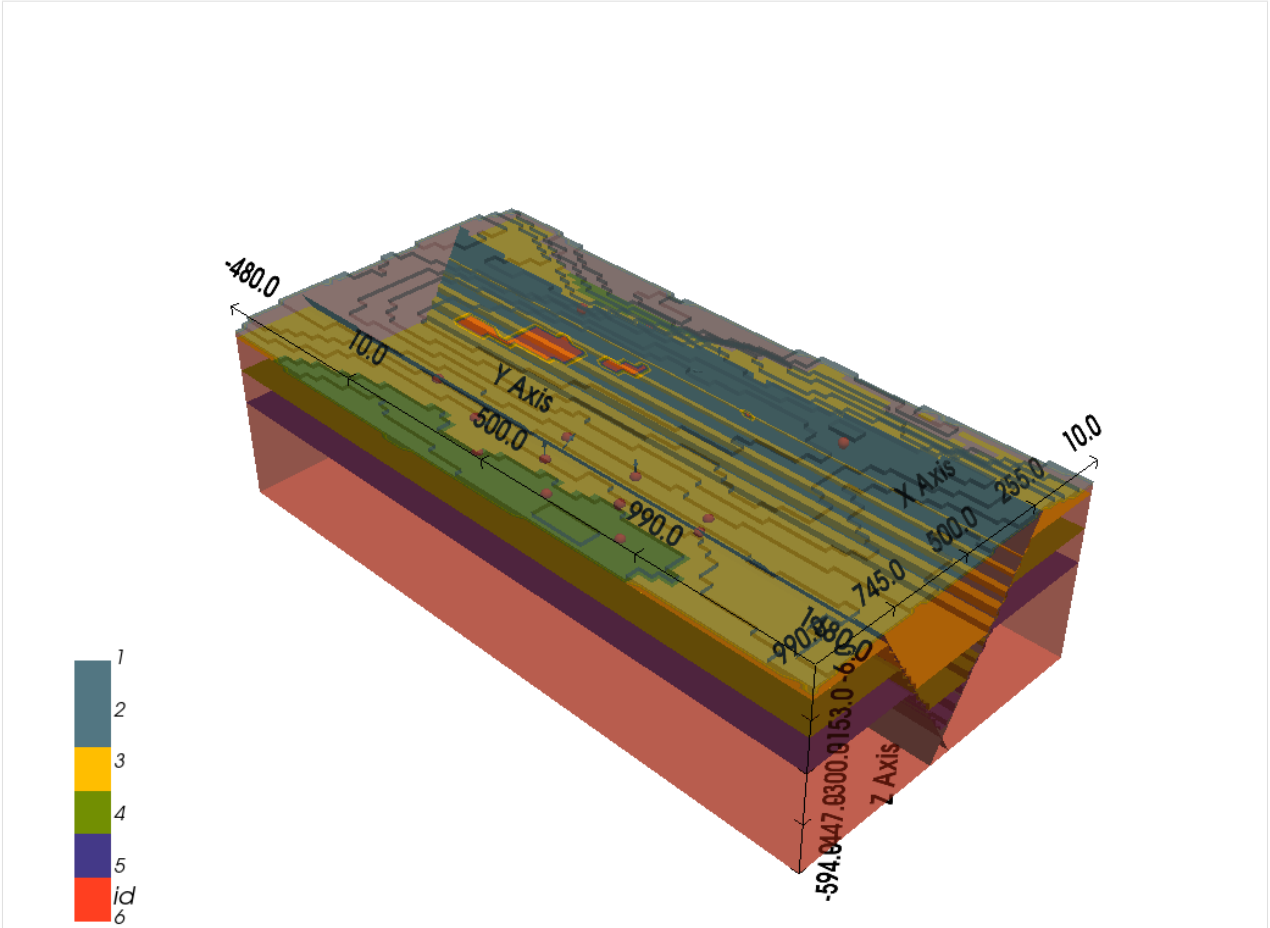
[7]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[7]:
```
<gempy.plot.vista.GemPyToVista at 0x25557e1ba90>
```

[8]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

[9]:
```
gp.plot_3d(geo_model, notebook=True)
```



[9]:
```
<gempy.plot.vista.GemPyToVista at 0x2555b0d0340>
```

6.70.4 Creating Finite Faults from GemPy Model

Finite faults will be created during a postprocessing step performed in GemGIS. Here, we are using the function `clip_fault_of_gempy_model(..)` to clip a fault. By default, the fault will be clipped at the first or last interface point (both sides can be chosen). In addition, a buffer along the strike of the fault can be chosen to allow the fault to extend beyond the last interface point if wished.

Here, we are choosing to clip the fault on both ends. For the one end, we would like to have a 250 m buffer so that the fault extends beyond the first point. For the last point, we do not want to have a buffer.

[10]:
```
mesh = gg.postprocessing.clip_fault_of_gempy_model(geo_model,fault='Fault1',
                                                   which='both',
                                                   buffer_first=250,
                                                   buffer_last=0)
mesh
```

[10]:
```
{'Fault1': [PolyData (0x2555e8d5300)
    N Cells:    3512
    N Points:   1881
    N Strips:   0
    X Bounds:   2.165e+02, 5.352e+02
```

(continues on next page)

(continued from previous page)

```
    Y Bounds:   -2.000e+02, 9.500e+02
    Z Bounds:   -5.940e+02, -4.200e+01
    N Arrays:   1,
  '#527682']}
```

[11]:
```
fault_polydata = pv.PolyData(interfaces[interfaces['formation']=='Fault1'][['X', 'Y', 'Z
↪']].values)
fault_polydata
```

[11]:
```
PolyData (0x2556e6f3220)
  N Cells:    2
  N Points:   2
  N Strips:   0
  X Bounds:   2.500e+02, 2.500e+02
  Y Bounds:   5.000e+01, 9.500e+02
  Z Bounds:   -1.000e+02, -1.000e+02
  N Arrays:   0
```

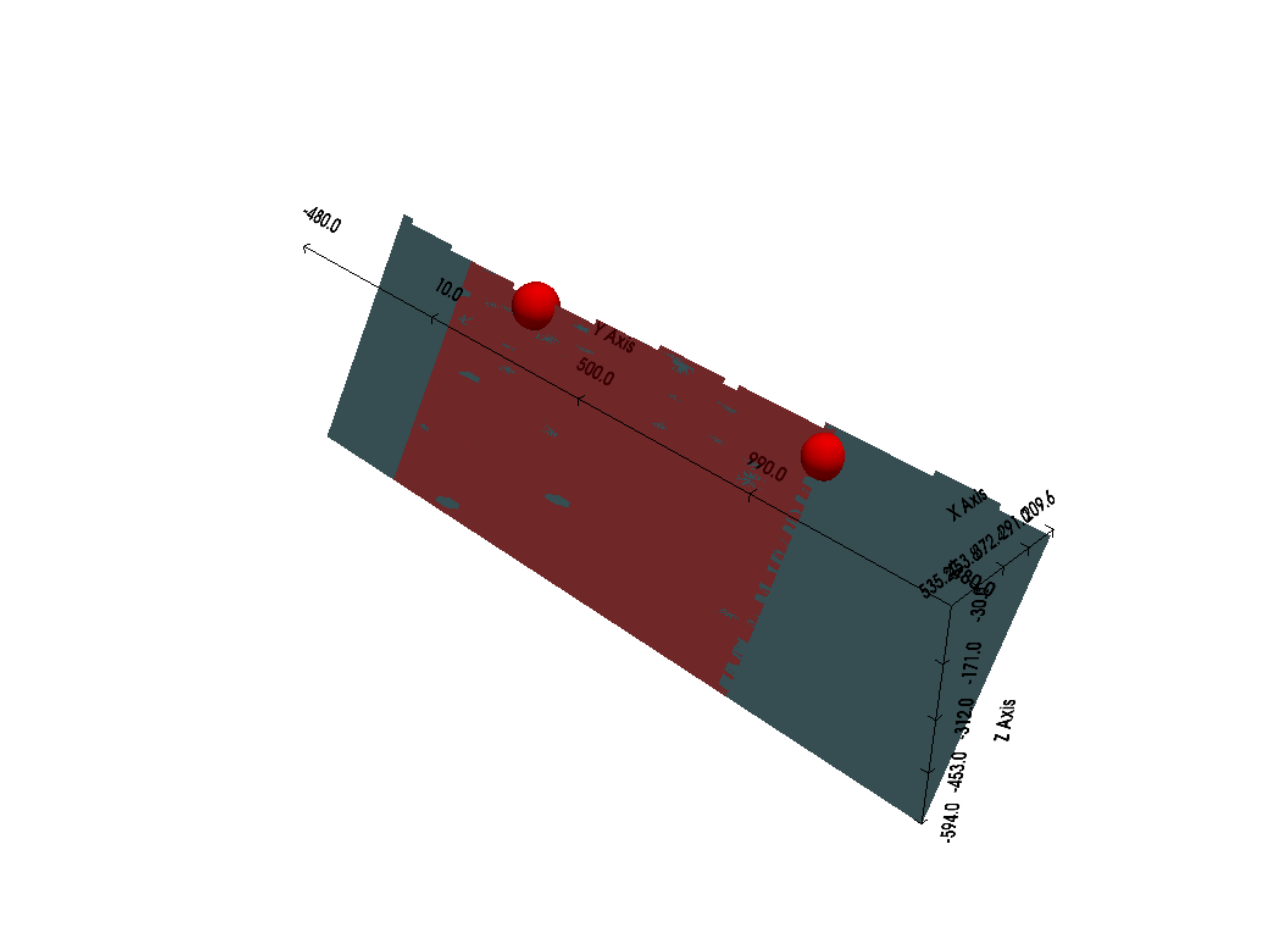
[12]:
```
mesh1 = gg.visualization.create_depth_maps_from_gempy(geo_model, surfaces='Fault1')
mesh1
```

[12]:
```
{'Fault1': [PolyData (0x2556ca0b5e0)
    N Cells:    5902
    N Points:   3070
    N Strips:   0
    X Bounds:   2.096e+02, 5.352e+02
    Y Bounds:   -4.800e+02, 1.480e+03
    Z Bounds:   -5.940e+02, -3.000e+01
    N Arrays:   1,
  '#527682']}
```

6.70.5 Plotting the Result

Plotting the result, we can see the interface points of the fault (red spheres), the original extent of the fault modeled by GemPy in gray and the clipped fault (finite fault) with a buffer on one side in red.

[13]:
```
p = pv.Plotter(notebook=True)

p.add_mesh(mesh['Fault1'][0], color='red', opacity=0.5)
p.add_mesh(mesh1['Fault1'][0], color=mesh1['Fault1'][1])

p.add_mesh(fault_polydata, color='red', point_size=40, render_points_as_spheres=True)

p.set_background('white')
p.set_scale(1,1,1)
p.show_bounds(color='black', font_size=12)

p.show()
```



6.71 69 Export GemPy model into blender

This notebook illustrates how GemPy models can be import and editted in Blender. A few lines are added after computing a model as usual. A small custom script is executed in Blender (download for free here: https://www.blender.org/download/) to build surfaces into your scene. We show first steps to edit the model and make it visually more appealing by increasing surface thicknesses and adding colors. This model can be used easily for further animation steps, which is beyond the scope of this tutorial. We recommend to "build the popular Blender Donut" to understand the concepts of Blender like keyframing for animations (https://www.youtube.com/watch?v=LMA3S2EGM6U). We want to mention Blender as an excellent tool to manually edit and export meshes for further compuational steps.

[1]:
```
import gempy as gp
import pandas as pd
import numpy as np
import os
```

```
WARNING (aesara.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

[2]:
```
file_path ='data/69_Gempy_to_Blender/'
# gg.download_gemgis_data.download_tutorial_data(filename="68_creating_finite_faults_
↪with_gemgis.zip", dirpath=file_path)
```

6.71.1 Read data generated from example 02

[3]:
```
interfaces_coords = pd.read_csv(file_path + 'surfaces_example02.csv')
interfaces_coords
```

[3]:
```
     Unnamed: 0 formation                                        geometry  \
0             0         P  POINT (1652.8907623591306 2.1487791778915835)   \
1             1         P  POINT (1847.1027866783184 185.95658790855168)
2             2         P   POINT (1994.4958408491307 342.0198217364706)
3             3         P   POINT (2121.080463842887 484.21076811301884)
4             4         P   POINT (2235.5268353166944 607.3273192439327)
..          ...       ...                                           ...
121         121         T    POINT (1476.886115319865 3471.087659986244)
122         122         T  POINT (1608.6728461078853 3525.7097918260156)
123         123         T    POINT (1768.204151798647 3591.603157220026)
124         124         T   POINT (1903.4589544495102 3650.560378888351)
125         125         T  POINT (1951.1449425635965 3671.3688100654067)

                X            Y           Z
0    1652.890762     2.148779  162.705959
1    1847.102787   185.956588  196.793667
2    1994.495841   342.019822  252.925457
3    2121.080464   484.210768  305.557260
4    2235.526835   607.327319  349.545114
..           ...          ...         ...
121  1476.886115  3471.087660  421.321855
122  1608.672846  3525.709792  436.155939
123  1768.204152  3591.603157  459.647882
124  1903.458954  3650.560379  486.010124
125  1951.144943  3671.368810  490.748650

[126 rows x 6 columns]
```

[4]:
```
orientations = pd.read_csv(file_path + 'orientations_example02.csv')
orientations
```

[4]:
```
   Unnamed: 0        dip     azimuth      Z  \
0           0  23.226872  174.963534  250.0   \
1           1  22.257867  174.667406  350.0
2           2  21.786345  174.412613  450.0
3           3  22.066141  174.294562  250.0
4           4  22.175549  174.498431  450.0
5           5  22.938797  173.778172  450.0
6           6  21.439609  173.811400  550.0
7           7  23.406748  174.121550  650.0
8           8  21.789527  174.515310  450.0

                                       geometry  polarity formation
0  POINT (1070.7098738335671 247.54737664772748)       1.0         P  \
1   POINT (1289.3121973006528 511.4405510986238)       1.0         P
2      POINT (1463.0788542159257 780.2269581935)       1.0         P
3    POINT (972.9319227505433 734.0203194834023)       1.0         Q
4    POINT (732.9847007401183 1953.372711009733)       1.0         R
5   POINT (797.4691619398486 2204.6995438201116)       1.0         S
```

(continues on next page)

(continued from previous page)

```
6   POINT (881.6782818594968 2462.9625203561745)       1.0         S
7   POINT (1009.3466773103917 2723.501419052229)       1.0         S
8  POINT (1052.3887004456406 3474.4636111902337)       1.0         T

             X            Y
0  1070.709874   247.547377
1  1289.312197   511.440551
2  1463.078854   780.226958
3   972.931923   734.020319
4   732.984701  1953.372711
5   797.469162  2204.699544
6   881.678282  2462.962520
7  1009.346677  2723.501419
8  1052.388700  3474.463611
```

6.71.2 Create new Model

[5]:
```
geo_model = gp.create_model('Model')
```

6.71.3 Initiate Data

[6]:
```
gp.init_data(geo_model, [0, 2932, 0, 3677, -700, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[6]:
```
Model  2023-10-23 13:59
```

[7]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata': ('P', 'Q', 'R', 'S', 'T')},
                         remove_unused_series=True)
geo_model.add_surfaces('U')
```

[7]:

	surface	series	order_surfaces	color	id
0	P	Strata	1	#015482	1
1	Q	Strata	2	#9f0052	2
2	R	Strata	3	#ffbe00	3
3	S	Strata	4	#728f02	4
4	T	Strata	5	#443988	5
5	U	Strata	6	#ff3f20	6

[8]:
```
geo_model.set_topography(source='gdal', filepath='./data/example02/raster2.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[8]:
```
Grid Object. Values:
array([[  14.66      ,   18.385     , -691.5       ],
       [  14.66      ,   18.385     , -674.5       ],
       [  14.66      ,   18.385     , -657.5       ],
       ...,
       [2926.99658703, 3652.02038043,  629.80548096],
       [2926.99658703, 3662.01222826,  629.07196045],
       [2926.99658703, 3672.00407609,  628.35705566]])
```

[9]:
```
gp.set_interpolator(geo_model,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

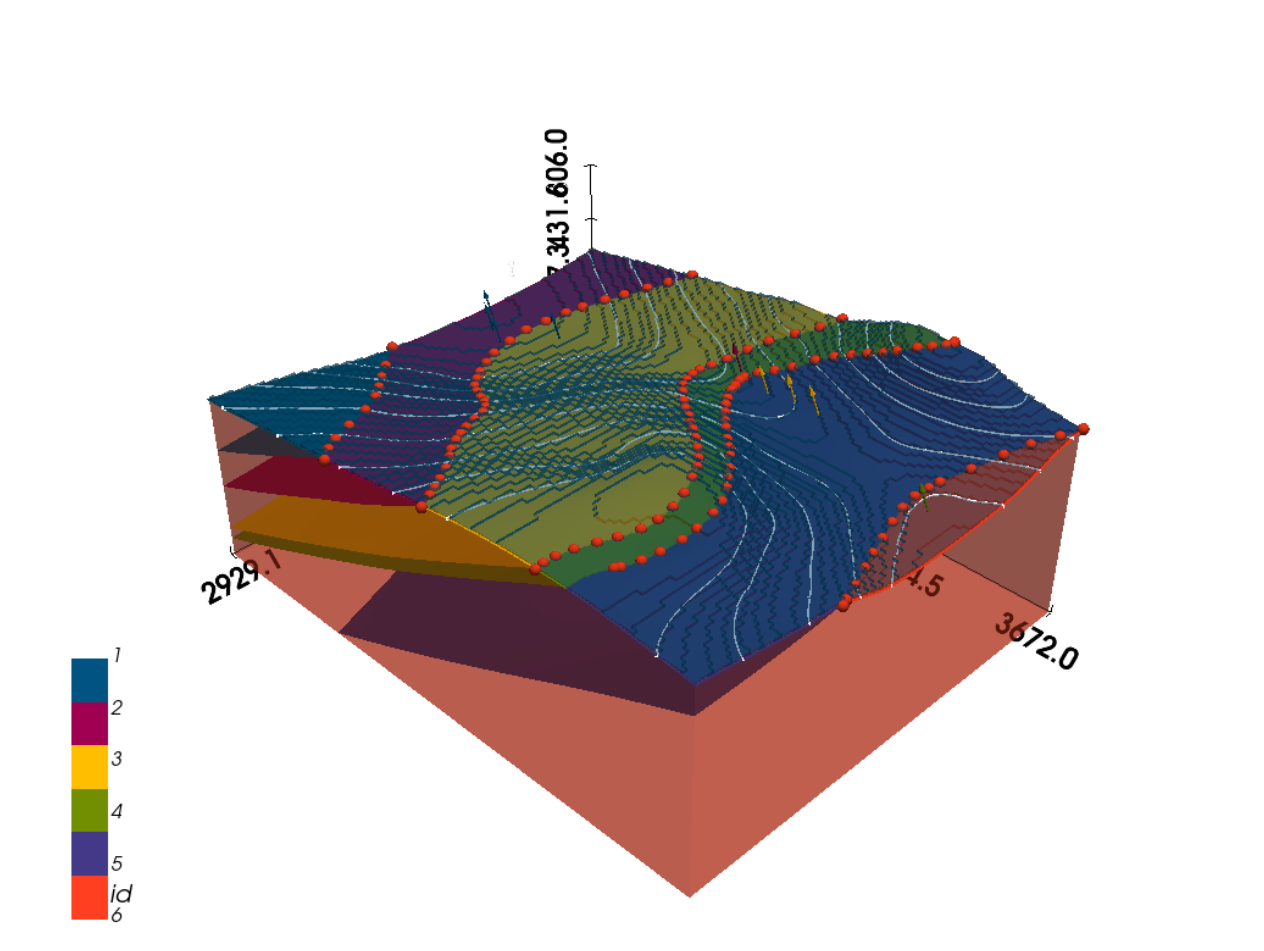
```
Compiling aesara function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                        values
range              5000.695252
$C_o$            595403.642857
drift equations              [3]
```

[9]:
```
<gempy.core.interpolator.InterpolatorModel at 0x19e8cffb610>
```

[10]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

[11]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```

```
C:\Users\chudalla\Desktop\Projects\gempy\env\lib\site-packages\pyvista\jupyter\notebook.
↪py:58: UserWarning: Failed to use notebook backend:

No module named 'trame'

Falling back to a static output.
  warnings.warn(
```



Get and store surfaces

In this step, the surface information (vertices and faces) is stored as npy files. Because Blender has a default origin at [0,0,0], real life coordinates can result in unhandy viewport navigation, we shift the model towards the default origin. For the same reason, we scale down the model to reduce the absolute width loaded into Blender. Depening on your aims, these steps might influence your outcome negatively!

[12]:
```
try:
    os.mkdir('vertices')
    os.mkdir('faces')
    print('Folders for mesh data created')

except:
    print('Folders already exist')
```

```
Folders already exist
```

[13]:
```
vertices, faces = gp.get_surfaces(geo_model)

# enter values to shift model
x0, y0, z0 = 0, 0, -700
```

(continues on next page)

(continued from previous page)

```
# save model
for i in range(len(vertices)):
    np.save('vertices/' + 'vert_%02d.npy' %i, vertices[i]-[x0, y0, z0])
    np.save('faces/' + 'faces_%02d.npy' %i, faces[i])
    print(len(vertices[i]))
```

```
1462
5148
12222
13722
15834
```

Import surface in Blender

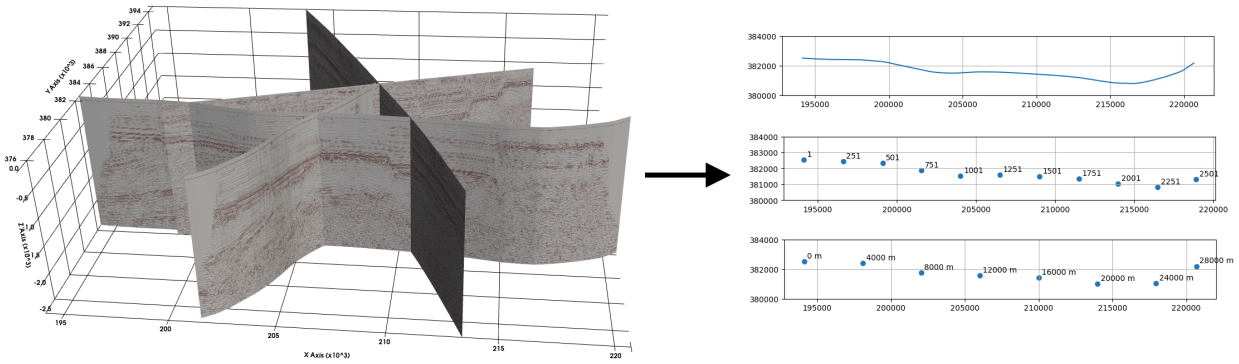
Adjust the storage locations of your vertice and face data (Line 7,8) in the script 'gempy_to_blender.txt' (downloaded in "data/69_Gempy_to_Blender"). Open Blender and open the scripting tab. Copy and paste the code. Then press the run symbol. For further steps, check out the appended screencast.

[14]:
```
%%HTML
<iframe width="560" height="315" src="https://www.youtube.com/embed/pDDMwdMjn3A?
↪si=gnhr5tlTn1zj85OR" title="YouTube video player" frameborder="0" allow="accelerometer;
↪ autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share"␣
↪allowfullscreen></iframe>
```

```
<IPython.core.display.HTML object>
```

6.72 70 Reprojecting Seismic Data and extracting path and CDP points from Seismic Data

This notebook illustrates how to reproject 2D seismic data from the original coordinate system to a new coordinate system. In this tutorial, we are making use of the `segysak` package (https://segysak.readthedocs.io/en/latest/) to load the seismic data in combination with `GeoPandas` (https://geopandas.org/en/stable/) to perform the coordinate reprojection. Further, it is shown how to export the path of the seismic data as LineString, as CDP points and as points every n-th meter along the line.



6.72.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")

# import gemgis as gg
import geopandas as gpd
import matplotlib.pyplot as plt
from typing import Union
import pyproj
import numpy as np
import shapely
from shapely.geometry import LineString
```

[2]:
```
file_path ='data/70_reprojecting_seismic_data/'
# gg.download_gemgis_data.download_tutorial_data(filename="70_reprojecting_seismic_data.
↪zip", dirpath=file_path)
```

6.72.2 Defining functions

As there may be conflicts between the dependencies of GemGIS and the `segysak` package, we define the functions that are needed to reproject the seismic data and for the further contents of the notebook. However, they are also implemented in the GemGIS `utils` module.

[3]:
```
def convert_crs_seismic_data(path_in: str,
                             path_out: str,
                             crs_in: Union[str, pyproj.crs.crs.CRS],
                             crs_out: Union[str, pyproj.crs.crs.CRS],
                             cdpx: int = 181,
                             cdpy: int = 185,
                             vert_domain: str = 'TWT',
                             coord_scalar: int = None):
    """Convert CDP coordinates of seismic data to a new CRS.

    Parameters
    ----------
        path_in : str
            Path to the original seismic data, e.g. ``path_in='seismic.sgy'``.
        path_out : str
            Path to the converted seismic data, e.g. ``path_out='seismic_converted.sgy'``.
        crs_in : str, pyproj.crs.crs.CRS
            Coordinate reference system of the original seismic data.
        crs_out : str, pyproj.crs.crs.CRS
            Coordinate reference system of the converted seismic data.
        cdpx : int
            Byte position for the X coordinates, default is ``cdpx=181``.
```

(continues on next page)

(continued from previous page)

```
        cdpy : int
            Byte position for the Y coordinates, default is ``cdpx=185``.
        vert_domain : str
            Vertical sampling domain. Options include ``'TWT'`` and ``'DEPTH'``, default is␣
↪``vert_domain='TWT'``.
        coord_scalar: int
            Coordinate scalar value to set if `NaN` columns are returned, default is␣
↪`coord_scalar=None`.

    .. versionadded:: 1.1.1

    """
    # Trying to import segysak but returning error if segysak is not installed
    try:
        from segysak.segy import segy_loader, segy_writer
    except ModuleNotFoundError:
        raise ModuleNotFoundError(
            'segysak package is not installed. Use pip install segysak to install the␣
↪latest version')

    # Checking that path_in is of type string
    if not isinstance(path_in, str):
        raise TypeError('path_in must be provided as string')

    # Checking that path_out is of type string
    if not isinstance(path_out, str):
        raise TypeError('path_out must be provided as string')

    # Checking that crs_in is of type string or pyproj CRS
    if not isinstance(crs_in, (str, pyproj.crs.crs.CRS)):
        raise TypeError('crs_in must be provided as string or pyproj CRS')

    # Checking that crs_out is of type string or pyproj CRS
    if not isinstance(crs_out, (str, pyproj.crs.crs.CRS)):
        raise TypeError('crs_out must be provided as string or pyproj CRS')

    # Checking that vert_domain is of type str
    if not isinstance(vert_domain, str):
        raise TypeError('vert_domain must be provided as string')

    # Checking that the coord_scalar is of type int or None
    if not isinstance(coord_scalar, (int, type(None))):
        raise TypeError('coord_scalar must be provided as int')

    # Loading seismic data
    seismic = segy_loader(path_in,
                          vert_domain=vert_domain,
                          cdpx=cdpx,
                          cdpy=cdpy)

    # Converting Seismic to DataFrame
    df_seismic = seismic.to_dataframe()
```

(continues on next page)

(continued from previous page)

```
    # Checking that the CDP coordinates are in the DataFrame
    if not {'cdp_x', 'cdp_y'}.issubset(df_seismic.columns):
        raise ValueError(
            'No coordinates found, please provide the byte positions where the X and Y␣
↪data of the CDPs is stored')

    # Extracting the length of the samples to reduce computing time
    samples = len(df_seismic.index.get_level_values(1).unique())

    # Resample DataFrame
    df_seismic_resampled = df_seismic[::samples]

    # Reprojecting Coordinates
    df_seismic_reprojected = gpd.GeoDataFrame(geometry=gpd.points_from_xy(x=df_seismic_
↪resampled['cdp_x'].values,
                                                                          y=df_seismic_
↪resampled['cdp_y'].values),
                                              crs=crs_in).to_crs(crs_out)

    # Extracting reprojected coordinates
    x = df_seismic_reprojected.geometry.x.values
    y = df_seismic_reprojected.geometry.y.values

    # Assigning new coordinates
    seismic['cdp_x'][:] = x
    seismic['cdp_y'][:] = y

    # Optionally setting a new coord_scalar
    if coord_scalar:
        seismic.attrs['coord_scalar'] = coord_scalar

    # Saving reprojected seismic data to file
    segy_writer(seismic,
                path_out,
                trace_header_map=dict(cdp_x=181,
                                      cdp_y=185))

    print('Seismic data was successfully reprojected and saved to file')


def get_cdp_linestring_of_seismic_data(path_in: str,
                                       crs_in: Union[str, pyproj.crs.crs.CRS],
                                       cdpx: int = 181,
                                       cdpy: int = 185,
                                       vert_domain: str = 'TWT'):
    """Extracting the path of the seismic data as LineString.

    Parameters
    ----------
        path_in : str
            Path to the original seismic data, e.g. ``path_in='seismic.sgy'``.
```

(continues on next page)

(continued from previous page)

```
        crs_in : str, pyproj.crs.crs.CRS
            Coordinate reference system of the original seismic data.
        cdpx : int
            Byte position for the X coordinates, default is ``cdpx=181``.
        cdpy : int
            Byte position for the Y coordinates, default is ``cdpx=185``.
        vert_domain : str
            Vertical sampling domain. Options include ``'TWT'`` and ``'DEPTH'``, default is␣
↪``vert_domain='TWT'``.

    Returns
    _______
        gpd.GeoDataFrame
            GeoDataFrame containing the surface path of the seismic data as LineString.

    .. versionadded:: 1.1.1

    """
    # Trying to import segysak but returning error if segysak is not installed
    try:
        from segysak.segy import segy_loader, segy_writer
    except ModuleNotFoundError:
        raise ModuleNotFoundError(
            'segysak package is not installed. Use pip install segysak to install the␣
↪latest version')

    # Checking that path_in is of type string
    if not isinstance(path_in, str):
        raise TypeError('path_in must be provided as string')

    # Checking that crs_in is of type string or pyproj CRS
    if not isinstance(crs_in, (str, pyproj.crs.crs.CRS)):
        raise TypeError('crs_in must be provided as string or pyproj CRS')

    # Checking that vert_domain is of type str
    if not isinstance(vert_domain, str):
        raise TypeError('vert_domain must be provided as string')

    # Loading seismic data
    seismic = segy_loader(path_in,
                          vert_domain=vert_domain,
                          cdpx=cdpx,
                          cdpy=cdpy)

    # Converting Seismic to DataFrame
    df_seismic = seismic.to_dataframe()

    # Checking that the CDP coordinates are in the DataFrame
    if not {'cdp_x', 'cdp_y'}.issubset(df_seismic.columns):
        raise ValueError(
            'No coordinates found, please provide the byte positions where the X and Y␣
↪data of the CDPs is stored')
```

(continues on next page)

(continued from previous page)

```
    # Extracting the length of the samples to reduce computing time
    samples = len(df_seismic.index.get_level_values(1).unique())

    # Resample DataFrame
    df_seismic_resampled = df_seismic[::samples]

    # Creating LineString from coordinates
    linestring = LineString(np.c_[(df_seismic_resampled['cdp_x'].values,
                                   df_seismic_resampled['cdp_y'].values)])

    # Reprojecting Coordinates
    gdf_seismic = gpd.GeoDataFrame(geometry=[linestring],
                                   crs=crs_in)

    return gdf_seismic


def get_cdp_points_of_seismic_data(path_in: str,
                                   crs_in: Union[str, pyproj.crs.crs.CRS],
                                   cdpx: int = 181,
                                   cdpy: int = 185,
                                   vert_domain: str = 'TWT',
                                   filter: int = None,
                                   n_meter: Union[int, float] = None):
    """Extracting the path of the seismic data as LineString.

    Parameters
    __________
        path_in : str
            Path to the original seismic data, e.g. ``path_in='seismic.sgy'``.
        crs_in : str, pyproj.crs.crs.CRS
            Coordinate reference system of the original seismic data.
        cdpx : int
            Byte position for the X coordinates, default is ``cdpx=181``.
        cdpy : int
            Byte position for the Y coordinates, default is ``cdpx=185``.
        vert_domain : str
            Vertical sampling domain. Options include ``'TWT'`` and ``'DEPTH'``, default is␣
↪``vert_domain='TWT'``.
        filter : int
            Filtering the points to only return every n-th point, e.g. ``filter=100`` to␣
↪return only every 100-th point.
        n_meter : int, float
            Parameter to select a point along the line every n-th meter.

    Returns
    _______
        gpd.GeoDataFrame
            GeoDataFrame containing the CDPs as Points.

    .. versionadded:: 1.1.1
```

(continues on next page)

(continued from previous page)

```
    """
    # Trying to import segysak but returning error if segysak is not installed
    try:
        from segysak.segy import segy_loader, segy_writer
    except ModuleNotFoundError:
        raise ModuleNotFoundError(
            'segysak package is not installed. Use pip install segysak to install the␣
↪latest version')

    # Checking that path_in is of type string
    if not isinstance(path_in, str):
        raise TypeError('path_in must be provided as string')

    # Checking that crs_in is of type string or pyproj CRS
    if not isinstance(crs_in, (str, pyproj.crs.crs.CRS)):
        raise TypeError('crs_in must be provided as string or pyproj CRS')

    # Checking that vert_domain is of type str
    if not isinstance(vert_domain, str):
        raise TypeError('vert_domain must be provided as string')

    # Loading seismic data
    seismic = segy_loader(path_in,
                          vert_domain=vert_domain,
                          cdpx=cdpx,
                          cdpy=cdpy)

    # Converting Seismic to DataFrame
    df_seismic = seismic.to_dataframe()

    # Checking that the CDP coordinates are in the DataFrame
    if not {'cdp_x', 'cdp_y'}.issubset(df_seismic.columns):
        raise ValueError(
            'No coordinates found, please provide the byte positions where the X and Y␣
↪data of the CDPs is stored')

    # Extracting the length of the samples to reduce computing time
    samples = len(df_seismic.index.get_level_values(1).unique())

    # Resample DataFrame
    df_seismic_resampled = df_seismic[::samples]

    if n_meter:

        # Creating LineString from coordinates
        linestring = LineString(np.c_[(df_seismic_resampled['cdp_x'].values,
                                       df_seismic_resampled['cdp_y'].values)])

        # Defining number of samples
        samples = np.arange(0, round(linestring.length / n_meter) + 1, 1)
```

(continues on next page)

(continued from previous page)

```
        # Getting points every n_meter
        points = [shapely.line_interpolate_point(linestring, n_meter * sample) for␣
↪sample in samples]

        # Creating GeoDataFrame from points
        gdf_seismic = gpd.GeoDataFrame(geometry=points,
                                       crs=crs_in)

        # Appending distance
        gdf_seismic['distance'] = samples * n_meter

    else:
        # Creating Points from coordinates
        gdf_seismic = gpd.GeoDataFrame(geometry=gpd.points_from_xy(x=df_seismic_
↪resampled['cdp_x'].values,
                                                                   y=df_seismic_
↪resampled['cdp_y'].values),
                                       data=df_seismic_resampled,
                                       crs=crs_in).reset_index().drop(['twt', 'data'],␣
↪axis=1)

        # Returning only every nth point
        if filter:
            gdf_seismic = gdf_seismic[::filter]

    return gdf_seismic
```

6.72.3 Reprojecting Seismic Data

The seismic data used here can be freely downloaded from NLOG.nl and is provided in the Coordinate References System (CRS) with `EPSG:28992`. To reproject the data, we are using the function `convert_crs_seismic_data(...)`. The output path and the output CRS are defined, optionally, the byte positions for the X and Y coordinates of each CDP and the vertical domain can be defined.

[4]:
```
convert_crs_seismic_data(path_in=file_path+'Seismic_EPSG_28992.sgy',
                         path_out=file_path+'Seismic_EPSG_25832.sgy',
                         crs_in='EPSG:28992',
                         crs_out='EPSG:25832',
                         cdpx = 181,
                         cdpy = 185,
                         vert_domain = 'TWT')
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

```
Loading as 2D
```

```
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

```
Seismic data was successfully reprojected and saved to file
```

6.72.4 Accounting for coordinate scalar errors

It may happen that columns are filled with NaN values. The problem is not so much the Python side but the SEGY side. For a more detailed description, see https://github.com/trhallam/segysak/issues/109#issuecomment-1717610828. The error can be mitigated by providing a coord_scalar value.

[5]:
```
from segysak.segy import segy_loader
seismic = segy_loader(file_path+'Seismic_EPSG_25832.sgy',
                          cdpx = 181,
                          cdpy = 185,
                          vert_domain = 'TWT')

# Converting Seismic to DataFrame
df_seismic = seismic.to_dataframe()
df_seismic
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

```
Loading as 2D
```

```
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

[5]:
```
             data         cdp_x  cdp_y
cdp  twt
1    0.0      0.0  287958.90625    NaN
     2.0      0.0  287958.90625    NaN
     4.0      0.0  287958.90625    NaN
     6.0      0.0  287958.90625    NaN
     8.0      0.0  287958.90625    NaN
...           ...           ...    ...
2702 5092.0   0.0  314511.46875    NaN
     5094.0   0.0  314511.46875    NaN
     5096.0   0.0  314511.46875    NaN
     5098.0   0.0  314511.46875    NaN
     5100.0   0.0  314511.46875    NaN

[6892802 rows x 3 columns]
```

6.72.5 Reprojecting Seismic Data - again

As you can see, reprojecting the seismic data again using a coord_scalar will add the correct coordinates to the segy file.

[6]:
```
convert_crs_seismic_data(path_in=file_path+'Seismic_EPSG_28992.sgy',
                         path_out=file_path+'Seismic_EPSG_25832.sgy',
                         crs_in='EPSG:28992',
                         crs_out='EPSG:25832',
                         cdpx = 181,
                         cdpy = 185,
                         vert_domain = 'TWT',
                         coord_scalar=-100)
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

```
Loading as 2D
```

```
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

```
Seismic data was successfully reprojected and saved to file
```

[7]:
```
from segysak.segy import segy_loader
seismic = segy_loader(file_path+'Seismic_EPSG_25832.sgy',
                          cdpx = 181,
                          cdpy = 185,
                          vert_domain = 'TWT')

# Converting Seismic to DataFrame
df_seismic = seismic.to_dataframe()
df_seismic
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

```
Loading as 2D
```

```
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

[7]:
```
             data        cdp_x      cdp_y
cdp  twt
1    0.0      0.0  287958.8750  5702121.5
     2.0      0.0  287958.8750  5702121.5
     4.0      0.0  287958.8750  5702121.5
     6.0      0.0  287958.8750  5702121.5
     8.0      0.0  287958.8750  5702121.5
...           ...          ...        ...
2702 5092.0   0.0  314511.4375  5700494.0
     5094.0   0.0  314511.4375  5700494.0
     5096.0   0.0  314511.4375  5700494.0
     5098.0   0.0  314511.4375  5700494.0
     5100.0   0.0  314511.4375  5700494.0

[6892802 rows x 3 columns]
```

6.72.6 Plotting the results

We are now plotting the surface trace of both seismic lines to illustrate the change in coordinates.

[8]:
```
seismic_old = segy_loader(file_path+'Seismic_EPSG_28992.sgy',
                          vert_domain='TWT',
                          cdpx=181,
                          cdpy=185)
# Converting Seismic to DataFrame
df_seismic_old = seismic_old.to_dataframe()
df_seismic_old
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

```
Loading as 2D
```

```
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

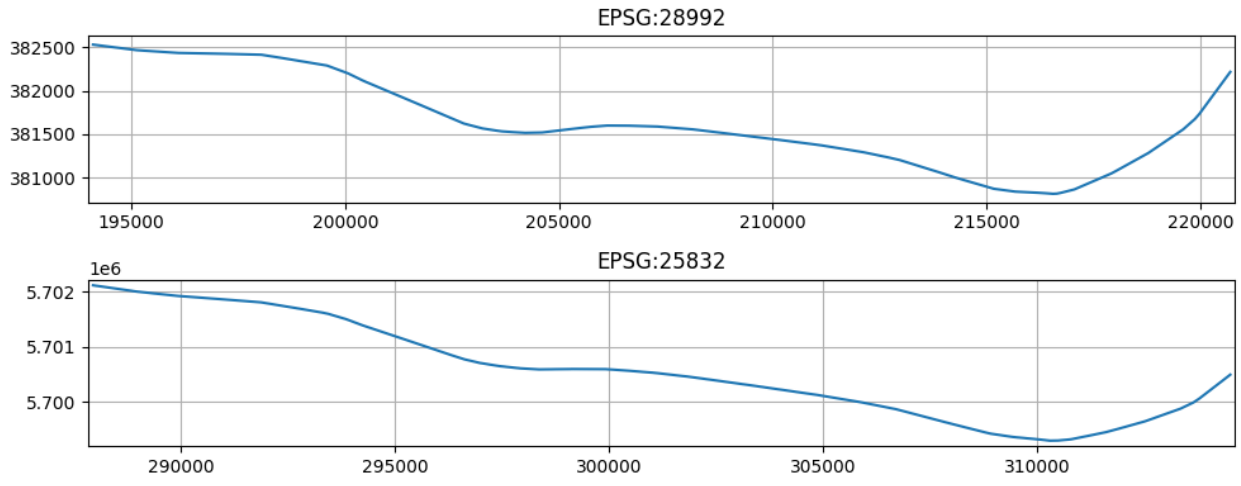
[8]:
```
             data          cdp_x        cdp_y
cdp  twt
```

(continues on next page)

(continued from previous page)

```
1    0.0      0.0  194105.421875  382534.0625
     2.0      0.0  194105.421875  382534.0625
     4.0      0.0  194105.421875  382534.0625
     6.0      0.0  194105.421875  382534.0625
     8.0      0.0  194105.421875  382534.0625
...           ...            ...          ...
2702 5092.0   0.0  220702.828125  382218.6250
     5094.0   0.0  220702.828125  382218.6250
     5096.0   0.0  220702.828125  382218.6250
     5098.0   0.0  220702.828125  382218.6250
     5100.0   0.0  220702.828125  382218.6250

[6892802 rows x 3 columns]
```

[9]:
```
fig, ax = plt.subplots(2, figsize = (10,4))

ax[0].plot(df_seismic_old[::2551]['cdp_x'], df_seismic_old[::2551]['cdp_y'])
ax[0].grid()

ax[0].set_xlim(min(df_seismic_old[::2551]['cdp_x'])-100, max(df_seismic_old[::2551]['cdp_
↪x'])+100)
ax[0].set_ylim(min(df_seismic_old[::2551]['cdp_y'])-100, max(df_seismic_old[::2551]['cdp_
↪y'])+100)
ax[0].set_title('EPSG:28992')

ax[1].plot(df_seismic[::2551]['cdp_x'], df_seismic[::2551]['cdp_y'])
ax[1].grid()

ax[1].set_xlim(min(df_seismic[::2551]['cdp_x'])-100, max(df_seismic[::2551]['cdp_x
↪'])+100)
ax[1].set_ylim(min(df_seismic[::2551]['cdp_y'])-100, max(df_seismic[::2551]['cdp_y
↪'])+100)
ax[1].set_title('EPSG:25832')

plt.tight_layout()
```



6.72.7 Extracting the path of the seismic data

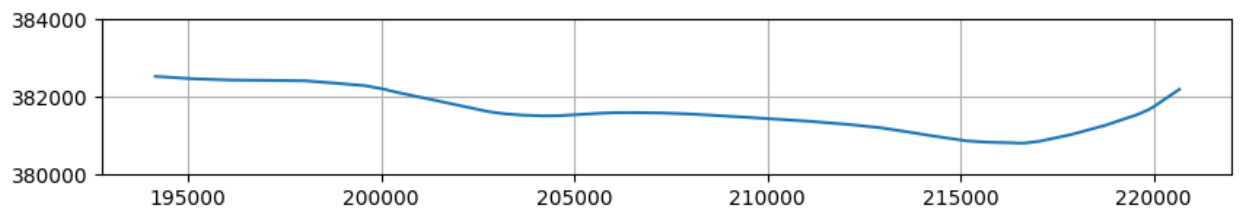
The path of the seismic data can be extracted using the function `get_cdp_linestring_of_seismic_data`. A GeoDataFrame containing a LineString will be returned.

[10]:
```
gdf_linestring = get_cdp_linestring_of_seismic_data(path_in=file_path+'Seismic_EPSG_
↪28992.sgy',
                                                    crs_in='EPSG:28992')
gdf_linestring
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
Loading as 2D
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

[10]:
```
                                            geometry
0  LINESTRING (194105.422 382534.062, 194115.391 ...
```

[11]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

gdf_linestring.plot(ax=ax)

ax.set_ylim(380000,384000)
ax.grid()
```



6.72.8 Extracting the CDP points of the seismic data

The CDP points of the seismic data can be extracted using the function `get_cdp_points_of_seismic_data`. By setting a `filter`, only the n-th point can be selected. A GeoDataFrame containing points will be returned.

[12]:
```
gdf_points = get_cdp_points_of_seismic_data(path_in=file_path+'Seismic_EPSG_28992.sgy',
                                            crs_in='EPSG:28992',
                                            filter=250)
gdf_points
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
Loading as 2D
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

[12]:

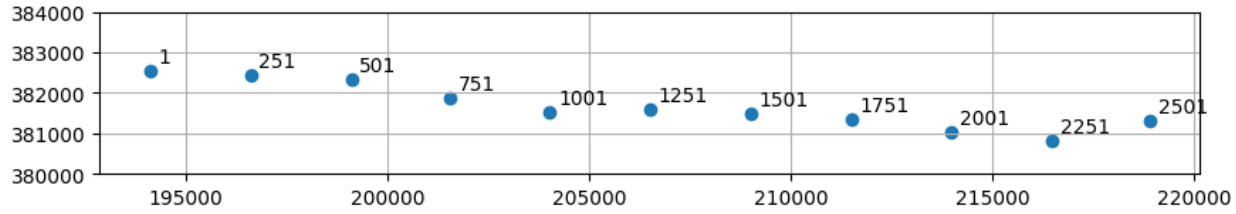
	cdp	cdp_x	cdp_y	geometry
0	1	194105.421875	382534.06250	POINT (194105.422 382534.062)
250	251	196602.859375	382433.34375	POINT (196602.859 382433.344)
500	501	199099.406250	382333.03125	POINT (199099.406 382333.031)
750	751	201554.859375	381880.46875	POINT (201554.859 381880.469)
1000	1001	204022.796875	381520.40625	POINT (204022.797 381520.406)

(continues on next page)

(continued from previous page)

```
1250  1251  206520.484375  381599.18750  POINT (206520.484 381599.188)
1500  1501  209018.203125  381503.37500  POINT (209018.203 381503.375)
1750  1751  211512.828125  381340.65625  POINT (211512.828 381340.656)
2000  2001  213993.937500  381043.43750  POINT (213993.938 381043.438)
2250  2251  216479.656250  380814.06250  POINT (216479.656 380814.062)
2500  2501  218918.343750  381326.12500  POINT (218918.344 381326.125)
```

[13]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

for label, x, y in zip(gdf_points['cdp'], gdf_points['cdp_x'], gdf_points['cdp_y']):
    plt.text(x+200,y+200,label)

gdf_points.plot(ax=ax)

ax.set_ylim(380000,384000)
ax.grid()
```



6.72.9 Extracting the CDP Points every n-th meter

A point every n-th meter along the seismic data can be extracted using the function `get_cdp_points_of_seismic_data`. By setting a value for the parameter `n_meter`, points only every n-th will be returned. A GeoDataFrame containing points will be returned. Be aware that these may not be the CDP points!

[14]:
```
gdf_points = get_cdp_points_of_seismic_data(path_in=file_path+'Seismic_EPSG_28992.sgy',
                                            crs_in='EPSG:28992',
                                            n_meter=4000)
gdf_points.head()
```

```
  0%|          | 0.00/2.70k [00:00<?, ? traces/s]
Loading as 2D
Converting SEGY:   0%|          | 0.00/2.70k [00:00<?, ? traces/s]
```

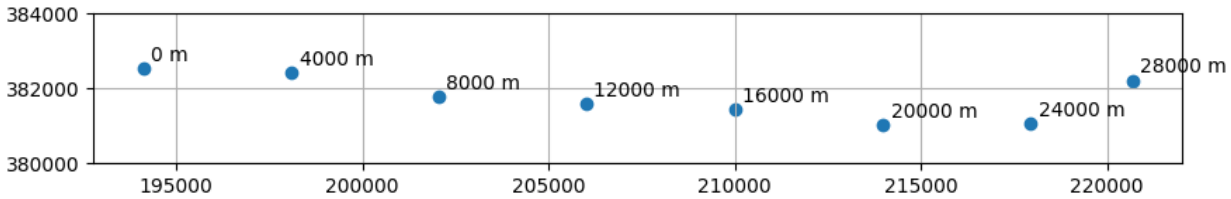
[14]:
```
                        geometry  distance
0  POINT (194105.422 382534.062)         0
1  POINT (198102.572 382412.468)      4000
2  POINT (202044.081 381777.218)      8000
3  POINT (206020.565 381595.751)     12000
4  POINT (210016.378 381442.716)     16000
```

[15]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

for label, x, y in zip(gdf_points['distance'], gdf_points.geometry.x, gdf_points.
↪geometry.y):
```

(continues on next page)

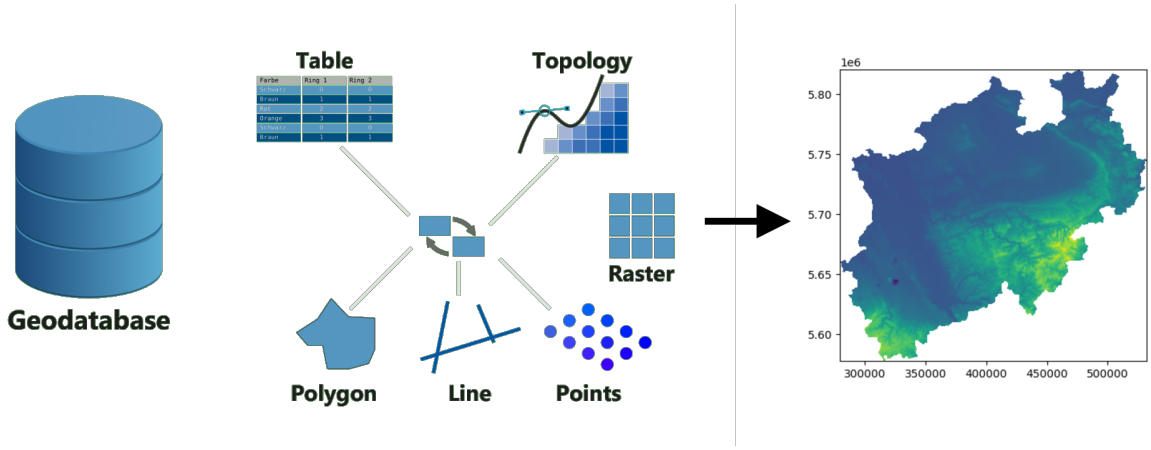
(continued from previous page)

```
    plt.text(x+200,y+200,str(label) + ' m')

gdf_points.plot(ax=ax)

ax.set_ylim(380000,384000)
ax.grid()
```



6.73 71 Opening Rasters from OpenFileGDB

This notebook illustrates how to open rasters that are stored in OpenFileGDBs. Currently, QGIS can open rasters from OpenFileGDBs since version 3.32. GeoPandas and the underlying fiona package are capable of opening vector files from OpenFileGDBs. It is not planned to support raster files (https://github.com/geopandas/geopandas/issues/2793). It is not planned to implement that feature into rasterio either (https://github.com/rasterio/rasterio/discussions/2914). Therefore, the decision was made to implement the functionality to open rasters from OpenFileGDBs directly using the `gdal` package.



6.73.1 Set File Paths and download Tutorial Data

If you downloaded the latest `GemGIS` version from the Github repository, append the path so that the package can be imported successfully. Otherwise, it is recommended to install `GemGIS` via `pip install gemgis` and import `GemGIS` using `import gemgis as gg`. In addition, the file path to the folder where the data is being stored is set. The tutorial data is downloaded using Pooch (https://www.fatiando.org/pooch/latest/index.html) and stored in the specified folder. Use `pip install pooch` if Pooch is not installed on your system yet.

[1]:
```
import warnings
warnings.filterwarnings("ignore")
import rasterio
```

(continues on next page)

(continued from previous page)

```
from rasterio.plot import show
import gemgis as gg
```

[2]:
```
file_path ='data/71_Opening_Rasters_From_OpenFileGDB/'
```

6.73.2 Opening OpenFileGDB

The OpenFileGDB can be opened using the GemGIS function `gg.raster.read_raster_gdb(...)`. It will automatically save all included rasters to file. If there is no projection included in the OpenFileGDB, a `crs` must be provided manually.

[4]:
```
gemgis.raster.read_raster_gdb(path=file_path + 'OpenFileGDB.gdb',
                              crs='EPSG:25832',
                              path_out=file_path)
```

```
RasterCenoman_Turon_top.tif successfully saved to file
RasterCenoman_Turon_m.tif successfully saved to file
RasterDevon_MK_m.tif successfully saved to file
RasterDevon_MK_top.tif successfully saved to file
RasterUKarb_m.tif successfully saved to file
RasterUKarb_top.tif successfully saved to file
RasterDGM.tif successfully saved to file
```

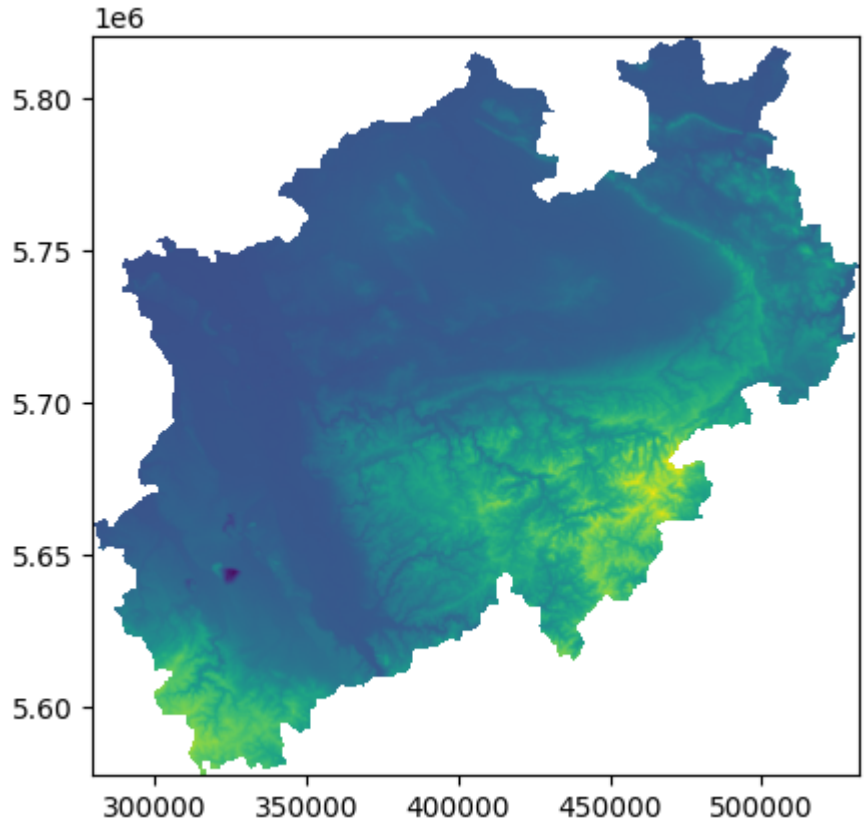
6.73.3 Opening Rasters using Rasterio

The downloaded rasters can be opened and displayed using the rasterio package as any other raster.

[5]:
```
dem = rasterio.open(file_path + 'RasterDGM.tif')
dem
```

[5]:
```
<open DatasetReader name='data/71_Opening_Rasters_From_OpenFileGDB/RasterDGM.tif' mode='r
↪'>
```

[6]:
```
show(dem);
```


1e6
5.80
5.75
5.70
5.65
5.60
300000
350000
400000
450000
500000

CHAPTER

SEVEN

EXAMPLE MODELS

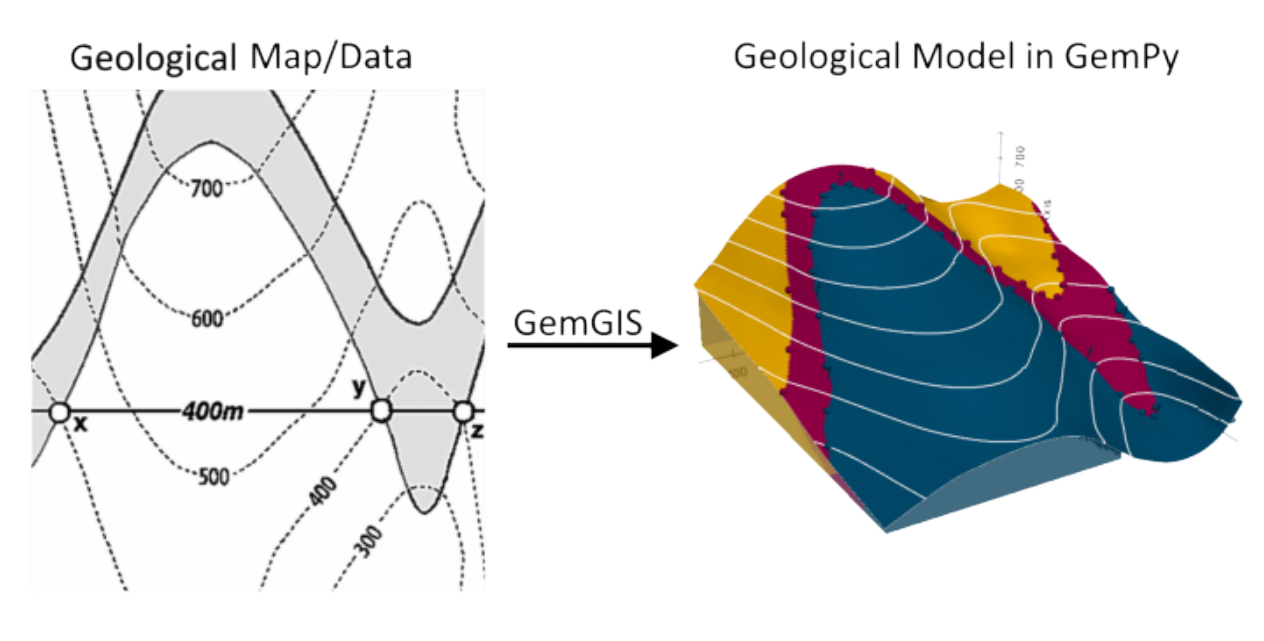
There is a series of examples available for GemGIS. In order to keep the size of the main GemGIS package as small as possible, the data is provided through a separated repository gemgis-data. You can also download the data directly following this link.

7.1 Example 1 - Planar Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 972 m wide (W-E extent) and 1069 m high (N-S extent). The vertical model extents varies between 300 m and 800 m. The model represents two planar stratigraphic units (blue and red) dipping towards the south above an unspecified basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and were used to calculate orientations for the `GemPy` model. These will be loaded into the model directly. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.1.1 Licensing

Computational Geosciences and Reservoir Engineering, RWTH Aachen University, Authors: Alexander Juestel. For more information contact: alexander.juestel(at)rwth-aachen.de

This work is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/)

7.1.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.1.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

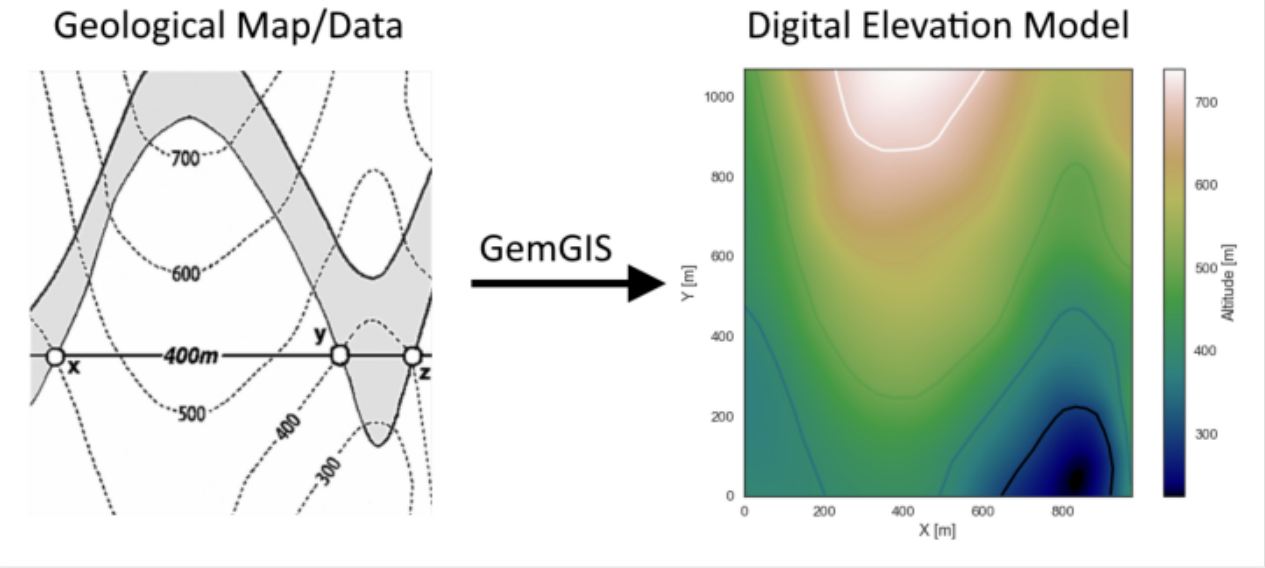
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example01/'
gg.download_gemgis_data.download_tutorial_data(filename="example01_planar_dipping_layers.
↪zip", dirpath=file_path)
```

7.1.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
img = mpimg.imread('../images/dem_example1.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo1.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	400	LINESTRING (0.741 475.441, 35.629 429.247, 77...
1	None	300	LINESTRING (645.965 0.525, 685.141 61.866, 724...
2	None	400	LINESTRING (490.292 0.525, 505.756 40.732, 519...
3	None	600	LINESTRING (911.433 1068.585, 908.856 1026.831...
4	None	700	LINESTRING (228.432 1068.585, 239.772 1017.037...

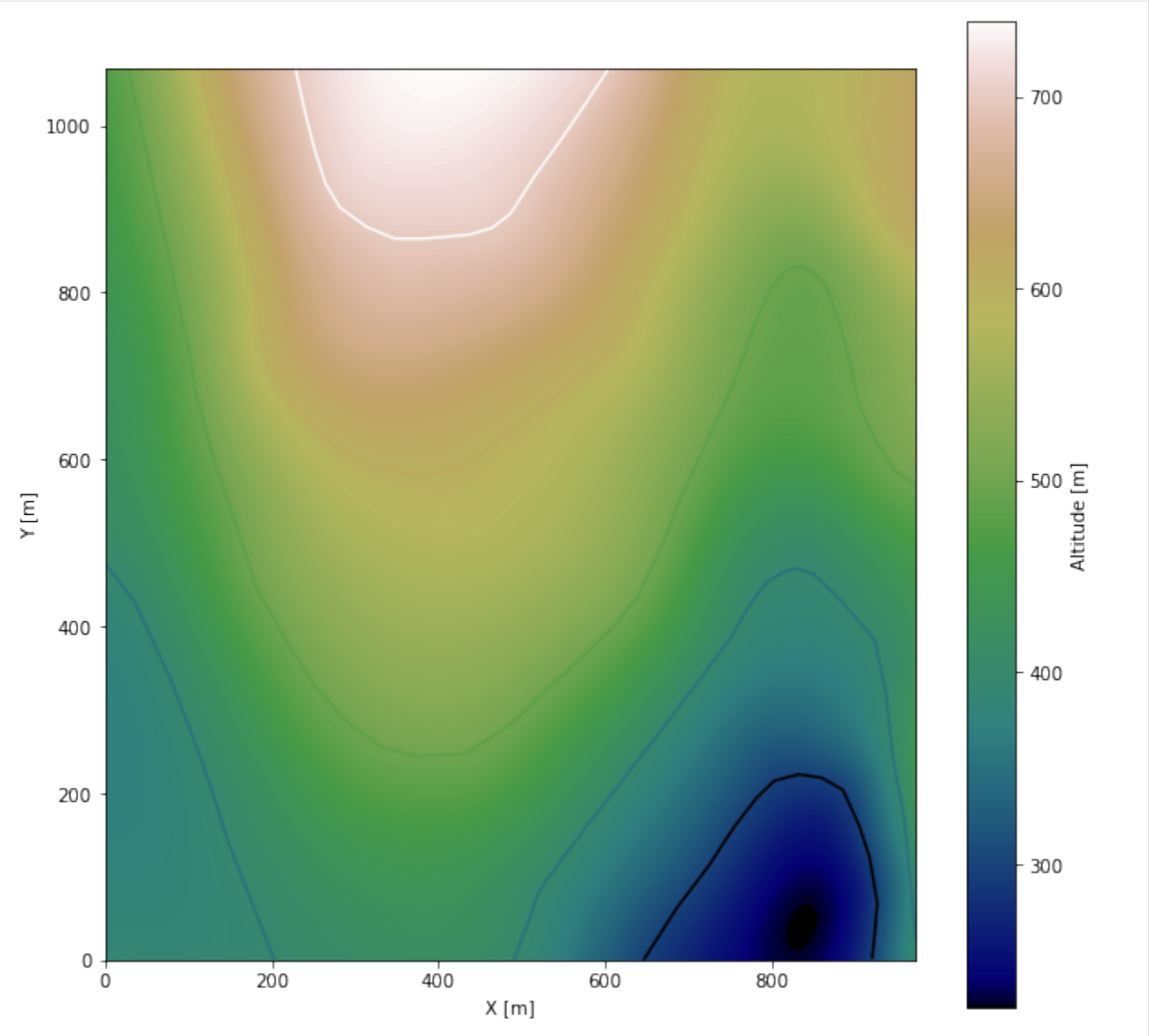
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 972, 0, 1069], cmap='gist_earth')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 972)
ax.set_ylim(0, 1069)
```

[8]:
```
(0.0, 1069.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster1.tif', extent = [0, 972, 0, 1069], crs ='EPSG : 4326', overwrite_f ile = True)$

Opening Raster

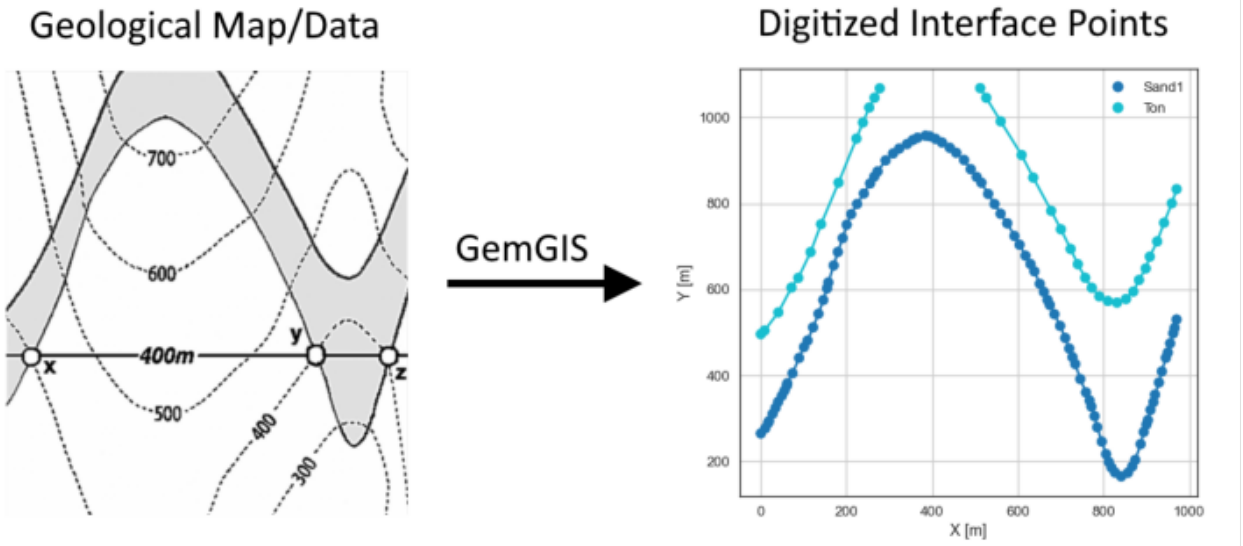
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster1.tif')
```

7.1.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../images/interfaces_example1.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces1_lines.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Sand1	LINESTRING (0.256 264.862, 10.593 276.734, 17...
1	None	Ton	LINESTRING (0.188 495.787, 8.841 504.142, 41.0...
2	None	Ton	LINESTRING (970.677 833.053, 959.372 800.023, ...

Extracting Z coordinate from Digital Elevation Model

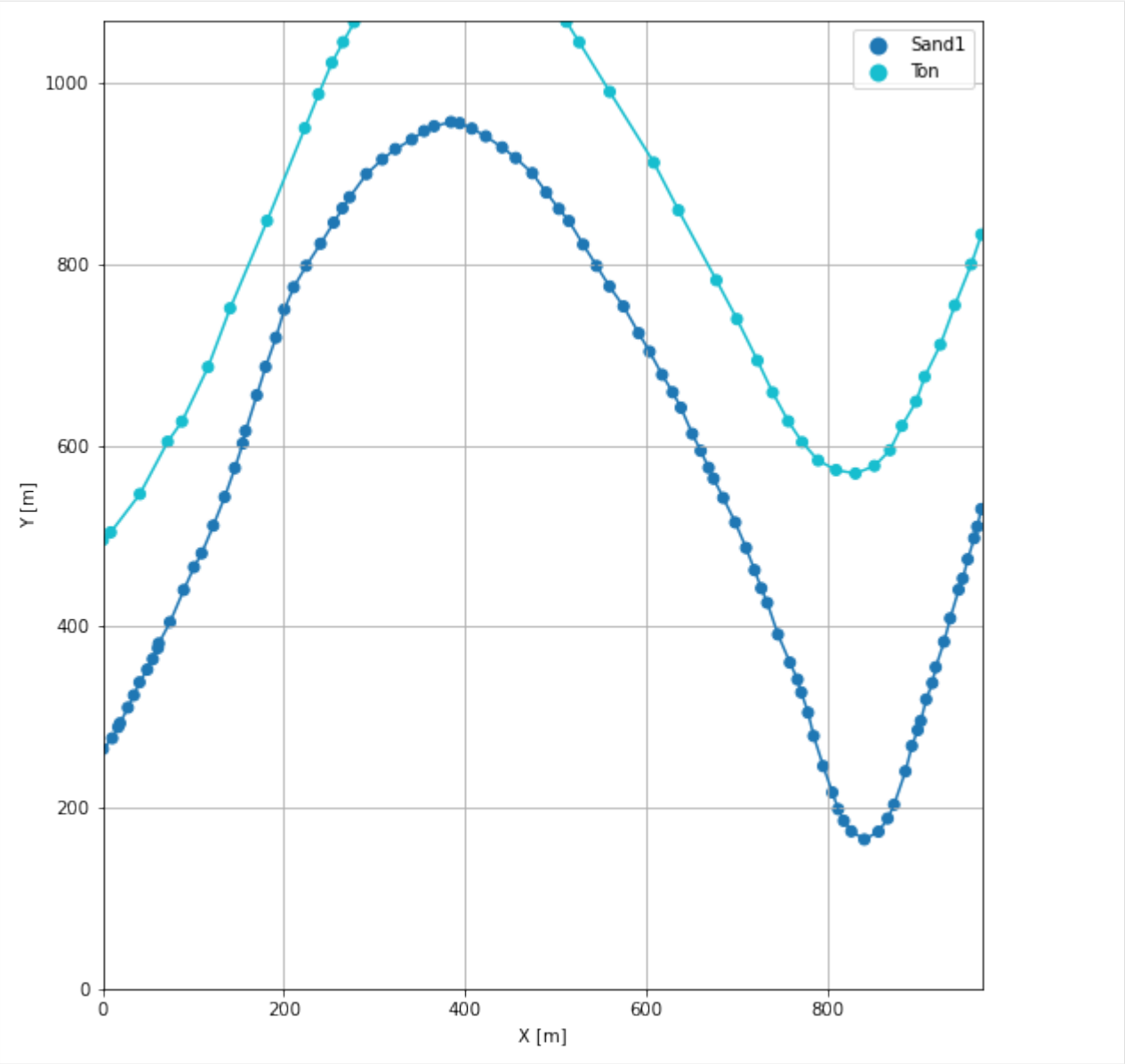
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords
```

[12]:
```
    formation                  geometry      X       Y      Z
0       Sand1     POINT (0.256 264.862)   0.26  264.86 353.97
1       Sand1    POINT (10.593 276.734)  10.59  276.73 359.04
2       Sand1    POINT (17.135 289.090)  17.13  289.09 364.28
3       Sand1    POINT (19.150 293.313)  19.15  293.31 364.99
4       Sand1    POINT (27.795 310.572)  27.80  310.57 372.81
..        ...                      ...    ...     ...    ...
126       Ton   POINT (636.023 859.788) 636.02  859.79 618.32
127       Ton   POINT (608.851 912.396) 608.85  912.40 647.91
128       Ton   POINT (560.110 990.617) 560.11  990.62 697.06
129       Ton  POINT (526.375 1045.388) 526.38 1045.39 724.56
130       Ton  POINT (512.240 1067.951) 512.24 1067.95 734.76

[131 rows x 5 columns]
```

Plotting the Interface Points

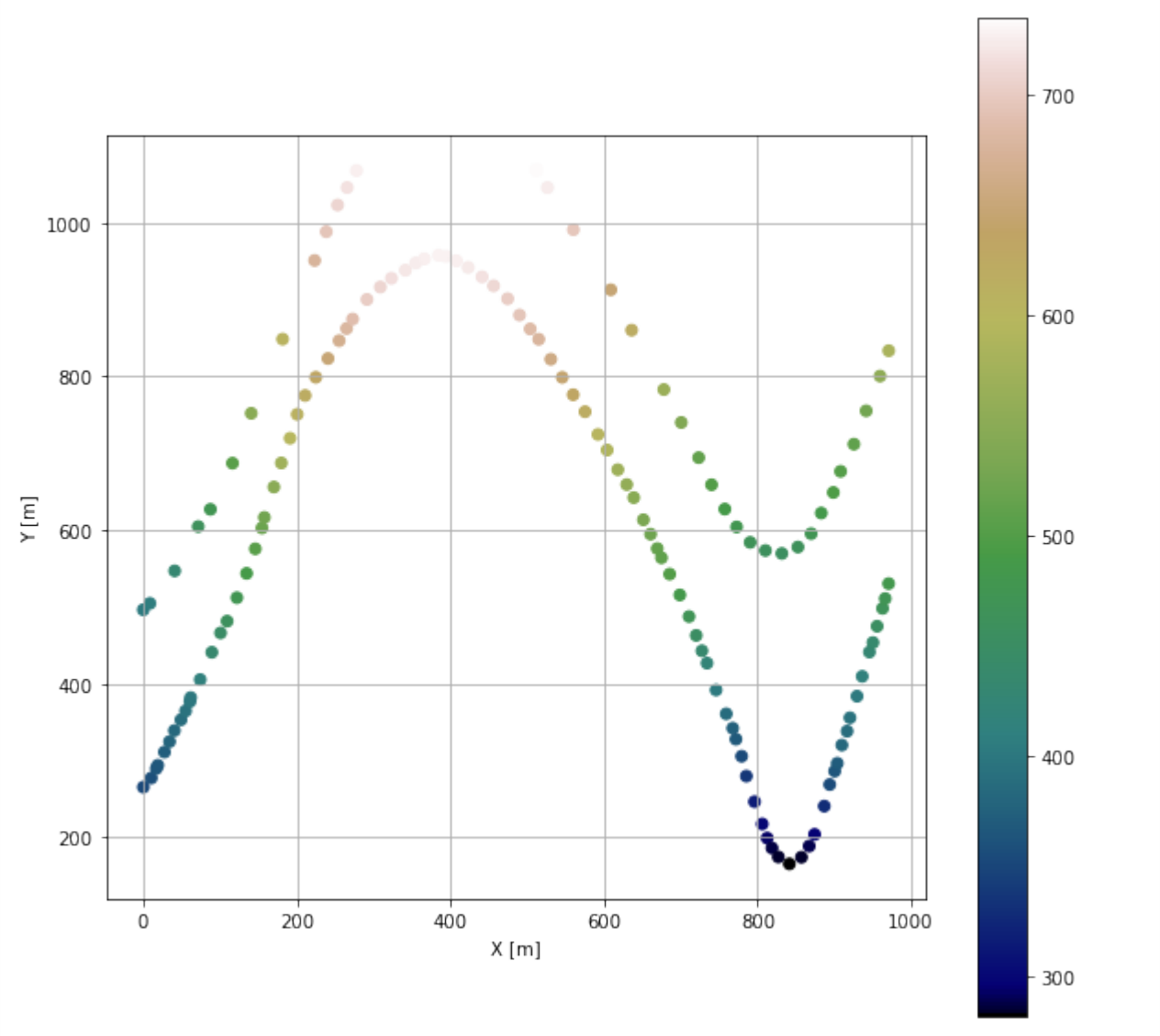
[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 972)
ax.set_ylim(0, 1069)
```

[13]:
```
(0.0, 1069.0)
```



[14]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

#interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='Z', legend=True, aspect='equal', cmap='gist_earth')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
# ax.set_xlim(0, 972)
# ax.set_ylim(0, 1069)
```

[14]:
```
Text(53.5, 0.5, 'Y [m]')
```



7.1.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

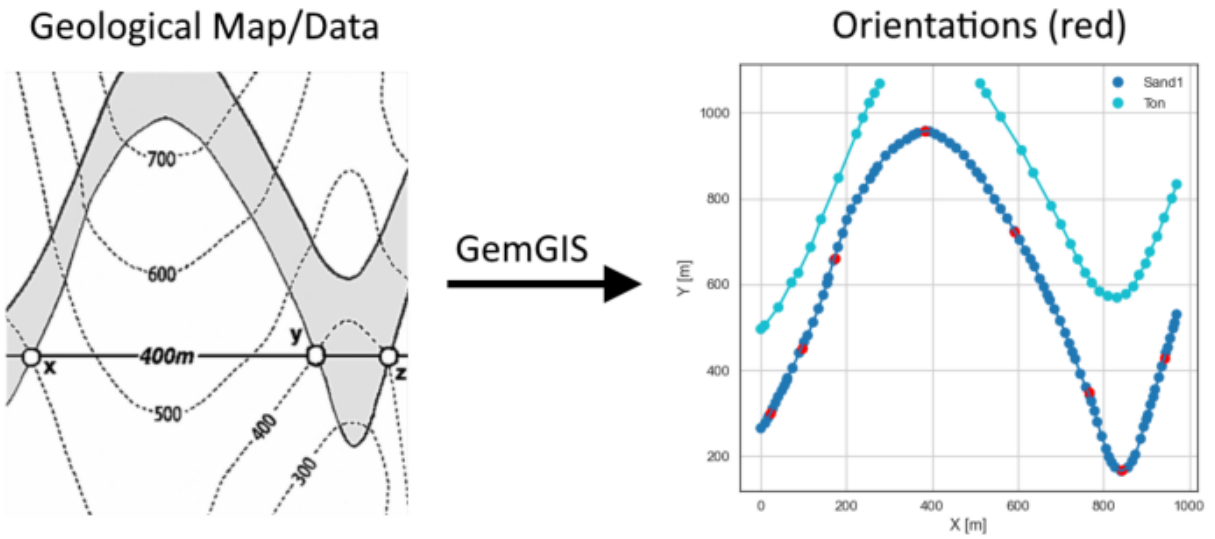
For this example, the orientations were calculated beforehand and will just be loaded into `GemPy`.

[15]:
```
img = mpimg.imread('../images/orientations_example1.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



[16]:
```
orientations = gpd.read_file(file_path + 'orientations1.shp')
orientations = gg.vector.extract_xyz(gdf=orientations, dem=topo_raster)
orientations['polarity'] = 1
orientations
```

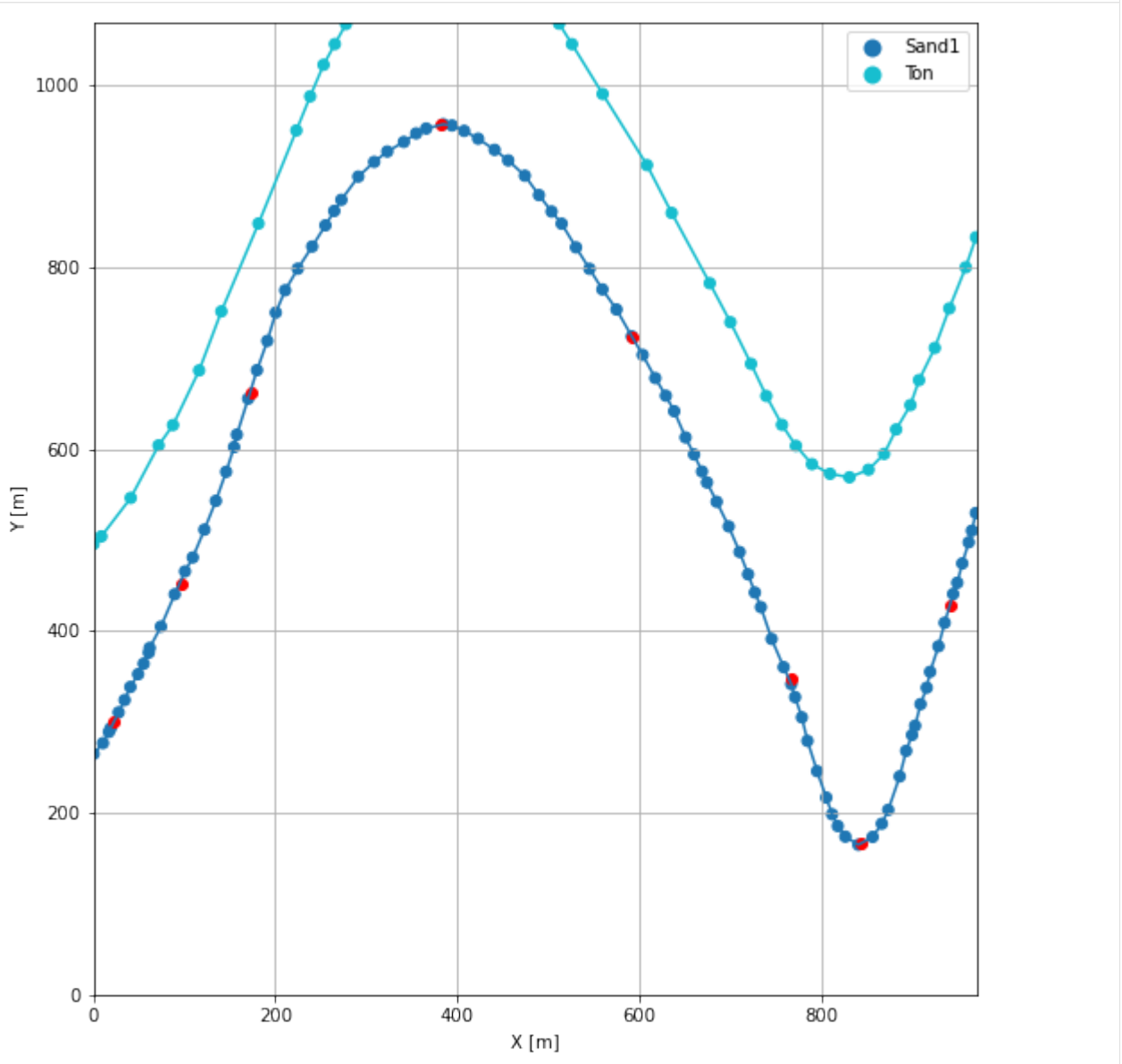
[16]:

	formation	dip	azimuth	geometry	X	Y	Z	polarity
0	Ton	30.50	180.00	POINT (96.471 451.564)	96.47	451.56	440.59	1
1	Ton	30.50	180.00	POINT (172.761 661.877)	172.76	661.88	556.38	1
2	Ton	30.50	180.00	POINT (383.074 957.758)	383.07	957.76	729.02	1
3	Ton	30.50	180.00	POINT (592.356 722.702)	592.36	722.70	601.55	1
4	Ton	30.50	180.00	POINT (766.586 348.469)	766.59	348.47	378.63	1
5	Ton	30.50	180.00	POINT (843.907 167.023)	843.91	167.02	282.61	1
6	Ton	30.50	180.00	POINT (941.846 428.883)	941.85	428.88	423.45	1
7	Ton	30.50	180.00	POINT (22.142 299.553)	22.14	299.55	368.05	1

Plotting the Orientations

[17]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 972)
ax.set_ylim(0, 1069)
```

[17]:
```
(0.0, 1069.0)
```



7.1.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[18]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[19]:
```
geo_model = gp.create_model('Model1')
geo_model
```

[19]:
```
Model1  2022-04-03 10:21
```

Initiate Data

[20]:
```
gp.init_data(geo_model, [0, 972, 0, 1069, 300, 800], [100, 100, 100],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[20]:
```
Model1  2022-04-03 10:21
```

Model Surfaces

[21]:
```
geo_model.surfaces
```

[21]:
```
  surface          series  order_surfaces    color  id
0   Sand1  Default series               1  #015482   1
1     Ton  Default series               2  #9f0052   2
```

Mapping the Stack to Surfaces

[22]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata': ('Sand1', 'Ton')},
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[22]:
```
    surface  series  order_surfaces    color  id
0     Sand1  Strata               1  #015482   1
1       Ton  Strata               2  #9f0052   2
2  Basement  Strata               3  #ffbe00   3
```

Showing the Number of Data Points

[23]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[23]:
```
    surface  series  order_surfaces    color  id  No. of Interfaces  No. of Orientations
0     Sand1  Strata               1  #015482   1                 95                    0
1       Ton  Strata               2  #9f0052   2                 36                    8
2  Basement  Strata               3  #ffbe00   3                  0                    0
```

Loading Digital Elevation Model

[24]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster1.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[24]:
```
Grid Object. Values:
array([[   4.86      ,    5.345     ,  302.5       ],
       [   4.86      ,    5.345     ,  307.5       ],
       [   4.86      ,    5.345     ,  312.5       ],
       ...,
       [ 970.056     , 1059.28181818,  622.0892334 ],
       [ 970.056     , 1063.16909091,  622.06713867],
       [ 970.056     , 1067.05636364,  622.05786133]])
```

Defining Custom Section

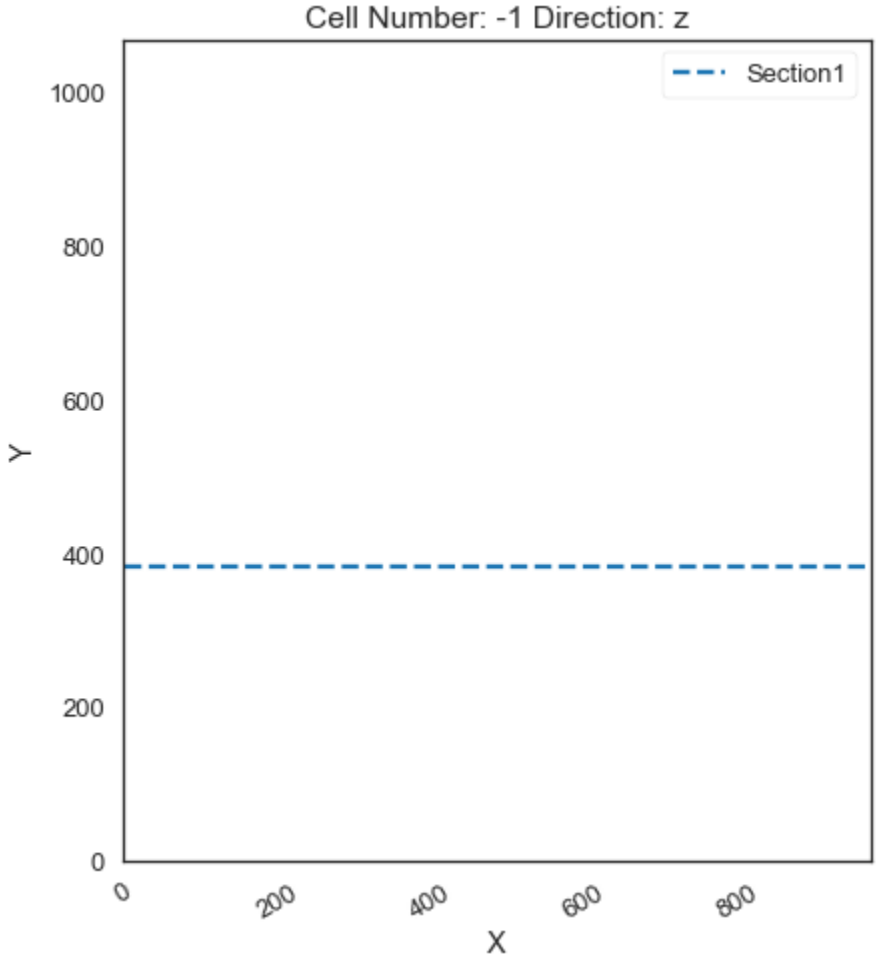
[25]:
```
custom_section = gpd.read_file(file_path + 'customsections1.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='section')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[25]:
```
                                          start                                       stop␣
↪resolution   dist
Section1  [1.372395262185787, 383.9794474025771]  [970.9954955186289, 383.8831909730347]␣
↪ [100, 80] 969.62
```

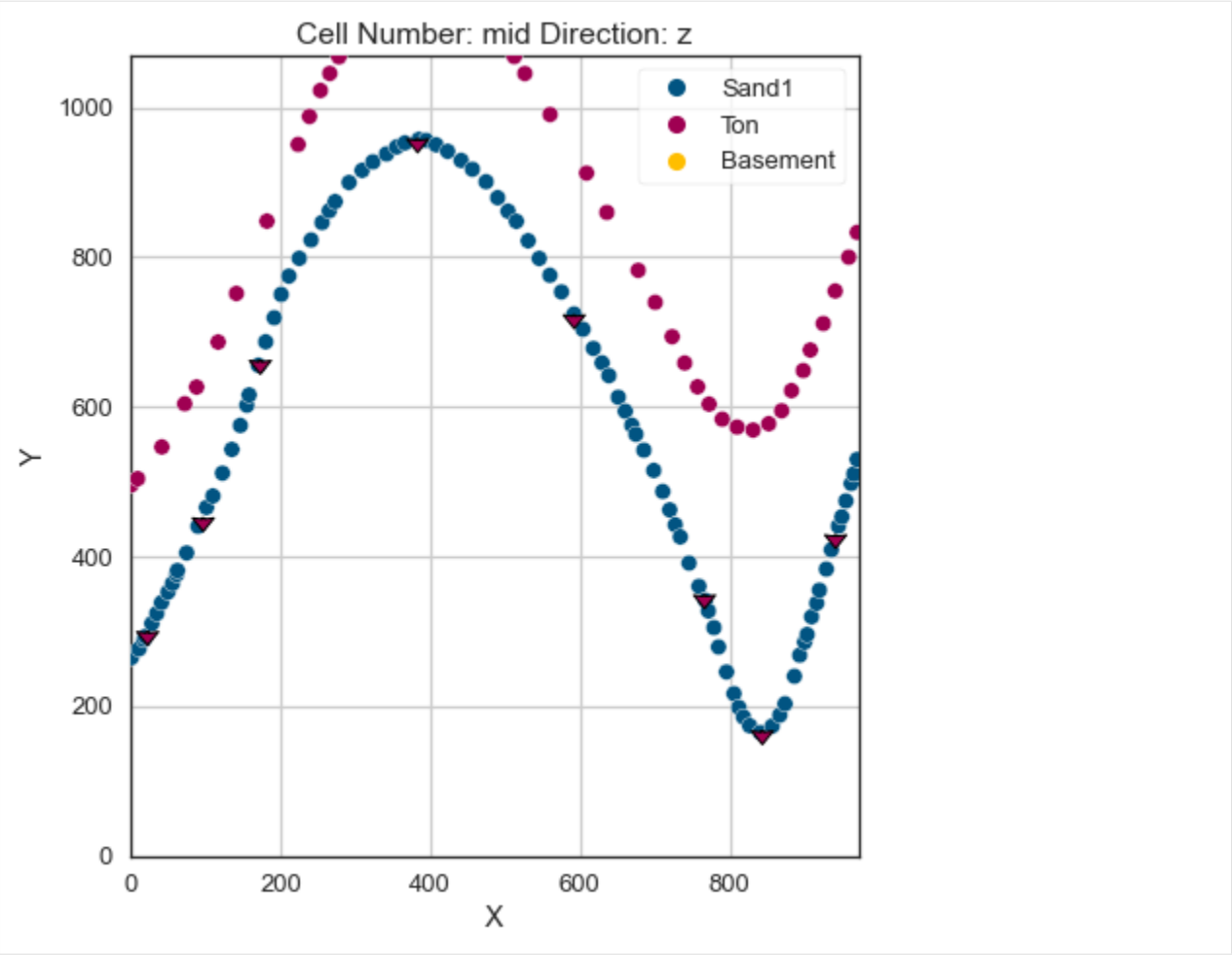
[26]:
```
gp.plot.plot_section_traces(geo_model)
```

[26]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1bbcdf9d850>
```

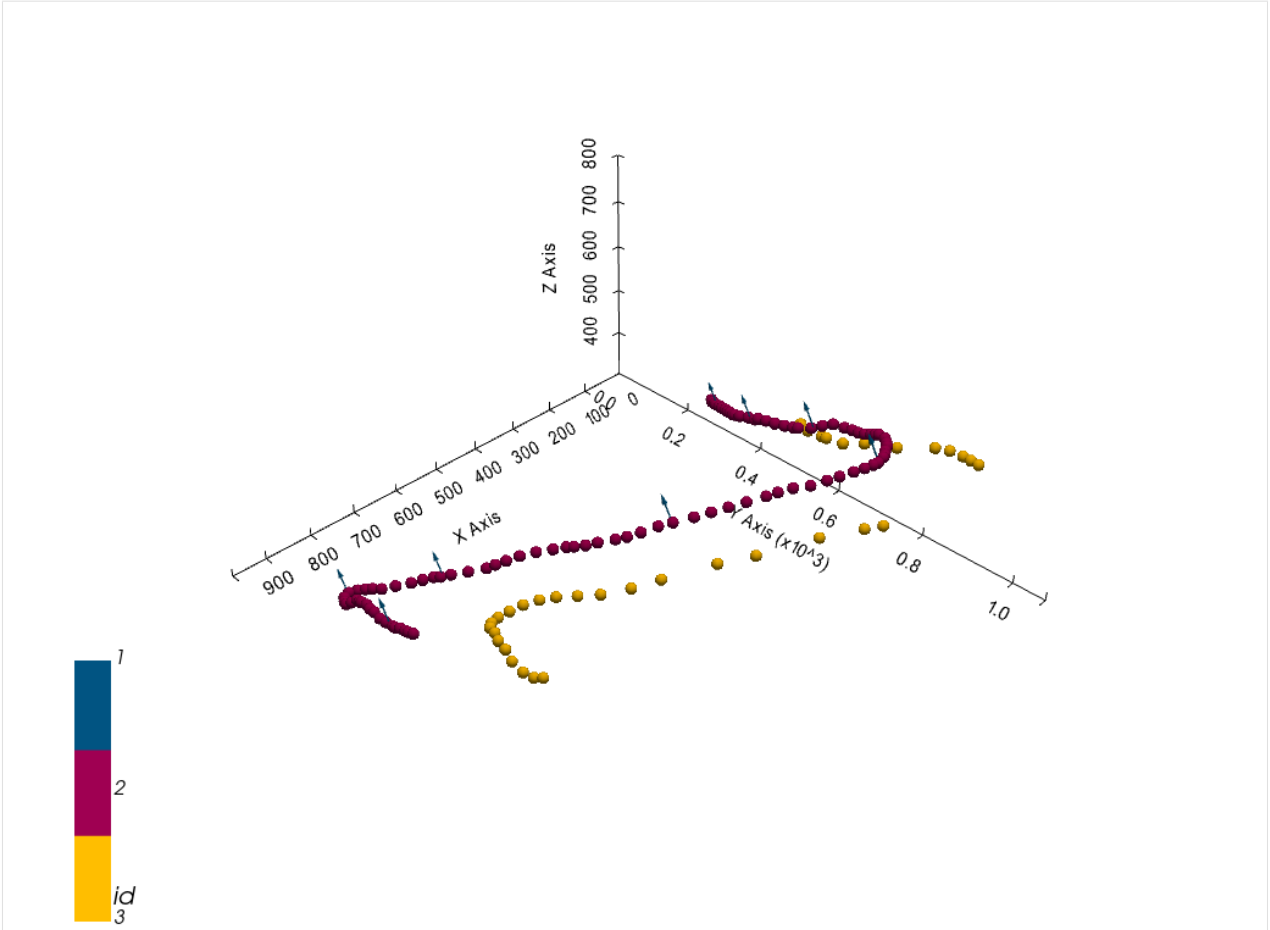


Plotting Input Data

[27]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[28]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[28]:
```
<gempy.plot.vista.GemPyToVista at 0x1bbd8884d90>
```

Setting the Interpolator

[29]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range             1528.9
$C_o$            55655.83
drift equations      [3]
```

[29]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1bbd7d8b7f0>
```

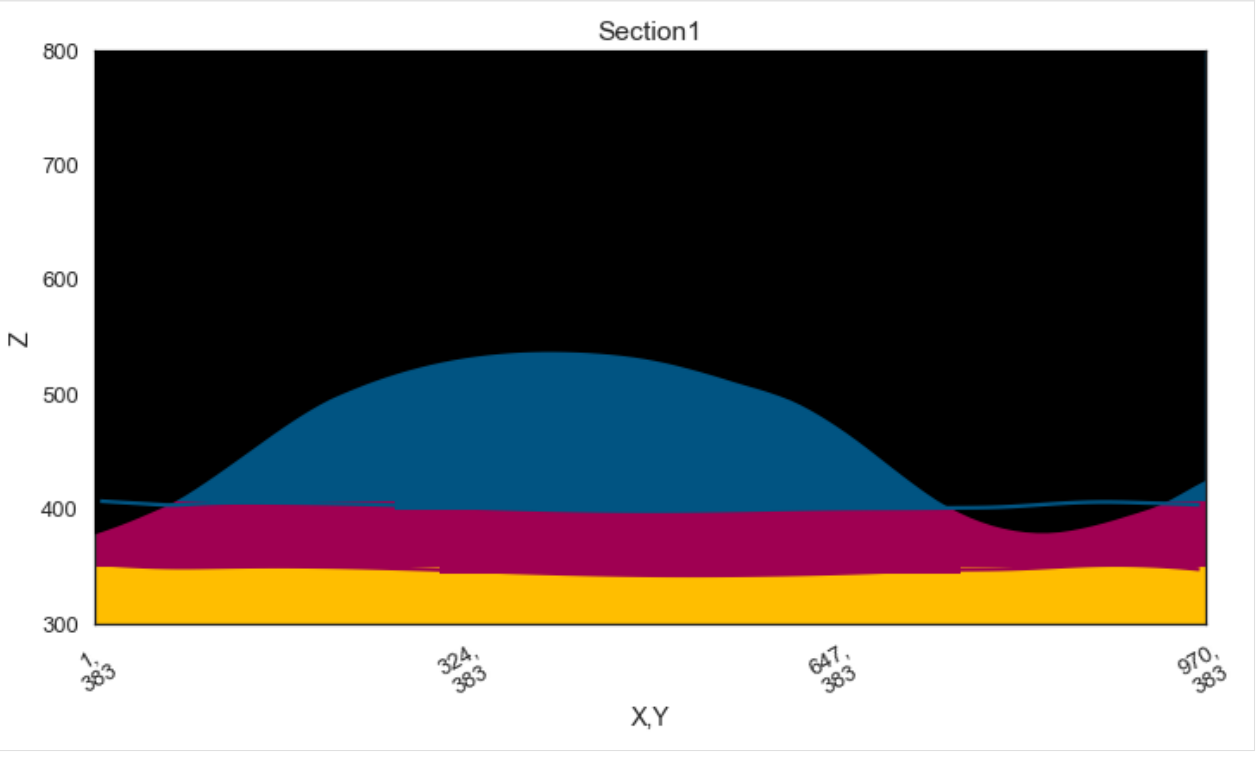
Computing Model

[30]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

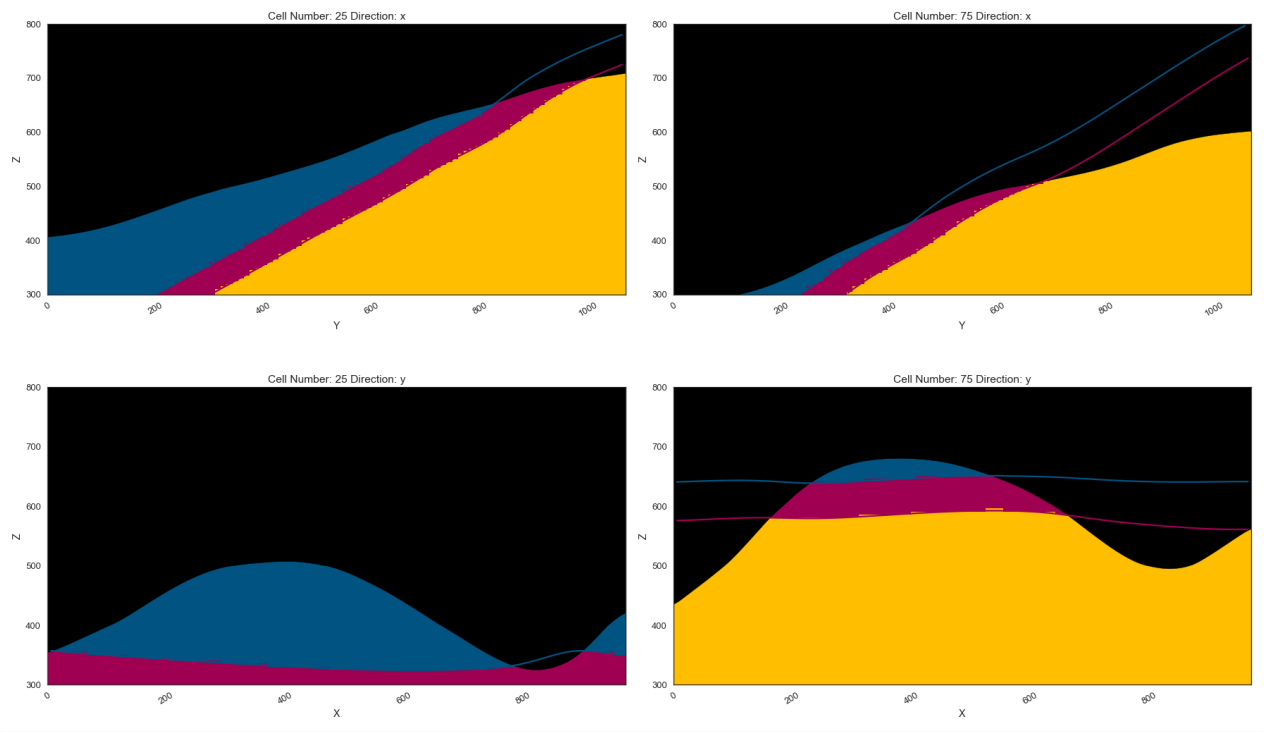
[31]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[31]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1bbdda5b340>
```

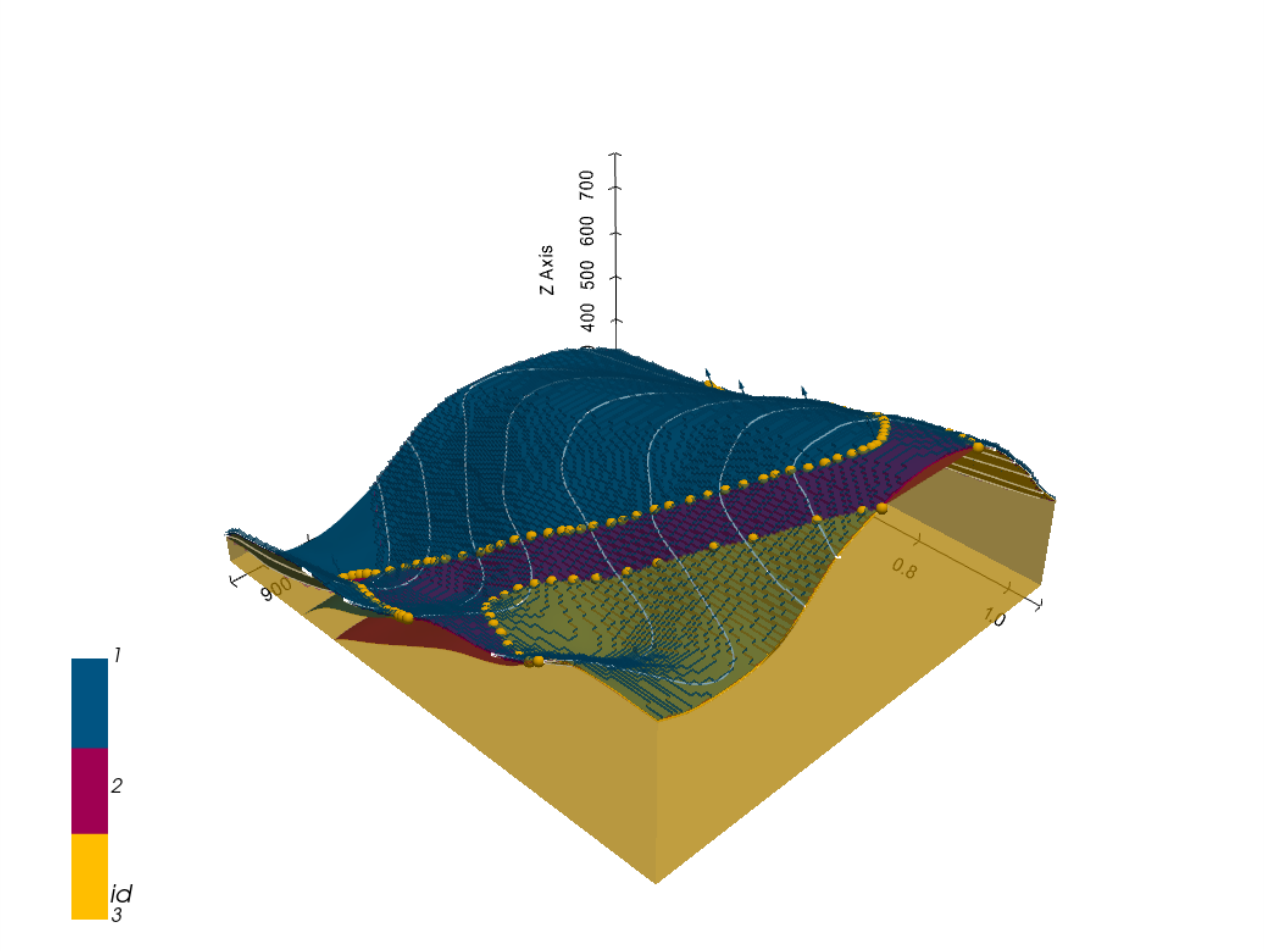


[32]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[32]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1bbefba4970>
```



[33]:
```
gpv = gp.plot_3d(geo_model,
                 image=False,
                 show_topography=True,
                 plotter_type='basic',
                 notebook=True,
                 show_lith=True,
                 show_boundaries=True)
```



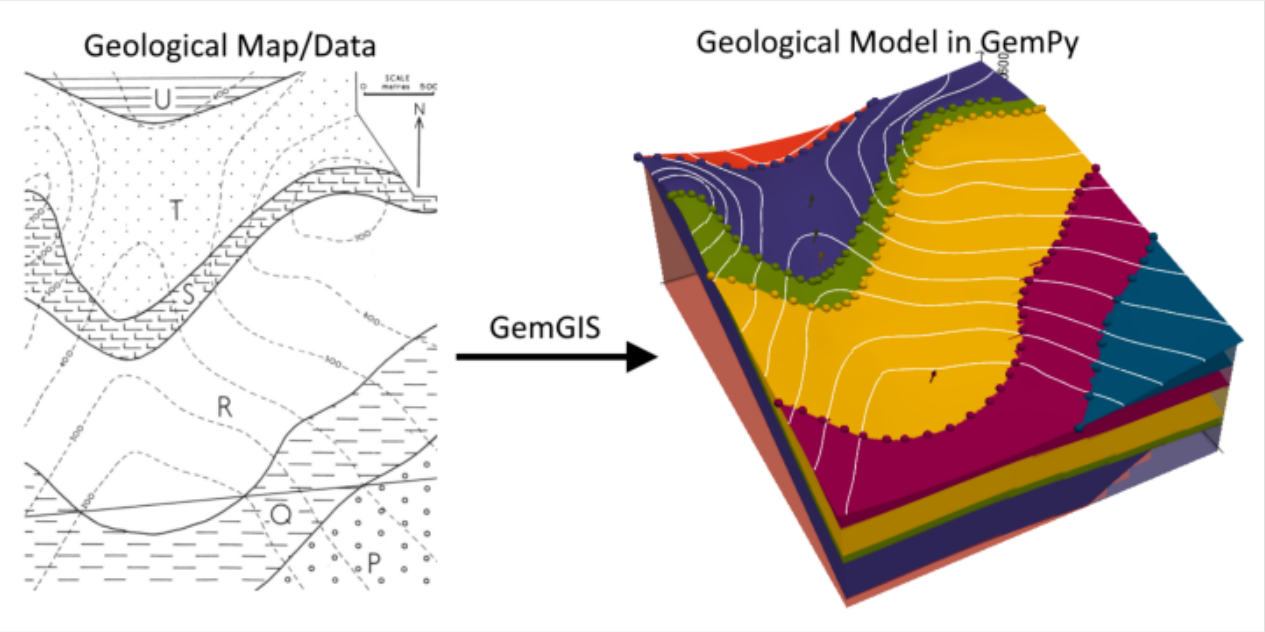
[ ]:

7.2 Example 2 - Planar Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 2932 m wide (W-E extent) and 3677 m high (N-S extent). The vertical model extent varies between -700 m and 1000 m. The model represents several planar stratigraphic units (blue to purple) dipping towards the south above an unspecified basement (light red). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example02.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.2.1 Licensing

7.2.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.2.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

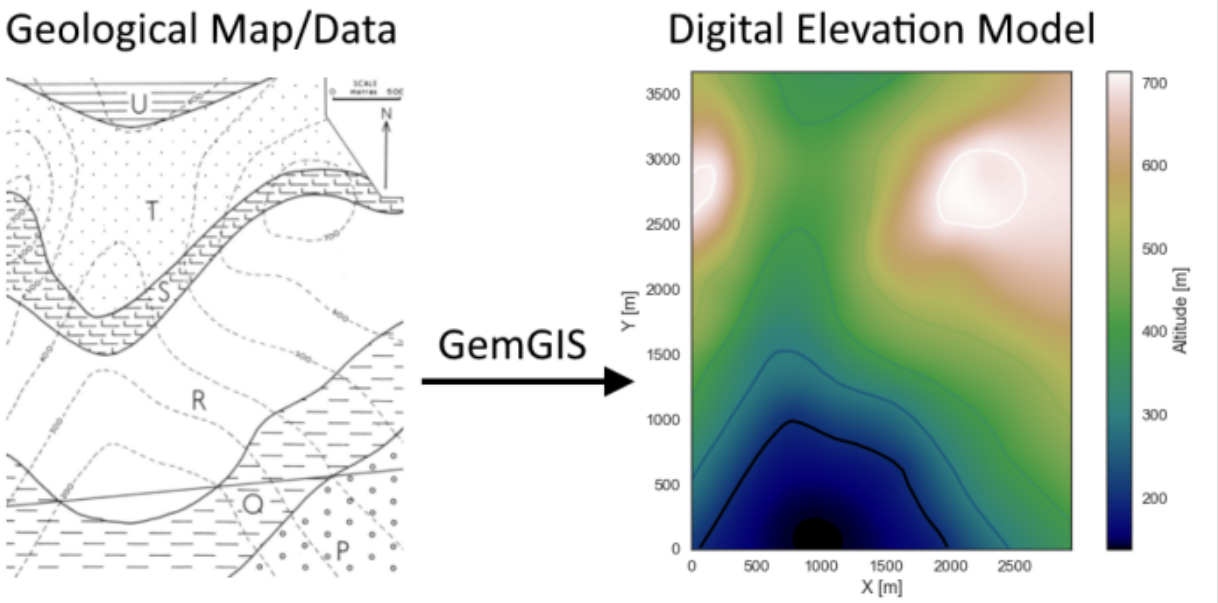
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = '../data/example02/'
gg.download_gemgis_data.download_tutorial_data(filename="example02_planar_dipping_layers.
↪zip", dirpath=file_path)
```

7.2.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example2.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo2.shp')
topo.head()
```

[6]:
```
     id    Z                                           geometry
0  None  200  LINESTRING (66.248 9.085, 187.630 201.563, 284...
1  None  300  LINESTRING (2.089 534.498, 109.599 713.104, 22...
2  None  400  LINESTRING (5.557 1167.421, 69.716 1294.006, 1...
3  None  700  LINESTRING (5.557 2894.521, 59.312 2939.606, 1...
4  None  600  LINESTRING (7.291 3267.338, 69.716 3288.147, 1...
```

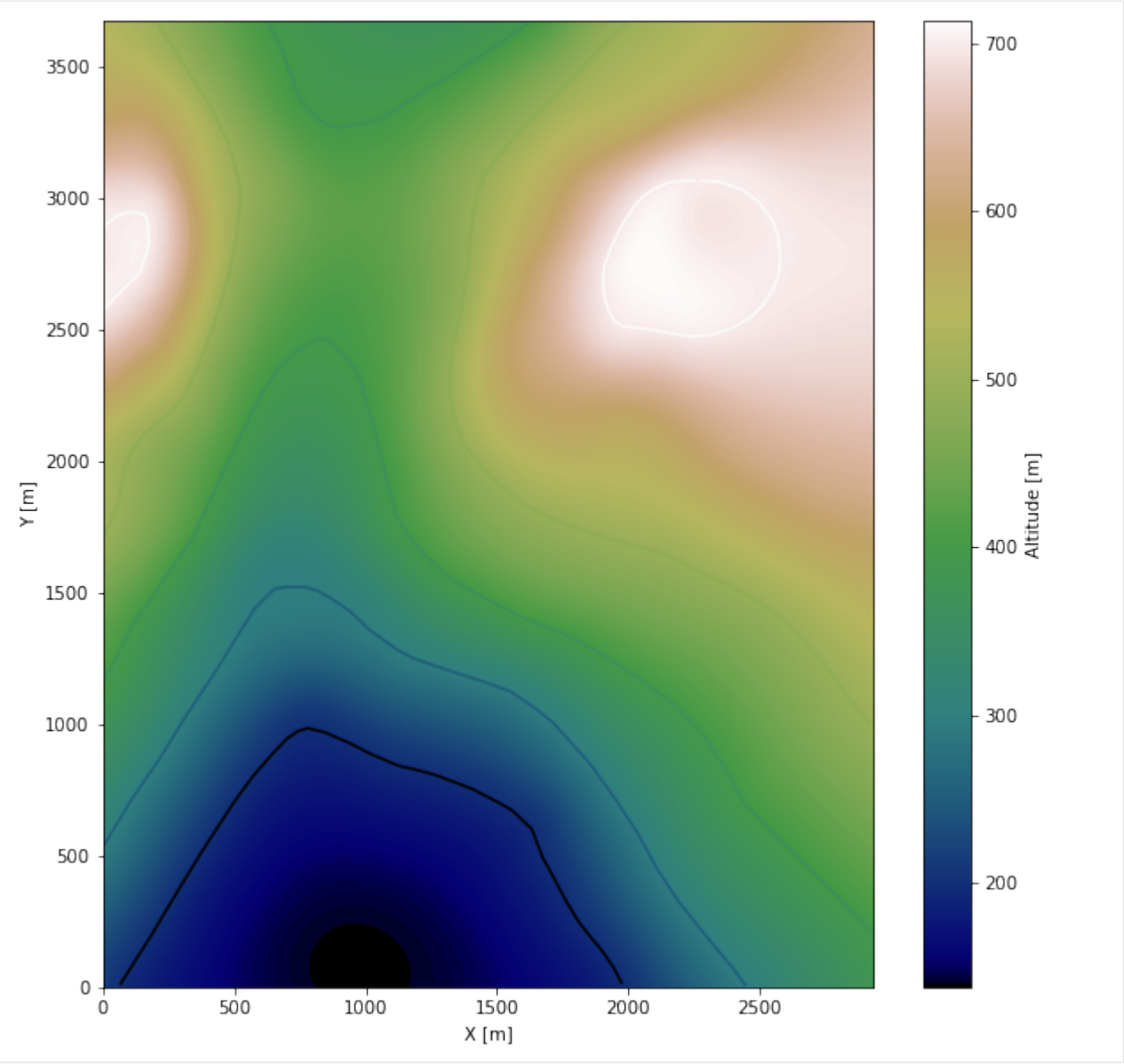
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 2932, 0, 3677], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2932)
ax.set_ylim(0, 3677)
```

[8]:
```
(0.0, 3677.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster2.tif', extent = [0, 2932, 0, 3677], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

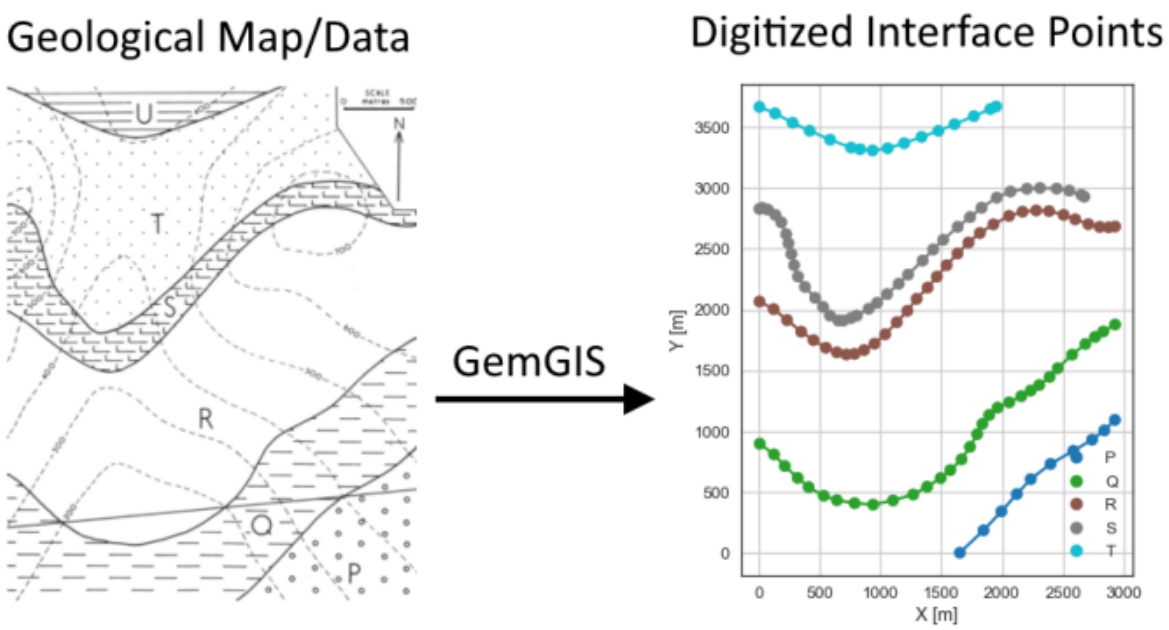
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster2.tif')
```

7.2.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example2.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces2.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	P	LINESTRING (1652.891 2.149, 1847.103 185.957, ...
1	None	Q	LINESTRING (7.291 898.645, 125.205 810.210, 21...
2	None	R	LINESTRING (5.557 2067.386, 121.737 2004.960, ...
3	None	S	LINESTRING (4.690 2829.494, 27.232 2837.298, 7...
4	None	T	LINESTRING (4.690 3667.901, 133.875 3615.013, ...

Extracting Z coordinate from Digital Elevation Model

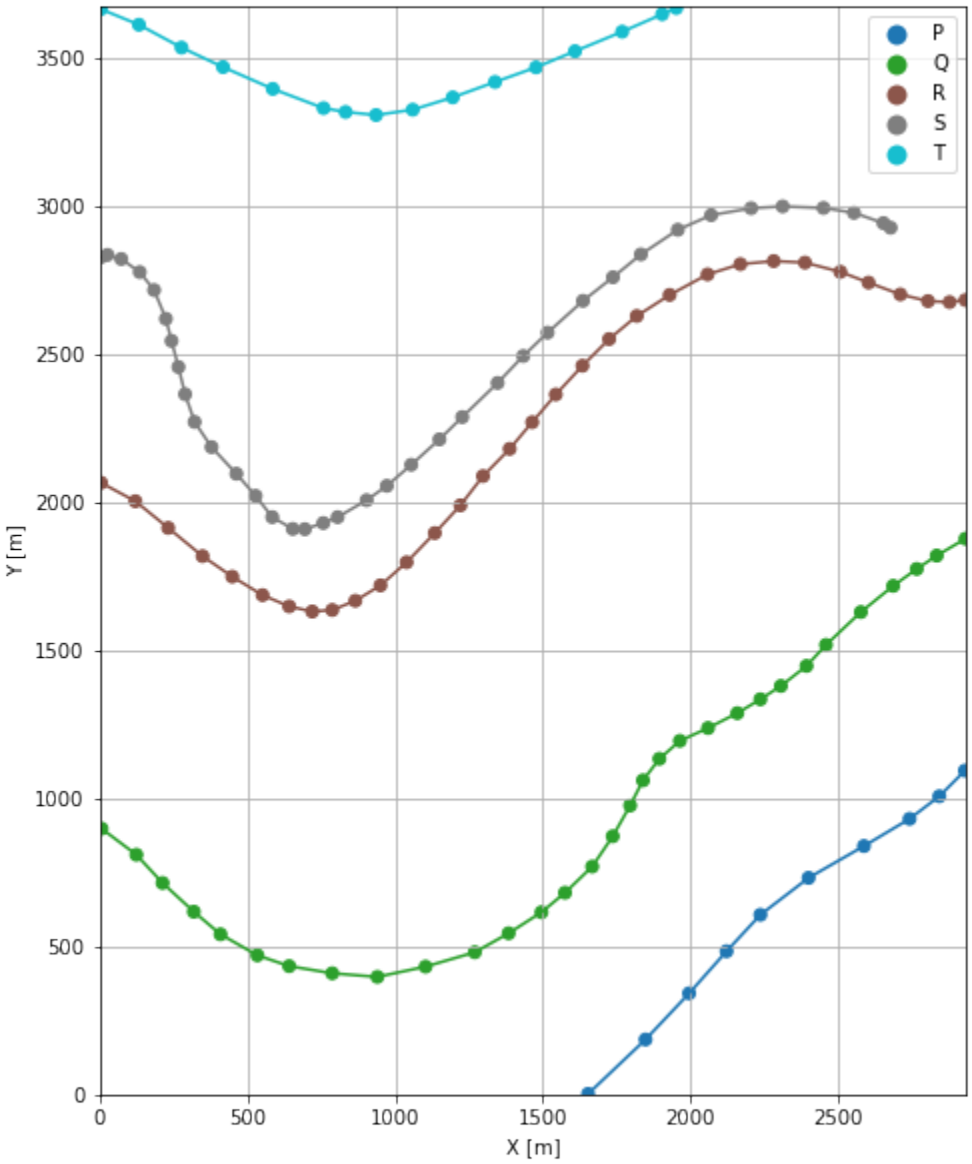
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords
```

[12]:
```
    formation                    geometry       X       Y      Z
0           P     POINT (1652.891 2.149) 1652.89    2.15 162.71
1           P   POINT (1847.103 185.957) 1847.10  185.96 196.79
2           P   POINT (1994.496 342.020) 1994.50  342.02 252.93
3           P   POINT (2121.080 484.211) 2121.08  484.21 305.56
4           P   POINT (2235.527 607.327) 2235.53  607.33 349.55
..        ...                        ...     ...     ...    ...
121         T  POINT (1476.886 3471.088) 1476.89 3471.09 421.32
122         T  POINT (1608.673 3525.710) 1608.67 3525.71 436.16
123         T  POINT (1768.204 3591.603) 1768.20 3591.60 459.65
124         T  POINT (1903.459 3650.560) 1903.46 3650.56 486.01
125         T  POINT (1951.145 3671.369) 1951.14 3671.37 490.75

[126 rows x 5 columns]
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2932)
ax.set_ylim(0, 3677)
```

[13]:
```
(0.0, 3677.0)
```



7.2.6 Orientations from Strike Lines

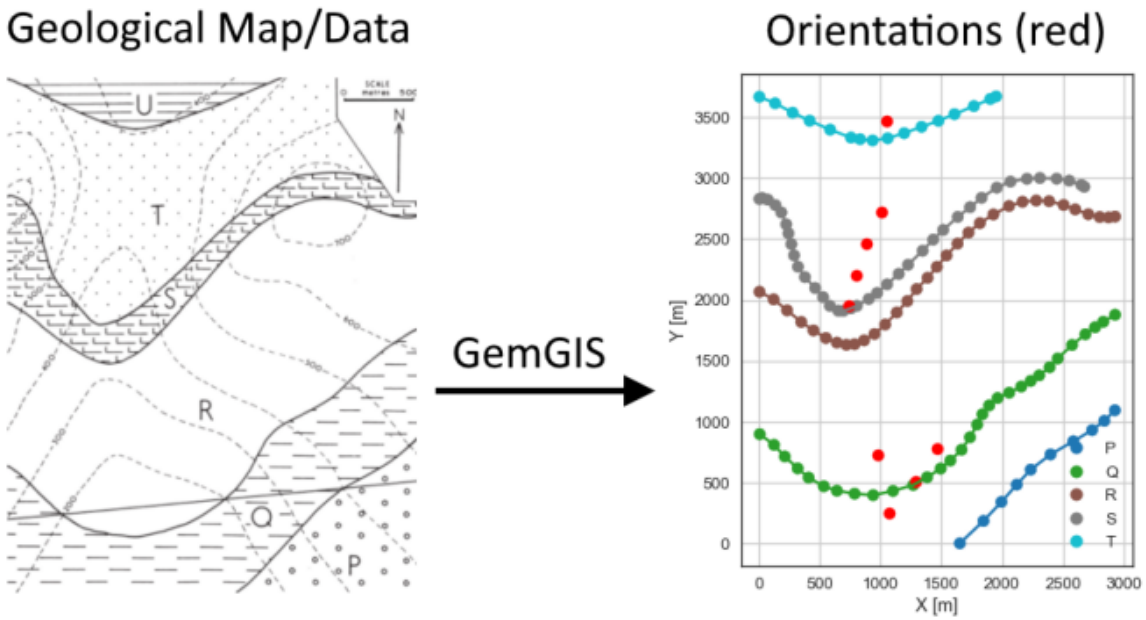
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/orientations_example2.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes2.shp')
strikes
```

[15]:

	id	formation	Z	geometry
0	1	Q	200	LINESTRING (399.616 550.104, 1562.287 669.319)
1	2	Q	300	LINESTRING (155.551 778.563, 1774.273 938.095)
2	1	R	400	LINESTRING (383.576 1797.309, 1101.467 1861.469)
3	2	R	500	LINESTRING (99.195 2015.798, 1347.700 2138.915)
4	1	S	400	LINESTRING (503.225 2054.380, 1035.140 2108.569)
5	2	S	500	LINESTRING (322.018 2272.002, 1329.493 2383.847)
6	3	S	600	LINESTRING (249.189 2524.304, 1626.013 2671.697)
7	4	S	700	LINESTRING (155.551 2755.798, 2006.634 2942.207)
8	1	T	400	LINESTRING (798.965 3326.642, 1129.884 3348.947)
9	2	T	500	LINESTRING (296.896 3529.200, 1983.810 3693.066)
10	4	P	500	LINESTRING (2881.434 1048.444, 154.424 778.065)
11	3	P	400	LINESTRING (2417.145 743.471, 399.313 550.928)
12	2	P	300	LINESTRING (2101.703 460.346, 239.088 291.017)
13	1	P	200	LINESTRING (1854.082 194.518, 87.967 44.308)

Calculate Orientations for each formation

[16]:
```
orientations_p = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'P'].sort_values(by='id', ascending=True).reset_index())
orientations_p
```

[16]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 23.23   174.96 250.00  POINT (1070.710 247.547)      1.00         P 1070.71
1 22.26   174.67 350.00  POINT (1289.312 511.441)      1.00         P 1289.31
2 21.79   174.41 450.00  POINT (1463.079 780.227)      1.00         P 1463.08

       Y
0 247.55
1 511.44
2 780.23
```

[17]:
```
orientations_q = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'Q'].reset_index())
orientations_q
```

[17]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 22.07   174.29 250.00  POINT (972.932 734.020)      1.00         Q 972.93

       Y
0 734.02
```

[18]:
```
orientations_r = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'R'].reset_index())
orientations_r
```

[18]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 22.18   174.50 450.00  POINT (732.985 1953.373)      1.00         R 732.98

        Y
0 1953.37
```

[19]:
```
orientations_s = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'S'].reset_index())
orientations_s
```

[19]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.94   173.78 450.00   POINT (797.469 2204.700)      1.00         S
1 21.44   173.81 550.00   POINT (881.678 2462.963)      1.00         S
2 23.41   174.12 650.00  POINT (1009.347 2723.501)      1.00         S

        X       Y
0  797.47 2204.70
1  881.68 2462.96
2 1009.35 2723.50
```

[20]:
```
orientations_t = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'T'].reset_index())
orientations_t
```

[20]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 21.79   174.52 450.00  POINT (1052.389 3474.464)      1.00         T

        X       Y
0 1052.39 3474.46
```

Merging Orientations

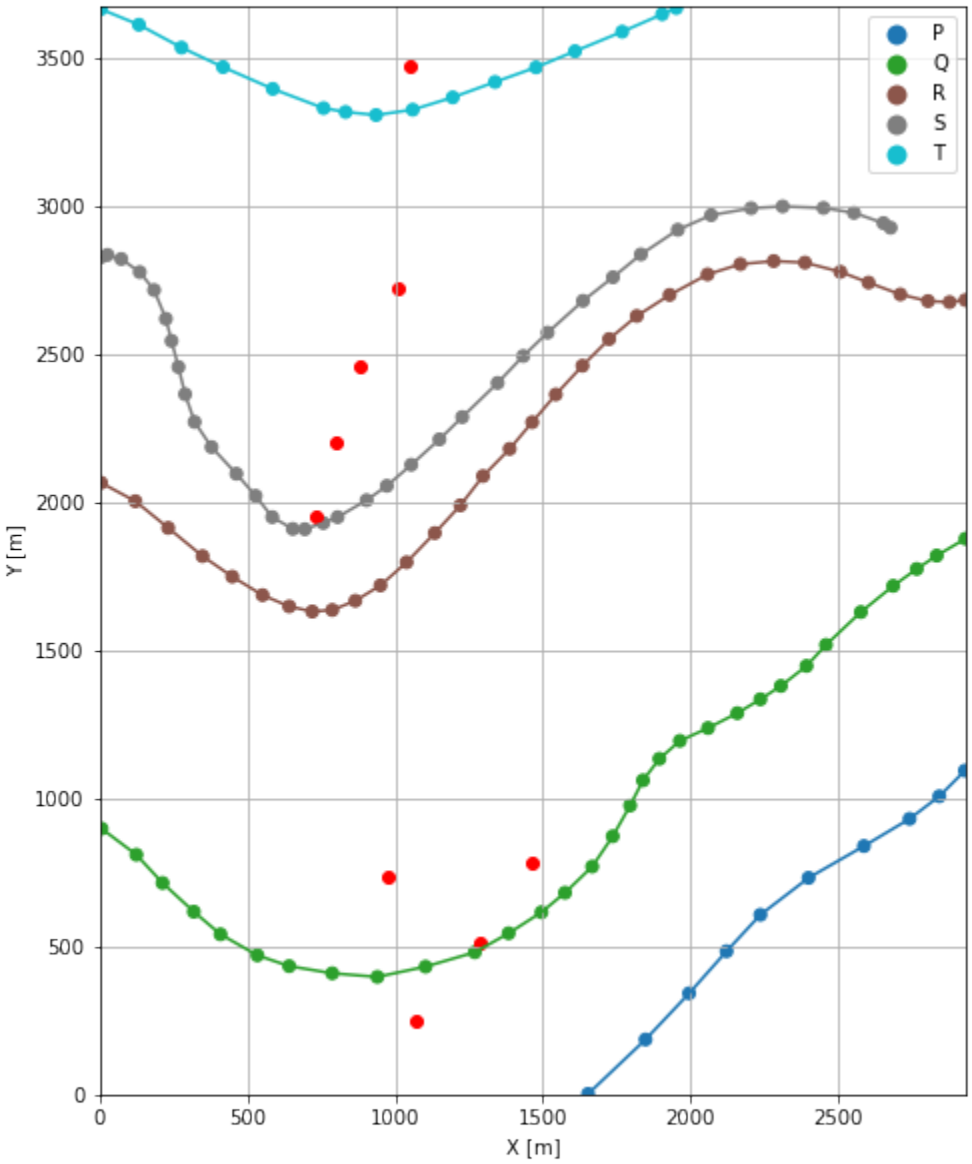
[21]:
```
import pandas as pd
orientations = pd.concat([orientations_p, orientations_q, orientations_r, orientations_s,
↪ orientations_t]).reset_index()
orientations
```

[21]:
```
   index   dip  azimuth      Z                   geometry  polarity formation  \
0      0 23.23   174.96 250.00   POINT (1070.710 247.547)      1.00         P
1      1 22.26   174.67 350.00   POINT (1289.312 511.441)      1.00         P
2      2 21.79   174.41 450.00   POINT (1463.079 780.227)      1.00         P
3      0 22.07   174.29 250.00    POINT (972.932 734.020)      1.00         Q
4      0 22.18   174.50 450.00   POINT (732.985 1953.373)      1.00         R
5      0 22.94   173.78 450.00   POINT (797.469 2204.700)      1.00         S
6      1 21.44   173.81 550.00   POINT (881.678 2462.963)      1.00         S
7      2 23.41   174.12 650.00  POINT (1009.347 2723.501)      1.00         S
8      0 21.79   174.52 450.00  POINT (1052.389 3474.464)      1.00         T

        X       Y
0 1070.71  247.55
1 1289.31  511.44
2 1463.08  780.23
3  972.93  734.02
4  732.98 1953.37
5  797.47 2204.70
6  881.68 2462.96
7 1009.35 2723.50
8 1052.39 3474.46
```

Plotting the Orientations

[22]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2932)
ax.set_ylim(0, 3677)
```

[22]:
```
(0.0, 3677.0)
```



7.2.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[23]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[24]:
```
geo_model = gp.create_model('Model2')
geo_model
```

[24]:
```
Model2  2022-04-03 21:47
```

Initiate Data

[25]:
```
gp.init_data(geo_model, [0, 2932, 0, 3677, -700, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[25]:
```
Model2  2022-04-03 21:47
```

Model Surfaces

[26]:
```
geo_model.surfaces
```

[26]:

	surface	series	order_surfaces	color	id
0	P	Default series	1	#015482	1
1	Q	Default series	2	#9f0052	2
2	R	Default series	3	#ffbe00	3
3	S	Default series	4	#728f02	4
4	T	Default series	5	#443988	5

Mapping the Stack to Surfaces

[27]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata': ('P', 'Q', 'R', 'S', 'T')},
                         remove_unused_series=True)
geo_model.add_surfaces('U')
```

[27]:

	surface	series	order_surfaces	color	id
0	P	Strata	1	#015482	1
1	Q	Strata	2	#9f0052	2
2	R	Strata	3	#ffbe00	3
3	S	Strata	4	#728f02	4
4	T	Strata	5	#443988	5
5	U	Strata	6	#ff3f20	6

Showing the Number of Data Points

[28]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[28]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	P	Strata	1	#015482	1	10	3
1	Q	Strata	2	#9f0052	2	31	1
2	R	Strata	3	#ffbe00	3	32	1
3	S	Strata	4	#728f02	4	37	3
4	T	Strata	5	#443988	5	16	1
5	U	Strata	6	#ff3f20	6	0	0

Loading Digital Elevation Model

[29]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster2.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[29]:
```
Grid Object. Values:
array([[  14.66      ,   18.385     , -691.5       ],
       [  14.66      ,   18.385     , -674.5       ],
       [  14.66      ,   18.385     , -657.5       ],
       ...,
       [2926.99658703, 3652.02038043,  629.80548096],
       [2926.99658703, 3662.01222826,  629.07196045],
       [2926.99658703, 3672.00407609,  628.35705566]])
```

Defining Custom Section

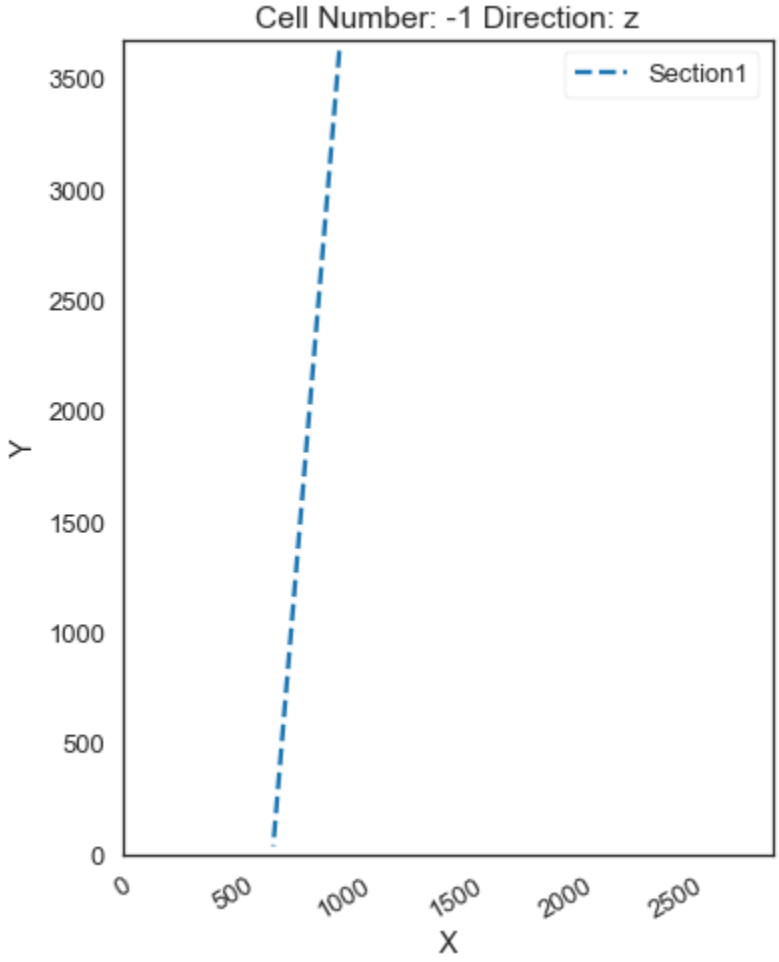
[30]:
```
custom_section = gpd.read_file(file_path + 'customsection2.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='name')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[30]:
```
                                          start                                                ␣
↪stop resolution    dist
Section1  [974.6659586819704, 3627.5193764495925]  [676.0649712912196, 38.
↪845314576599776]  [100, 80] 3601.08
```

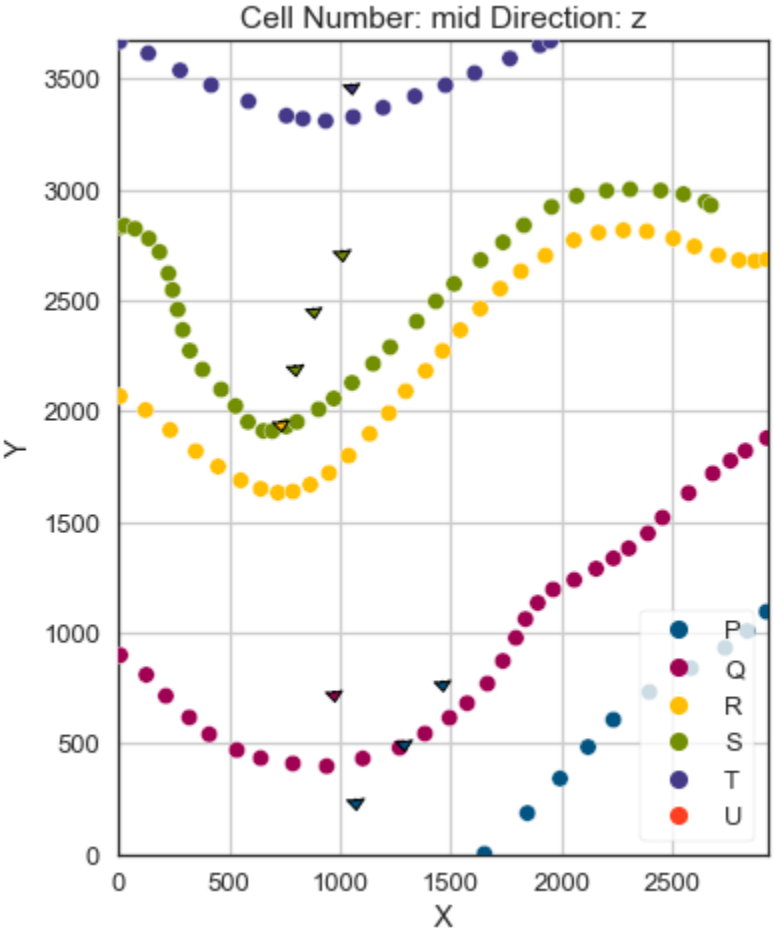
[31]:
```
gp.plot.plot_section_traces(geo_model)
```

[31]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1c7084f8250>
```

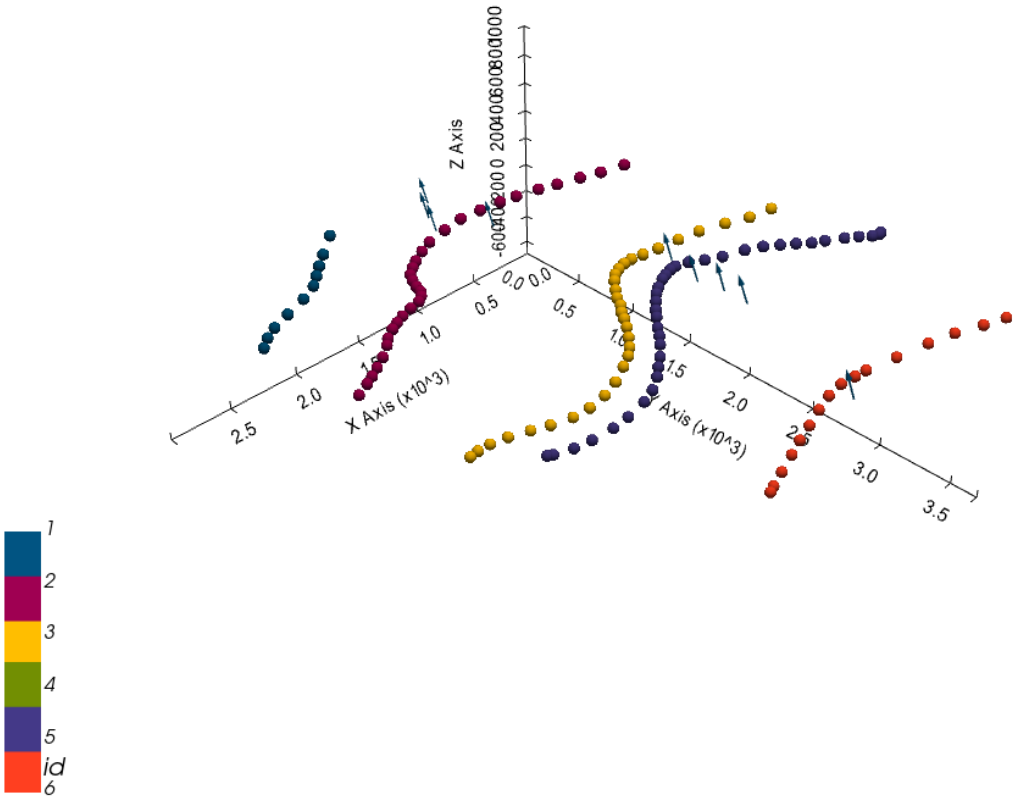


Plotting Input Data

[32]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[33]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[33]:
```
<gempy.plot.vista.GemPyToVista at 0x1c706daf3d0>
```

Setting the Interpolator

[34]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              5000.7
$C_o$           595403.64
drift equations       [3]
```

[34]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1c705297ee0>
```

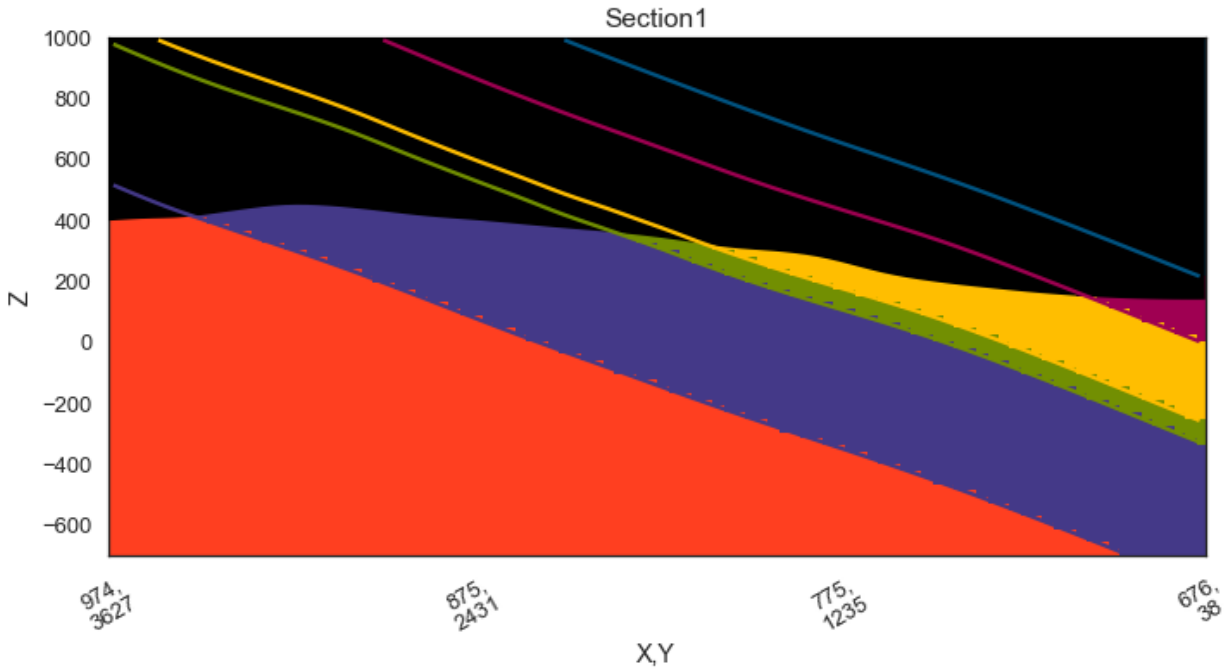
Computing Model

[35]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

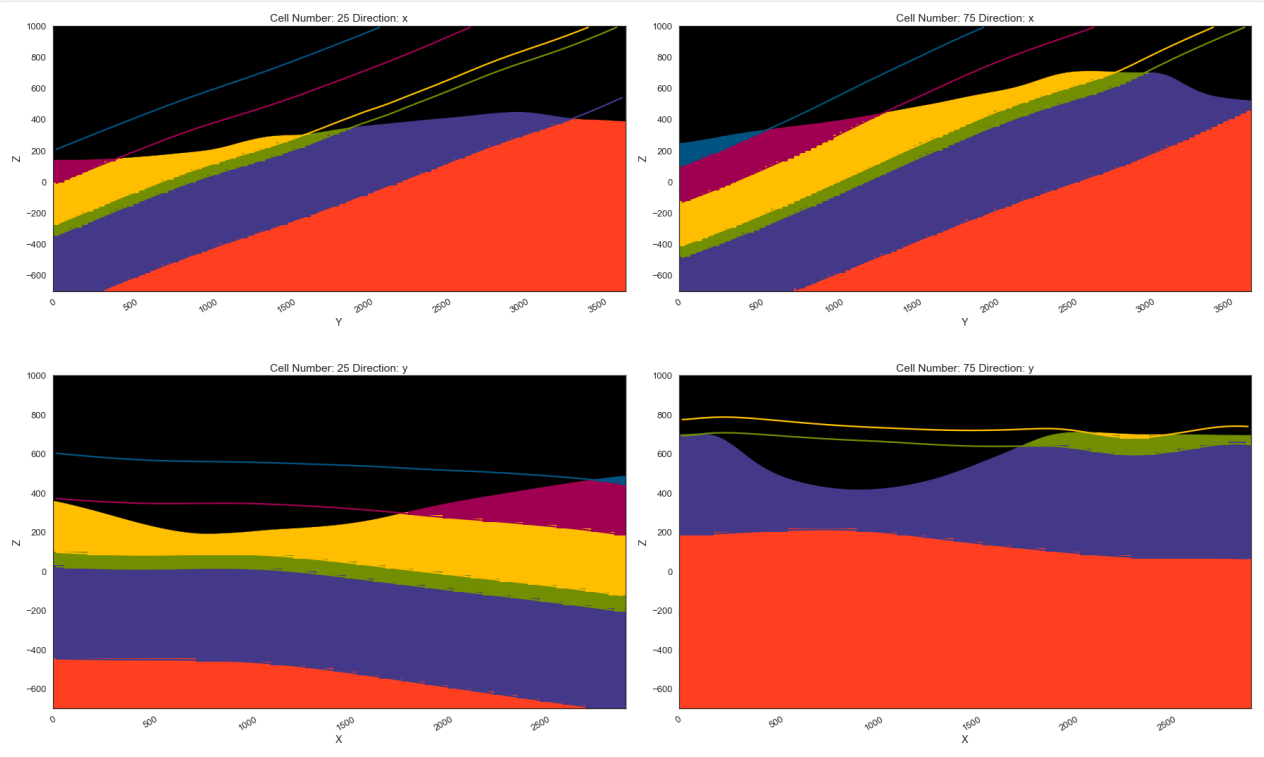
[36]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[36]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1c709a2f430>
```



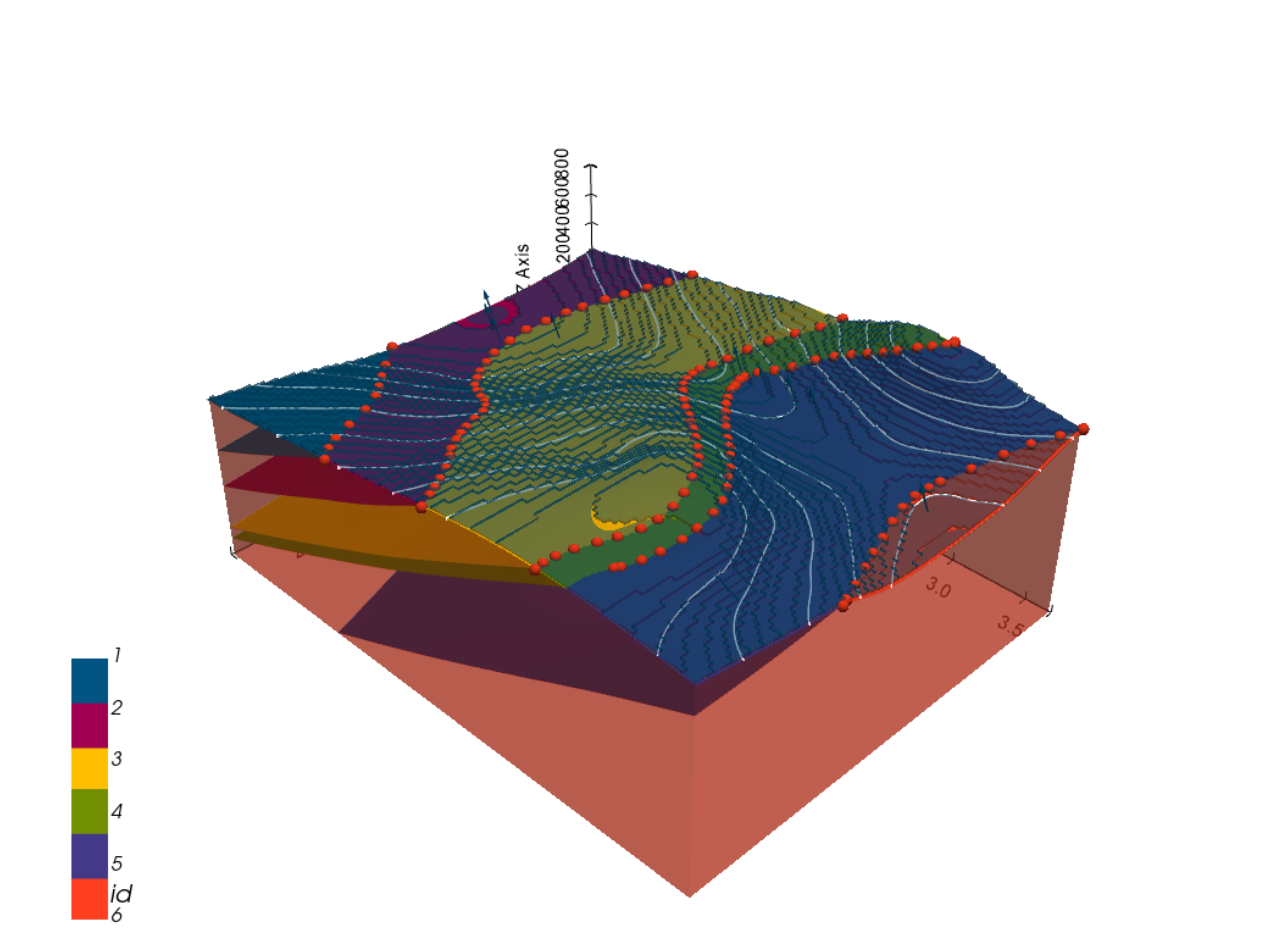
[37]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[37]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1c709a2f5b0>
```



Plotting 3D Model

[38]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



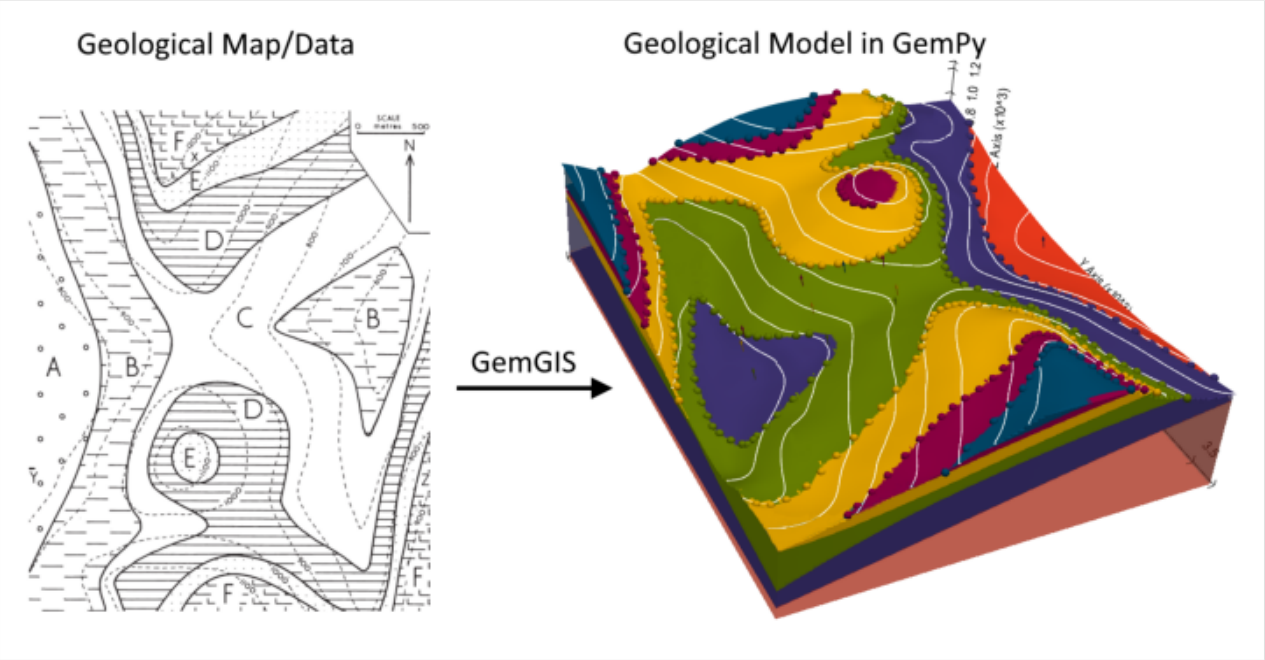
[ ]:

7.3 Example 3 - Planar Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3000 m wide (W-E extent) and 3740 m high (N-S extent). The vertical model extent varies between 200 m and 1200 m. The model represents several planar stratigraphic units (blue to purple) dipping towards the east above an unspecified basement (light red). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example03.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.3.1 Licensing

7.3.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.3.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example03/'
gg.download_gemgis_data.download_tutorial_data(filename="example03_planar_dipping_layers.
↪zip", dirpath=file_path)
```

(continues on next page)

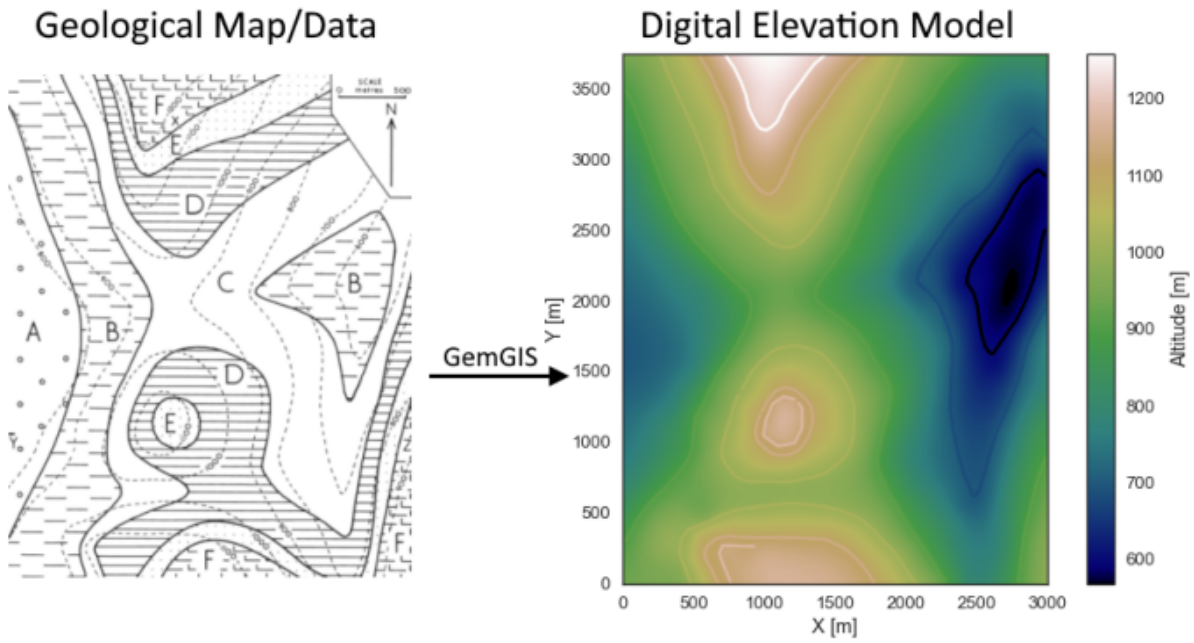
(continued from previous page)

```
Downloading file 'example03_planar_dipping_layers.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2Fexample03_planar_dipping_layers.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\example\data\example03'.
```

7.3.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example03.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo3.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	800	LINESTRING (6.948 674.298, 97.151 781.847, 168...
1	None	900	LINESTRING (10.418 291.517, 78.648 414.099, 12...
2	None	1100	LINESTRING (978.357 1207.417, 1009.581 1271.02...
3	None	1000	LINESTRING (834.959 1363.536, 882.373 1447.956...
4	None	1100	LINESTRING (1778.613 2.407, 1712.696 100.704, ...

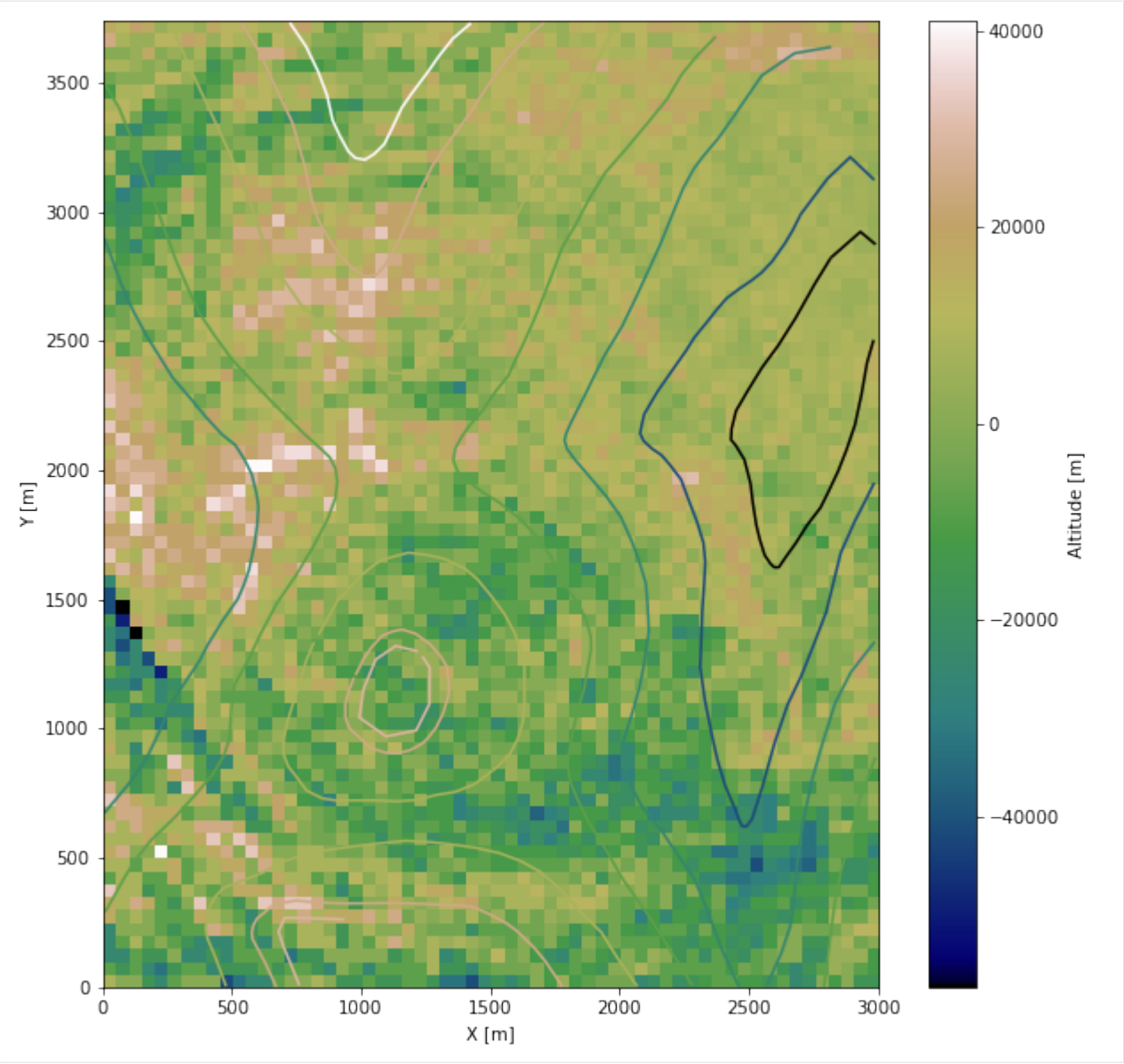
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=50)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 3000, 0, 3740], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3000)
ax.set_ylim(0, 3740)
```

[8]:
```
(0.0, 3740.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster3.tif', extent = [0, 3000, 0, 3740], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

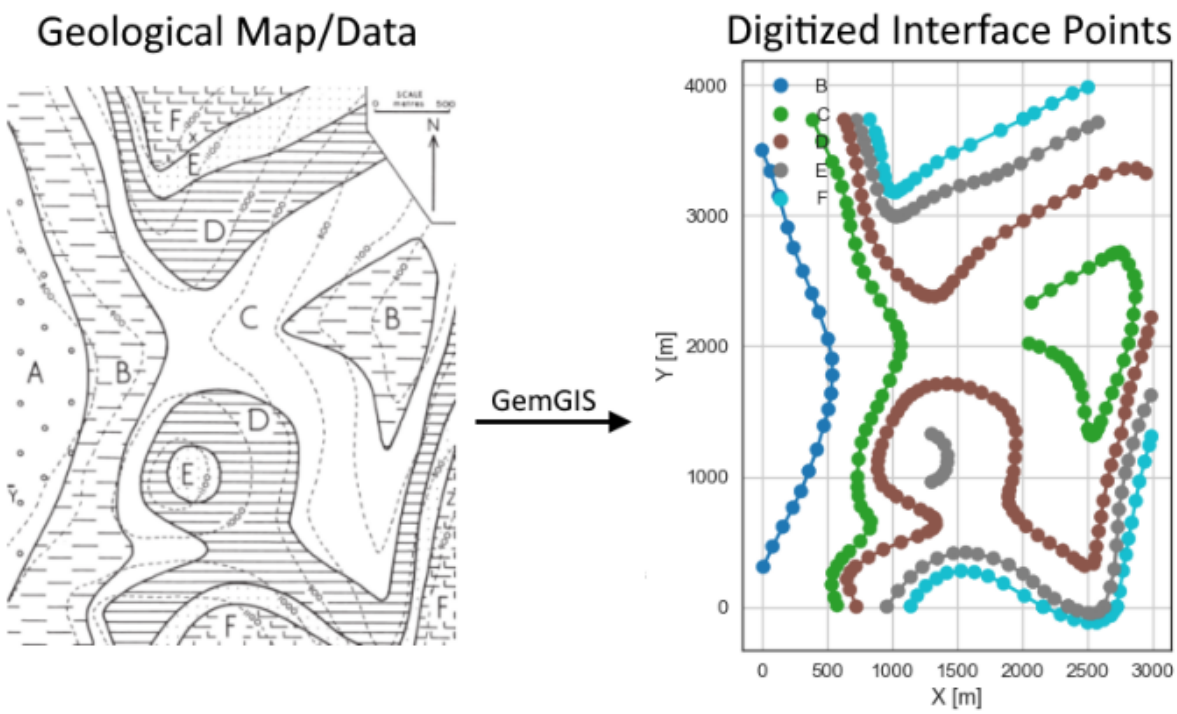
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster3.tif')
```

7.3.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example03.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces3.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	B	LINESTRING (2.323 3499.479, 65.927 3338.734, 1...
1	None	C	LINESTRING (389.730 3730.767, 463.742 3570.022...
2	None	C	LINESTRING (2067.723 2331.475, 2214.591 2427.4...

(continues on next page)

(continued from previous page)

```
3  None         D  LINESTRING (2522.204 3256.627, 2393.839 3180.3...
4  None         E  LINESTRING (2389.213 3623.218, 2259.692 3551.5...
```

Extracting Z coordinate from Digital Elevation Model

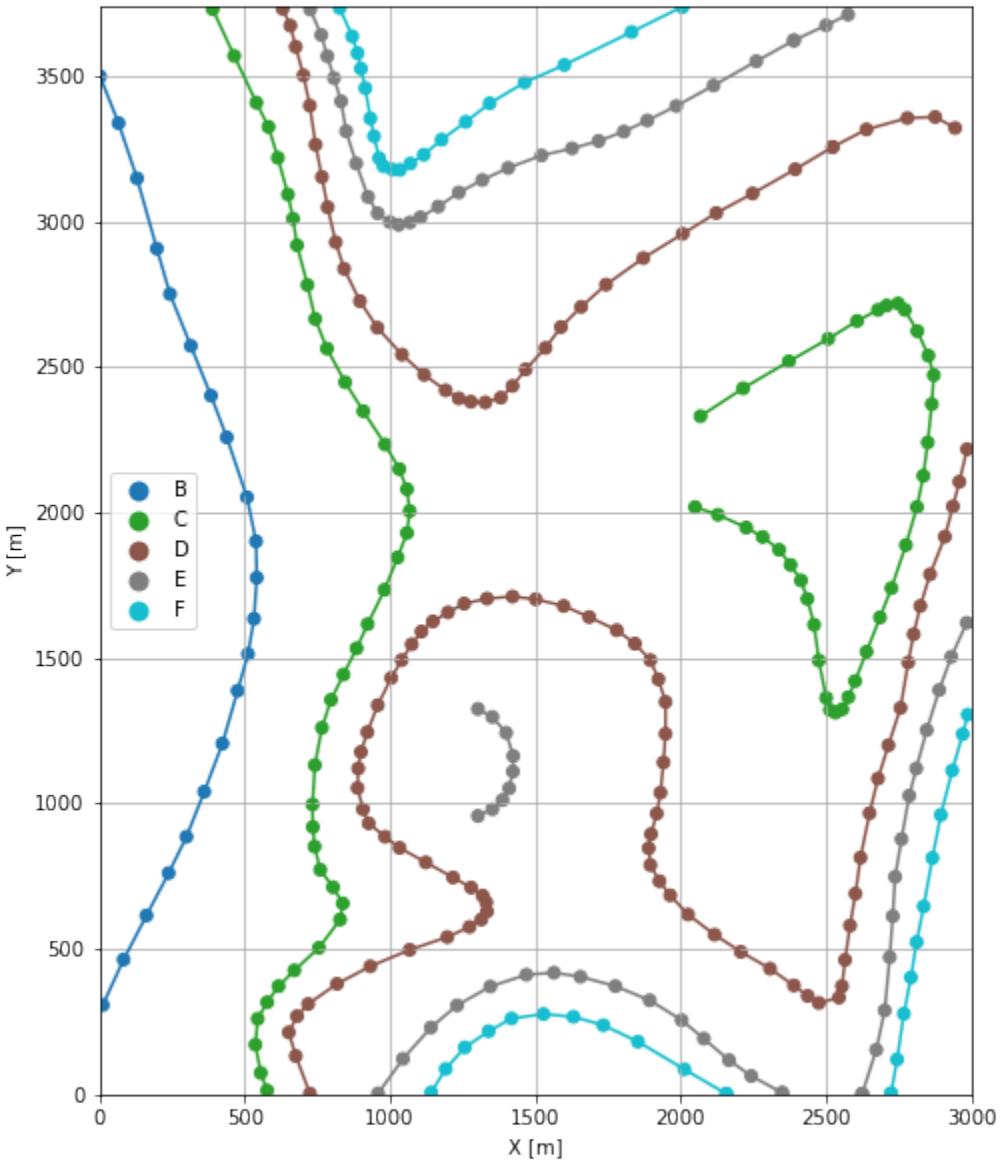
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords
```

[12]:
```
    formation                  geometry       X       Y       Z
169         F  POINT (932.100 3354.924)  932.10 3354.92 1214.19
170         F  POINT (914.753 3459.004)  914.75 3459.00 1213.03
181         F  POINT (1630.589 267.231) 1630.59  267.23 1085.09
180         F  POINT (1527.666 276.483) 1527.67  276.48 1101.00
179         F  POINT (1417.804 261.449) 1417.80  261.45 1118.56
..        ...                        ...     ...     ...     ...
17          B   POINT (238.236 761.031)  238.24  761.03  854.53
18          B   POINT (161.911 615.320)  161.91  615.32  870.34
19          B    POINT (83.273 463.826)   83.27  463.83  886.53
20          B    POINT (11.574 307.707)   11.57  307.71  895.69
0           B    POINT (2.323 3499.479)    2.32 3499.48  898.42

[339 rows x 5 columns]
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3000)
ax.set_ylim(0, 3740)
```

[13]:
```
(0.0, 3740.0)
```



7.3.6 Orientations from Strike Lines

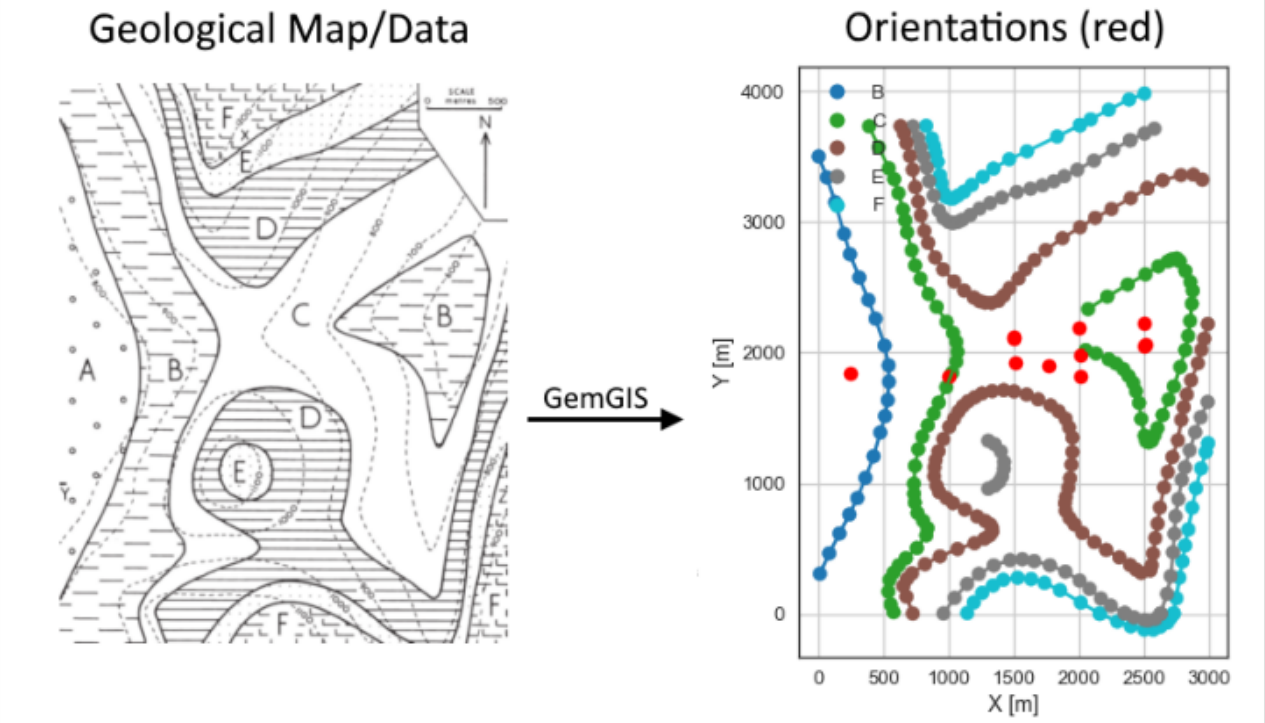
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/orientations_example03.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes3.shp')
strikes
```

[15]:

	id	formation	Z	geometry
0	2	C	700	LINESTRING (2236.563 2441.337, 2244.658 1935.973)
1	1	C	600	LINESTRING (2752.335 2715.413, 2750.022 1809.921)
2	2	D	800	LINESTRING (2242.924 3096.460, 2290.338 441.276)
3	3	D	900	LINESTRING (1739.873 2780.752, 1758.376 1605.810)
4	4	D	1000	LINESTRING (1236.822 2394.501, 1239.134 1677.509)
5	4	E	1100	LINESTRING (1245.495 953.000, 1236.243 3100.507)
6	3	E	1000	LINESTRING (1749.124 3284.381, 1790.756 364.372)
7	2	E	900	LINESTRING (2242.345 63.698, 2259.692 3551.519)
8	1	F	1000	LINESTRING (2012.214 86.827, 2005.275 3738.862)
9	2	F	1100	LINESTRING (1534.026 272.435, 1499.911 3496.010)
10	1	A	800	LINESTRING (486.292 2124.473, 495.544 1456.051)
11	2	A	900	LINESTRING (2.323 3499.479, 11.574 307.707)
12	5	C	1000	LINESTRING (736.009 2693.445, 761.729 511.476)
13	1	B	800	LINESTRING (489.213 2109.525, 496.562 1470.183)
14	2	B	900	LINESTRING (2.323 3499.479, 11.574 307.707)
15	1	D	700	LINESTRING (2752.631 1310.960, 2747.732 3388.209)
16	1	E	800	LINESTRING (2745.282 835.740, 2750.181 3753.197)
17	3	C	800	LINESTRING (1759.324 1605.522, 1739.727 2783.774)

(continues on next page)

(continued from previous page)

```
18   4         C   900  LINESTRING (1238.786 1676.560, 1237.562 2396.739)
```

Calculate Orientations for each formation

[16]:
```
orientations_b = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B'].sort_values(by='id', ascending=True).reset_index())
orientations_b
```

[16]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 11.70    89.81 850.00  POINT (249.918 1846.723)      1.00         B 249.92

        Y
0 1846.72
```

[17]:
```
orientations_c = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C'].sort_values(by='id', ascending=True).reset_index())
orientations_c
```

[17]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 11.19    89.89 650.00  POINT (2495.895 2225.661)      1.00         C
1 11.52    89.05 750.00  POINT (1995.068 2191.652)      1.00         C
2 11.12    89.28 850.00  POINT (1493.850 2115.649)      1.00         C
3 11.52    89.38 950.00   POINT (993.522 1819.555)      1.00         C

        X       Y
0 2495.89 2225.66
1 1995.07 2191.65
2 1493.85 2115.65
3  993.52 1819.56
```

[18]:
```
orientations_d = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D'].sort_values(by='id', ascending=True).reset_index())
orientations_d
```

[18]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 11.82    89.31 750.00  POINT (2508.406 2059.226)      1.00         D
1 11.12    89.00 850.00  POINT (2007.877 1981.074)      1.00         D
2 11.11    89.29 950.00  POINT (1493.551 2114.643)      1.00         D

        X       Y
0 2508.41 2059.23
1 2007.88 1981.07
2 1493.55 2114.64
```

[19]:
```
orientations_e = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'E'].sort_values(by='Z', ascending=True).reset_index())
orientations_e
```

[19]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 11.53    90.21  850.00  POINT (2499.375 2051.039)      1.00         E
1 12.45    89.83  950.00  POINT (2010.479 1815.993)      1.00         E
2 10.98    89.38 1050.00  POINT (1505.405 1925.565)      1.00         E
```

(continues on next page)

(continued from previous page)

```
        X       Y
0 2499.38 2051.04
1 2010.48 1815.99
2 1505.40 1925.57
```

[20]:
```
orientations_f = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'F'].sort_values(by='id', ascending=True).reset_index())
orientations_f
```

[20]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 11.82    89.67 1050.00  POINT (1762.857 1898.533)      1.00         F

        X       Y
0 1762.86 1898.53
```

Merging Orientations

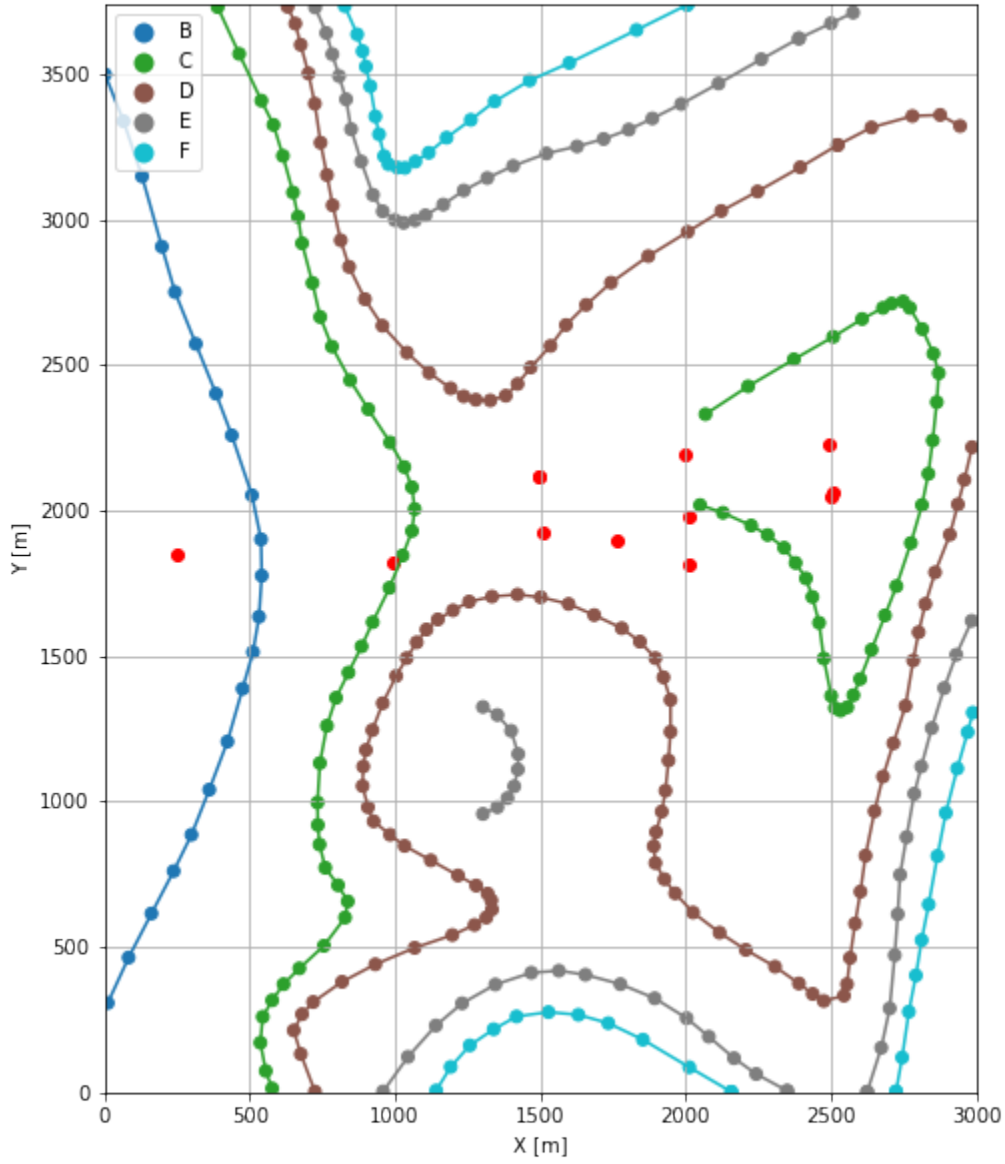
[21]:
```
import pandas as pd
orientations = pd.concat([orientations_b, orientations_c, orientations_d, orientations_e,
↪ orientations_f])
orientations = orientations[orientations['formation'].isin(['F', 'E', 'D', 'C', 'B'])].
↪reset_index()
orientations
```

[21]:
```
    index   dip  azimuth       Z                   geometry  polarity  \
0       0 11.70    89.81  850.00   POINT (249.918 1846.723)      1.00
1       0 11.19    89.89  650.00  POINT (2495.895 2225.661)      1.00
2       1 11.52    89.05  750.00  POINT (1995.068 2191.652)      1.00
3       2 11.12    89.28  850.00  POINT (1493.850 2115.649)      1.00
4       3 11.52    89.38  950.00   POINT (993.522 1819.555)      1.00
5       0 11.82    89.31  750.00  POINT (2508.406 2059.226)      1.00
6       1 11.12    89.00  850.00  POINT (2007.877 1981.074)      1.00
7       2 11.11    89.29  950.00  POINT (1493.551 2114.643)      1.00
8       0 11.53    90.21  850.00  POINT (2499.375 2051.039)      1.00
9       1 12.45    89.83  950.00  POINT (2010.479 1815.993)      1.00
10      2 10.98    89.38 1050.00  POINT (1505.405 1925.565)      1.00
11      0 11.82    89.67 1050.00  POINT (1762.857 1898.533)      1.00

   formation       X       Y
0          B  249.92 1846.72
1          C 2495.89 2225.66
2          C 1995.07 2191.65
3          C 1493.85 2115.65
4          C  993.52 1819.56
5          D 2508.41 2059.23
6          D 2007.88 1981.07
7          D 1493.55 2114.64
8          E 2499.38 2051.04
9          E 2010.48 1815.99
10         E 1505.40 1925.57
11         F 1762.86 1898.53
```

Plotting the Orientations

[22]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3000)
ax.set_ylim(0, 3740)
```

[22]:
```
(0.0, 3740.0)
```



7.3.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[23]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[24]:
```
geo_model = gp.create_model('Model3')
geo_model
```

[24]:
```
Model3  2022-04-04 15:53
```

Initiate Data

[25]:
```
gp.init_data(geo_model, [0, 3000, 0, 3740, 200, 1200], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[25]:
```
Model3  2022-04-04 15:53
```

Model Surfaces

[26]:
```
geo_model.surfaces
```

[26]:

	surface	series	order_surfaces	color	id
0	F	Default series	1	#015482	1
1	E	Default series	2	#9f0052	2
2	D	Default series	3	#ffbe00	3
3	C	Default series	4	#728f02	4
4	B	Default series	5	#443988	5

Mapping the Stack to Surfaces

[27]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('F'),
                          'Strata2': ('E'),
                          'Strata3': ('D'),
                          'Strata4': ('C'),
                          'Strata5': ('B'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('A')
```

[27]:

	surface	series	order_surfaces	color	id
0	F	Strata1	1	#015482	1
1	E	Strata2	1	#9f0052	2
2	D	Strata3	1	#ffbe00	3
3	C	Strata4	1	#728f02	4
4	B	Strata5	1	#443988	5
5	A	Strata5	2	#ff3f20	6

Showing the Number of Data Points

[28]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[28]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	F	Strata1	1	#015482	1	45	1
1	E	Strata2	1	#9f0052	2	68	3
2	D	Strata3	1	#ffbe00	3	113	3
3	C	Strata4	1	#728f02	4	77	4
4	B	Strata5	1	#443988	5	21	1
5	A	Strata5	2	#ff3f20	6	0	0

Loading Digital Elevation Model

[29]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster3.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[29]:
```
Grid Object. Values:
array([[  15.        ,   18.7       ,  205.        ],
       [  15.        ,   18.7       ,  215.        ],
       [  15.        ,   18.7       ,  225.        ],
       ...,
       [2992.5       , 3702.6       ,  833.75482178],
       [2992.5       , 3717.56      ,  838.74261475],
       [2992.5       , 3732.52      ,  843.78088379]])
```

Defining Custom Section

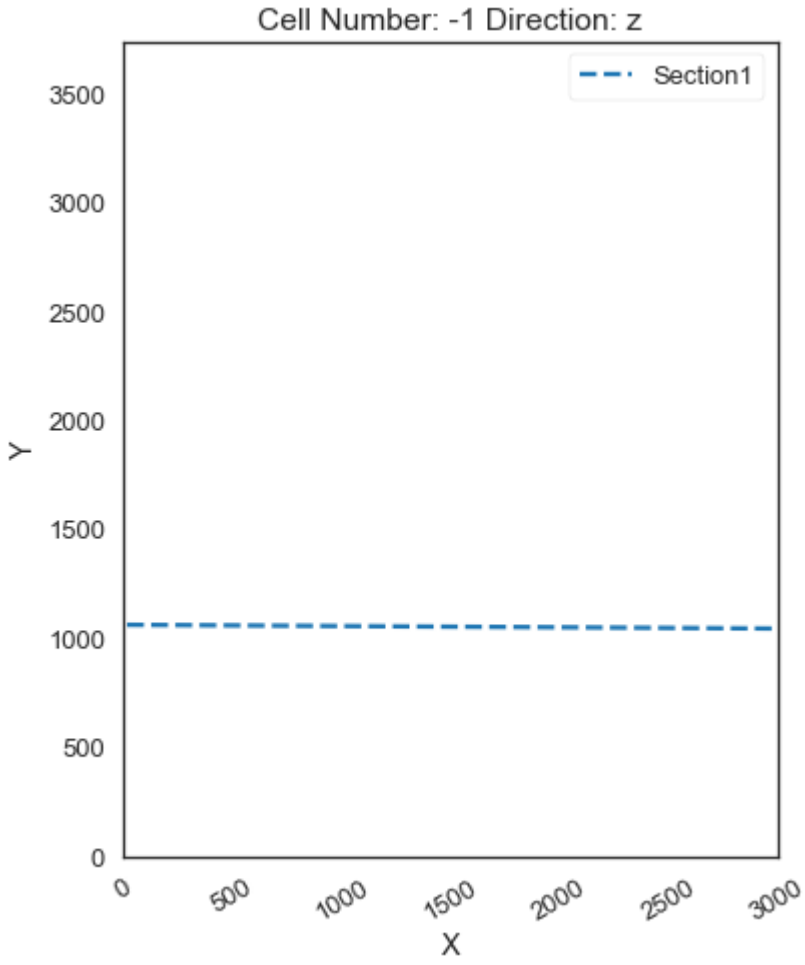
[30]:
```
custom_section = gpd.read_file(file_path + 'customsection3.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='name')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[30]:
```
                                                start                                                 ␣
↪stop resolution    dist
Section1  [15.217892876134727, 1063.5514258527073]  [2991.4655480049296, 1046.
↪4043200206897]  [100, 80] 2976.30
```

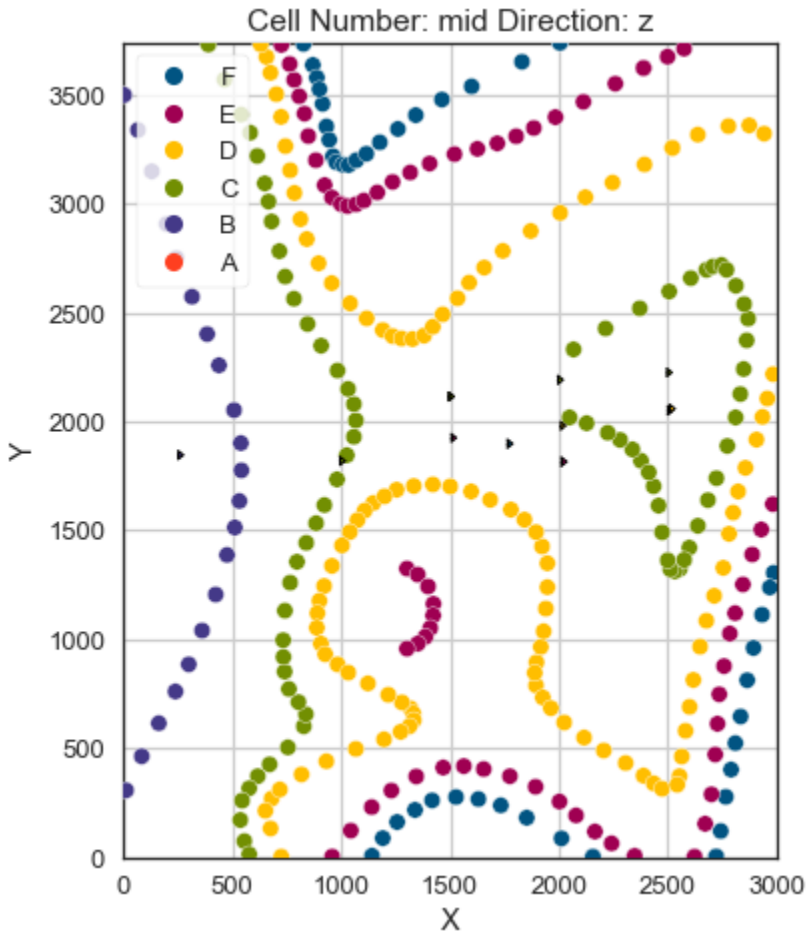
[31]:
```
gp.plot.plot_section_traces(geo_model)
```

[31]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1d547775a90>
```

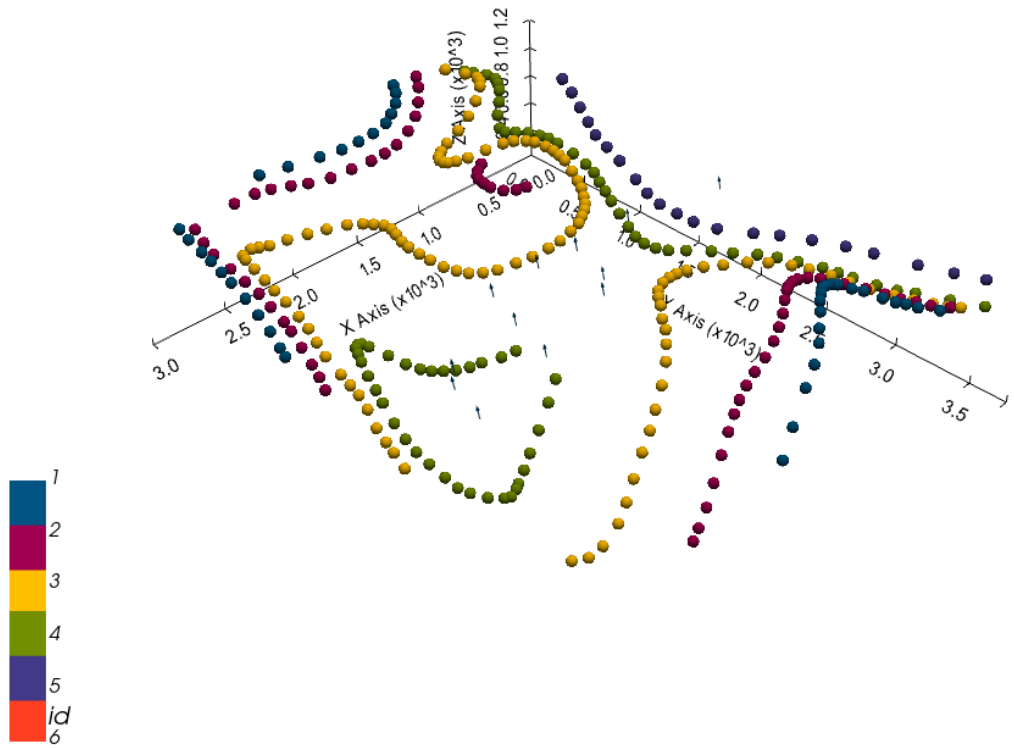


Plotting Input Data

[32]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[33]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[33]: <gempy.plot.vista.GemPyToVista at 0x1d5431d25e0>

Setting the Interpolator

[34]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                         values
range                   4897.71
$C_o$                 571133.33
drift equations  [3, 3, 3, 3, 3]
```

[34]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1d541724ac0>
```

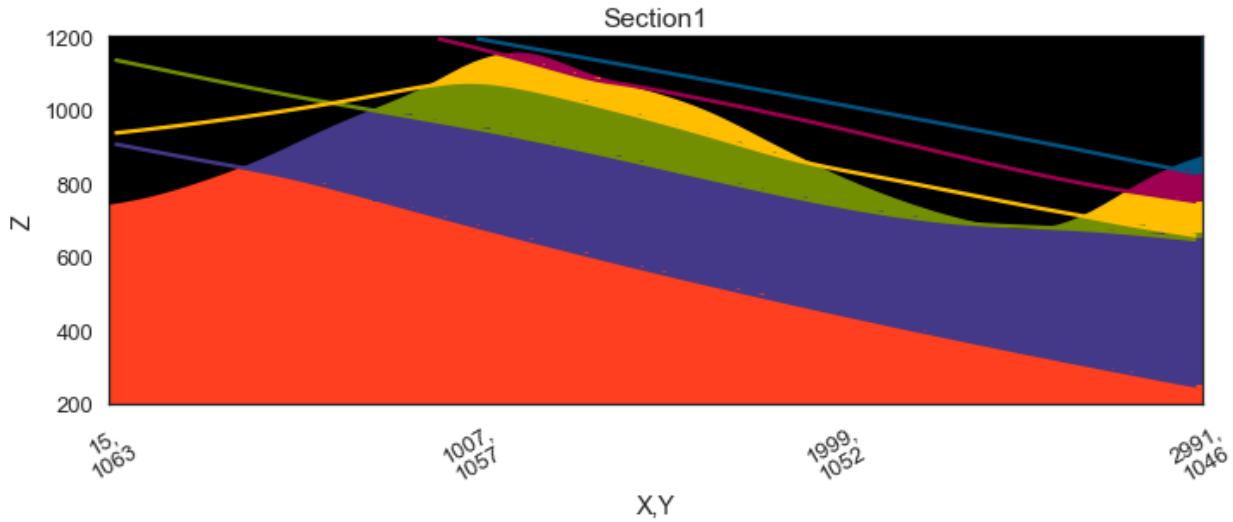
Computing Model

[35]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

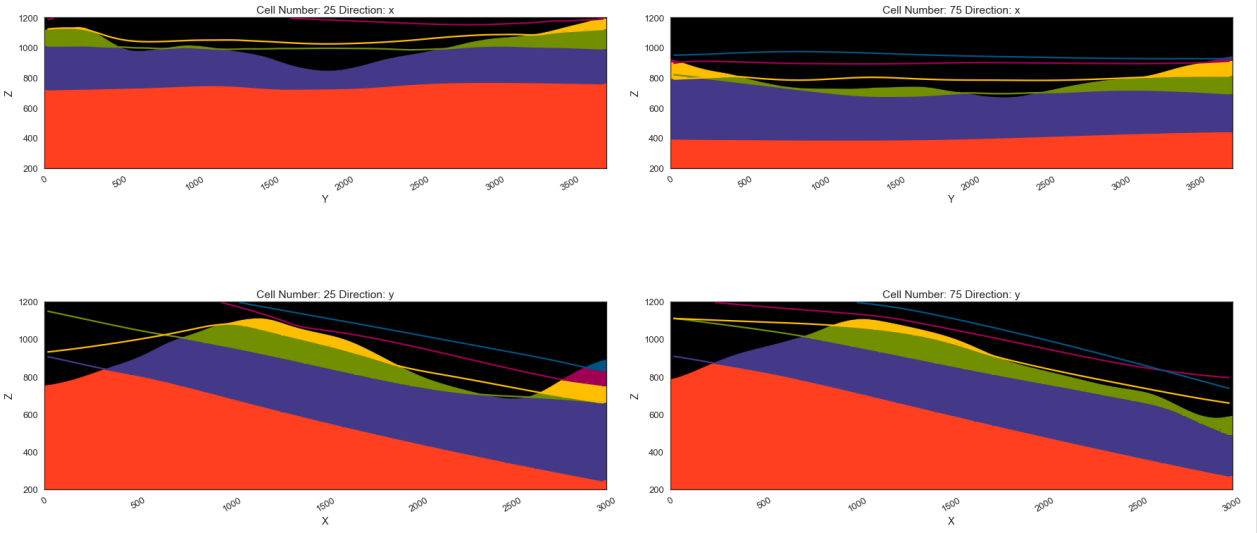
[36]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[36]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1d545715c70>
```



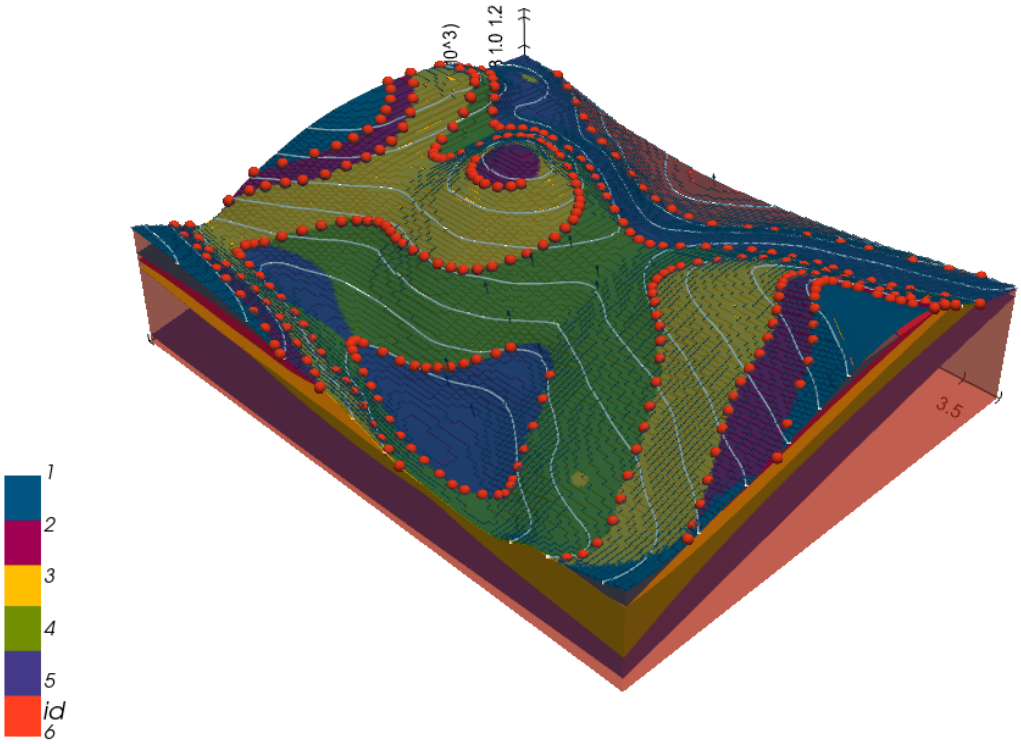
[37]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[37]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1d54868da30>
```



Plotting 3D Model

[38]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:
```
```

7.4 Example 4 - Unconformably Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 2963 m wide (W-E extent) and 3715 m high (N-S extent). The vertical model extent varies between 0 m and 1000 m. The model represents several planar stratigraphic units (yellow to light red) dipping towards the west above an unspecified basement (green). The blue and purple unit overlay the remaining strata unconformably and dip to the north. The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

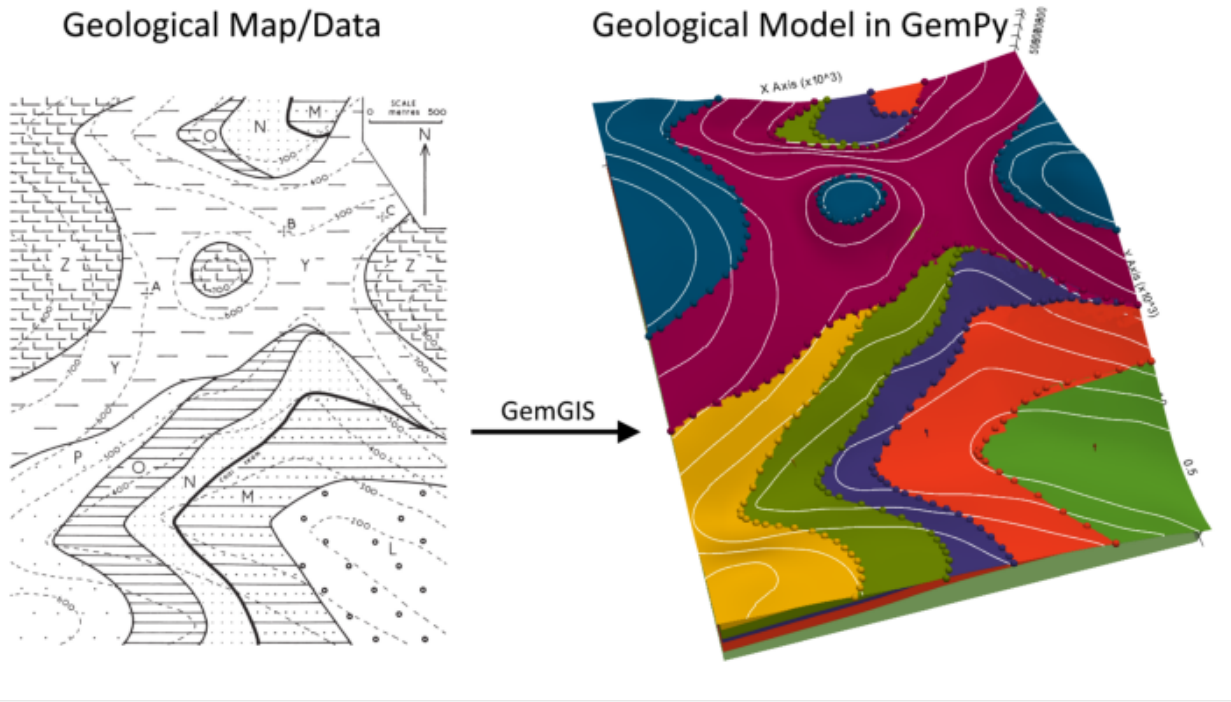
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/cover_example04.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.4.1 Licensing

7.4.2 Import GemGIS

If you have installed `GemGIS` via pip and conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.4.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

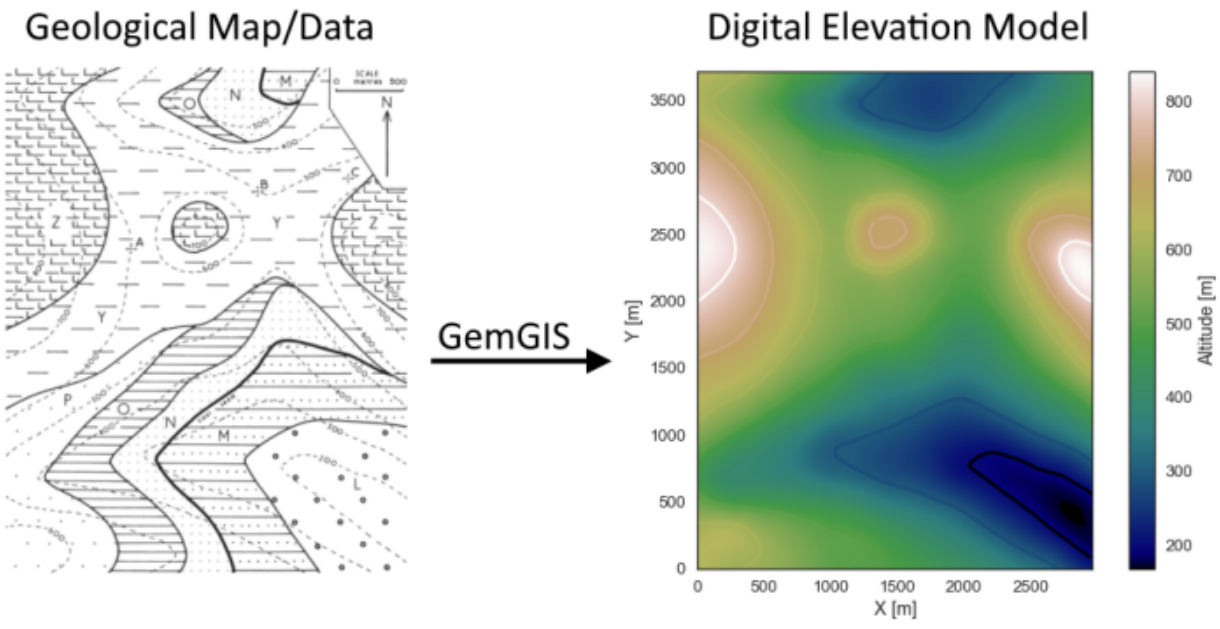
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example04/'
gg.download_gemgis_data.download_tutorial_data(filename="example04_unconformably_dipping_
↪layers.zip", dirpath=file_path)
```

7.4.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example04.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo4.shp')
topo.head()
```

[6]:
```
     id    Z                                           geometry
0  None  600  LINESTRING (-1.172 334.506, 98.350 346.728, 21...
1  None  500  LINESTRING (1088.333 6.258, 1182.617 58.638, 1...
2  None  200  LINESTRING (2947.824 83.082, 2785.446 201.810,...
3  None  300  LINESTRING (2694.654 8.004, 2549.736 86.574, 2...
4  None  400  LINESTRING (2202.282 2.766, 2032.919 95.304, 1...
```

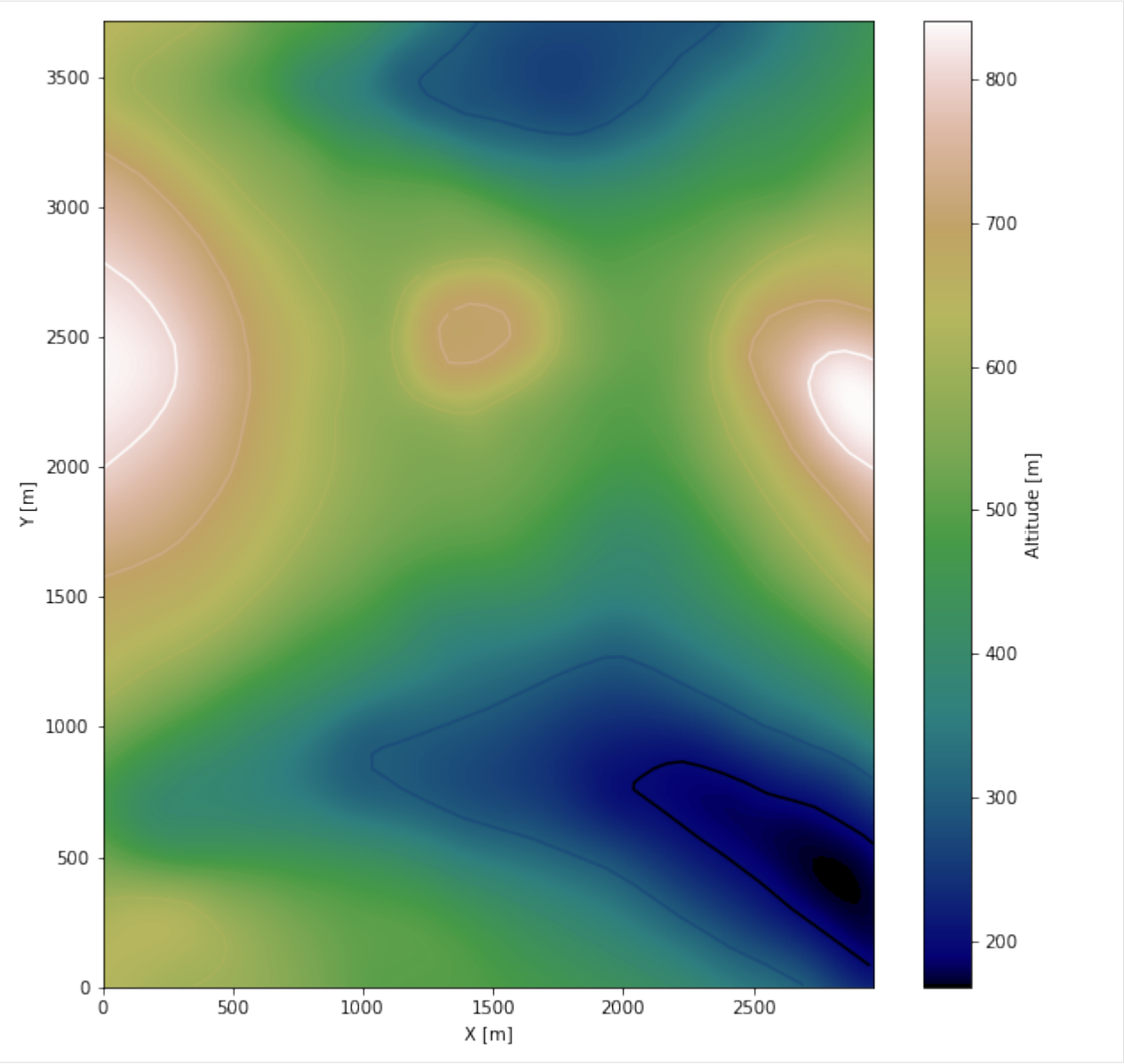
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 2963, 0, 3715], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2963)
ax.set_ylim(0, 3715)
```

[8]:
```
(0.0, 3715.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path =' raster4.tif', extent = [0, 2963, 0, 3715], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

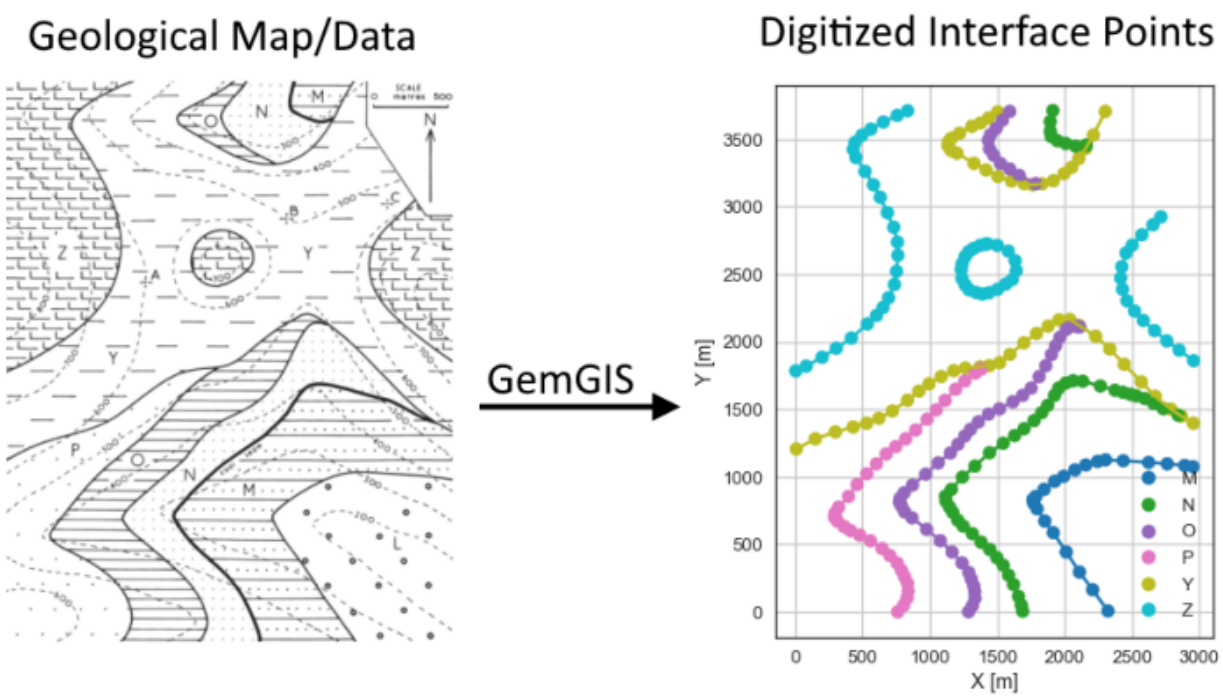
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster4.tif')
```

7.4.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example04.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces4.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	M	LINESTRING (2320.137 7.131, 2206.647 164.271, ...
1	None	N	LINESTRING (1688.084 5.385, 1677.608 50.781, 1...
2	None	O	LINESTRING (1286.504 1.893, 1314.440 45.543, 1...
3	None	P	LINESTRING (760.957 0.147, 804.607 49.035, 830...
4	None	Y	LINESTRING (8.431 1206.634, 149.857 1281.712, ...

Extracting Z coordinate from Digital Elevation Model

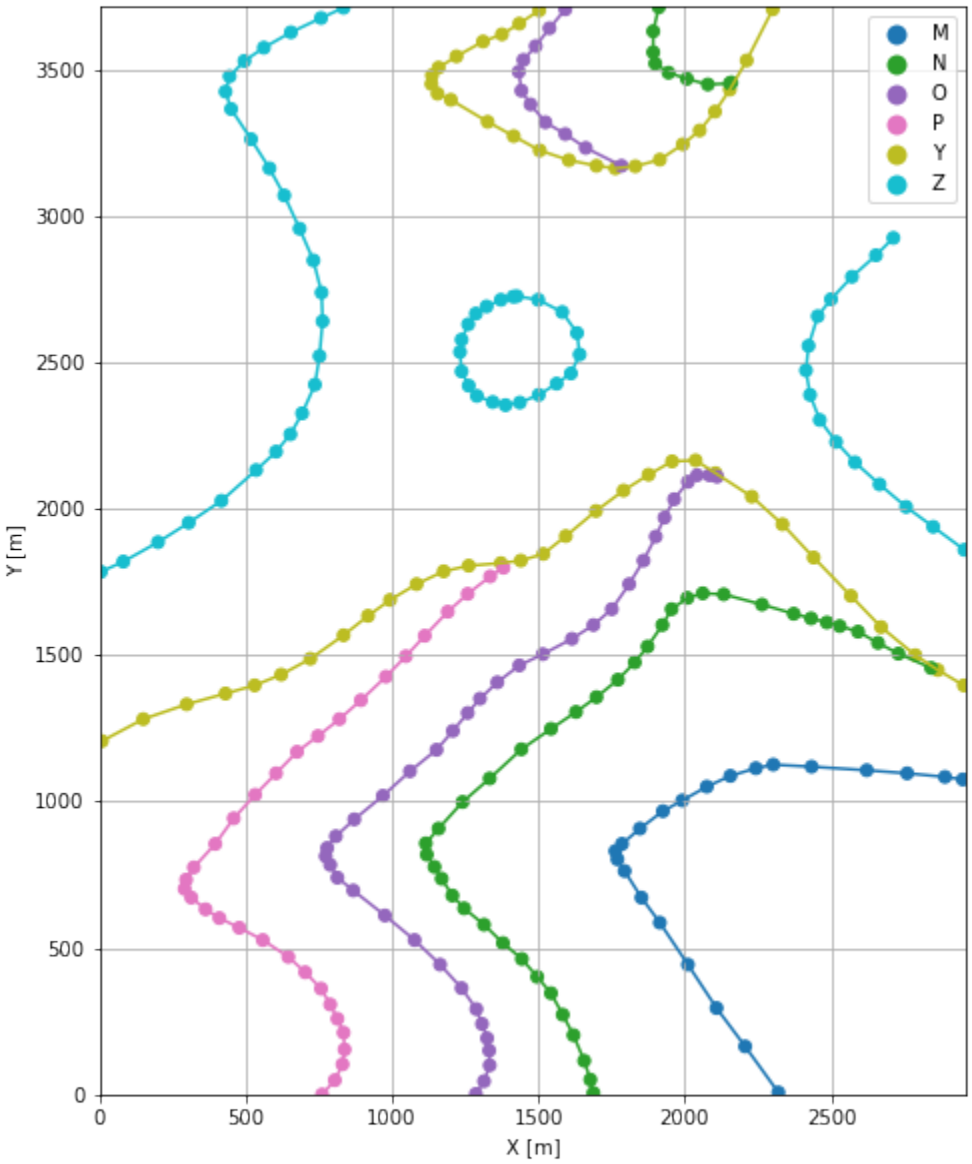
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords
```

[12]:
```
    formation                    geometry       X       Y      Z
193         Z  POINT (1581.578 2671.529) 1581.58 2671.53 663.80
196         Z  POINT (1611.260 2462.009) 1611.26 2462.01 680.92
187         Z   POINT (305.251 1952.176)  305.25 1952.18 742.32
188         Z   POINT (200.491 1885.828)  200.49 1885.83 748.24
189         Z    POINT (81.763 1819.480)   81.76 1819.48 752.76
..        ...                        ...     ...     ...    ...
19          M  POINT (2760.129 1096.636) 2760.13 1096.64 378.75
20          M  POINT (2889.333 1084.414) 2889.33 1084.41 414.59
21          M  POINT (2950.443 1075.684) 2950.44 1075.68 430.87
1           M   POINT (2206.647 164.271) 2206.65  164.27 343.14
0           M     POINT (2320.137 7.131) 2320.14    7.13 375.57

[271 rows x 5 columns]
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2963)
ax.set_ylim(0, 3715)
```

[13]:
```
(0.0, 3715.0)
```



7.4.6 Orientations from Strike Lines

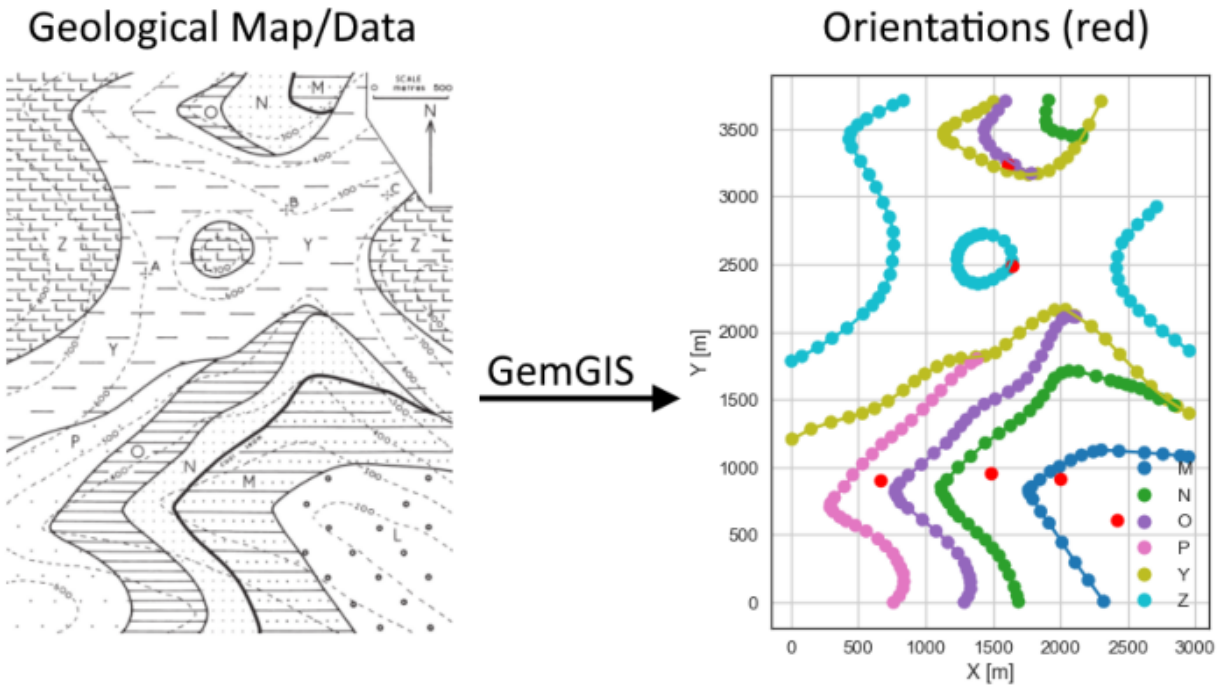
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/orientations_example04.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes4.shp')
strikes
```

[15]:

	id	formation	Z	geometry
0	3	Z	700	LINESTRING (532.231 2126.776, 2622.195 2125.030)
1	2	Z	600	LINESTRING (729.529 2846.129, 2660.607 2863.589)
2	1	M	300	LINESTRING (2307.915 1121.080, 2065.220 359.823)
3	2	M	400	LINESTRING (2836.953 1094.890, 2479.023 -125.566)
4	2	N	400	LINESTRING (2019.824 1695.514, 1534.436 356.331)
5	1	N	300	LINESTRING (1466.342 1660.594, 902.383 96.177)
6	2	P	500	LINESTRING (1197.458 1655.356, 738.259 386.013)
7	1	P	400	LINESTRING (516.517 1208.380, 186.523 361.569)
8	3	N	500	LINESTRING (2490.372 1610.833, 1942.127 4.512)
9	1	Z	500	LINESTRING (616.039 3606.513, 2409.183 3614.370)

Calculate Orientations for each formation

[16]:
```
orientations_m = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'M'].sort_values(by='id', ascending=True).reset_index())
orientations_m
```

[16]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 10.99   286.72 350.00  POINT (2422.278 612.557)      1.00         M 2422.28

       Y
0 612.56
```

[17]:
```
orientations_n = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'N'].sort_values(by='Z', ascending=True).reset_index())
orientations_n
```

[17]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 11.17   289.87 350.00  POINT (1480.746 952.154)      1.00         N 1480.75
1 11.95   289.29 450.00  POINT (1996.690 916.797)      1.00         N 1996.69

       Y
0 952.15
1 916.80
```

[18]:
```
orientations_p = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'P'].sort_values(by='Z', ascending=True).reset_index())
orientations_p
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 11.57   290.32 450.00  POINT (659.689 902.829)      1.00         P 659.69

       Y
0 902.83
```

[19]:
```
orientations_y = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'Z'].sort_values(by='Z', ascending=True).reset_index())
orientations_y
```

[19]:
```
   dip  azimuth      Z                   geometry  polarity formation       X  \
0 7.56   359.61 550.00  POINT (1603.840 3232.650)      1.00         Z 1603.84
1 7.91   359.79 650.00  POINT (1636.141 2490.381)      1.00         Z 1636.14

        Y
0 3232.65
1 2490.38
```

Merging Orientations

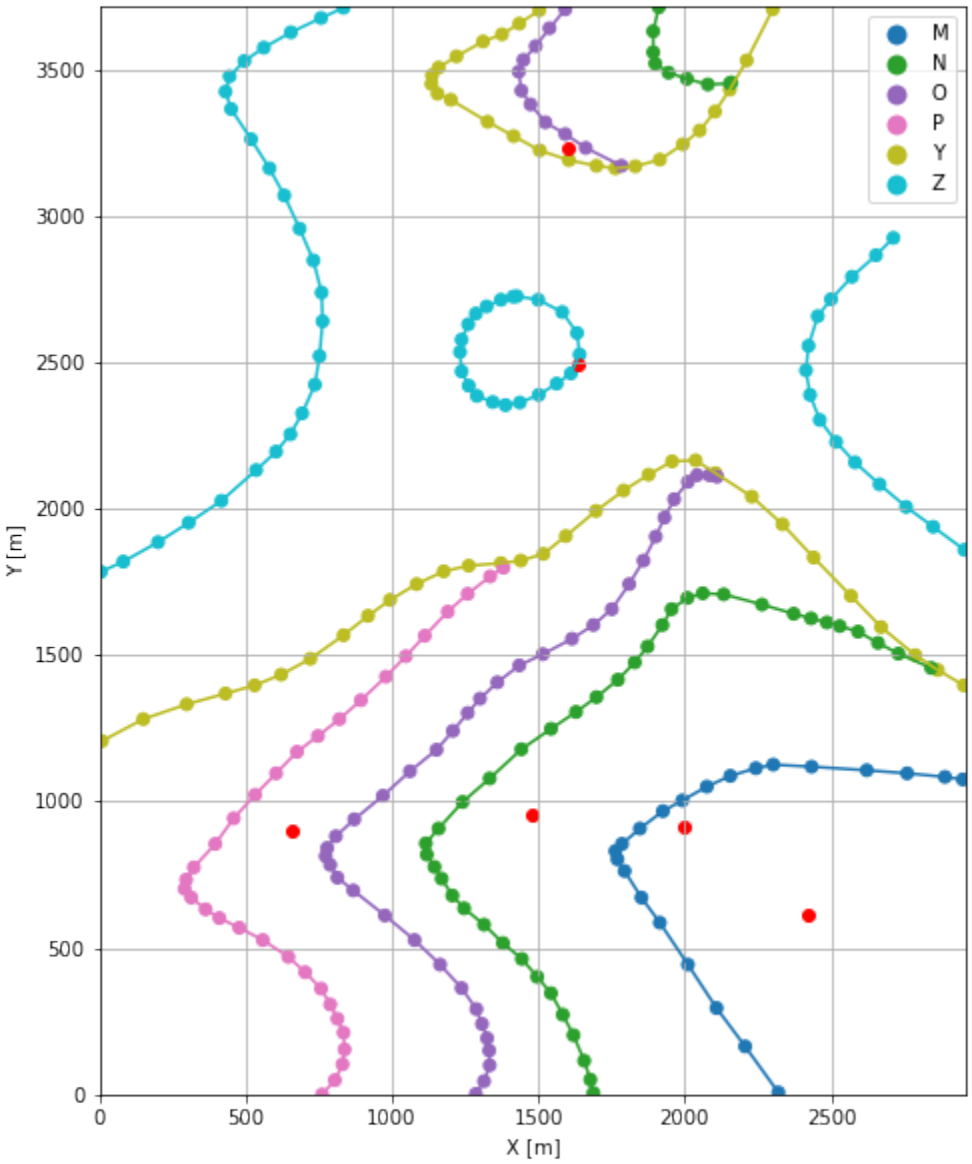
[20]:
```
import pandas as pd
orientations = pd.concat([orientations_m, orientations_n, orientations_p, orientations_
↪y]).reset_index()
orientations
```

[20]:
```
   index   dip  azimuth      Z                   geometry  polarity formation  \
0      0 10.99   286.72 350.00   POINT (2422.278 612.557)      1.00         M
1      0 11.17   289.87 350.00   POINT (1480.746 952.154)      1.00         N
2      1 11.95   289.29 450.00   POINT (1996.690 916.797)      1.00         N
3      0 11.57   290.32 450.00    POINT (659.689 902.829)      1.00         P
4      0  7.56   359.61 550.00  POINT (1603.840 3232.650)      1.00         Z
5      1  7.91   359.79 650.00  POINT (1636.141 2490.381)      1.00         Z

         X       Y
0 2422.28  612.56
1 1480.75  952.15
2 1996.69  916.80
3  659.69  902.83
4 1603.84 3232.65
5 1636.14 2490.38
```

Plotting the Orientations

[21]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2963)
ax.set_ylim(0, 3715)
```

[21]:
```
(0.0, 3715.0)
```



7.4.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[22]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[23]:
```
geo_model = gp.create_model('Model4')
geo_model
```

[23]:
```
Model4  2022-04-04 18:59
```

Initiate Data

[24]:
```
gp.init_data(geo_model, [0, 2963, 0, 3715, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[24]:
```
Model4  2022-04-04 18:59
```

Model Surfaces

[25]:
```
geo_model.surfaces
```

[25]:

	surface	series	order_surfaces	color	id
0	Z	Default series	1	#015482	1
1	Y	Default series	2	#9f0052	2
2	P	Default series	3	#ffbe00	3
3	O	Default series	4	#728f02	4
4	N	Default series	5	#443988	5
5	M	Default series	6	#ff3f20	6

Mapping the Stack to Surfaces

[26]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Z', 'Y'),
                          'Strata2': ('P', 'O', 'N', 'M'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[26]:

	surface	series	order_surfaces	color	id
0	Z	Strata1	1	#015482	1
1	Y	Strata1	2	#9f0052	2
2	P	Strata2	1	#ffbe00	3
3	O	Strata2	2	#728f02	4
4	N	Strata2	3	#443988	5
5	M	Strata2	4	#ff3f20	6
6	Basement	Strata2	5	#5DA629	7

Showing the Number of Data Points

[27]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[27]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Z	Strata1	1	#015482	1	62	2
1	Y	Strata1	2	#9f0052	2	55	0
2	P	Strata2	1	#ffbe00	3	33	1
3	O	Strata2	2	#728f02	4	50	0
4	N	Strata2	3	#443988	5	49	2
5	M	Strata2	4	#ff3f20	6	22	1
6	Basement	Strata2	5	#5DA629	7	0	0

Loading Digital Elevation Model

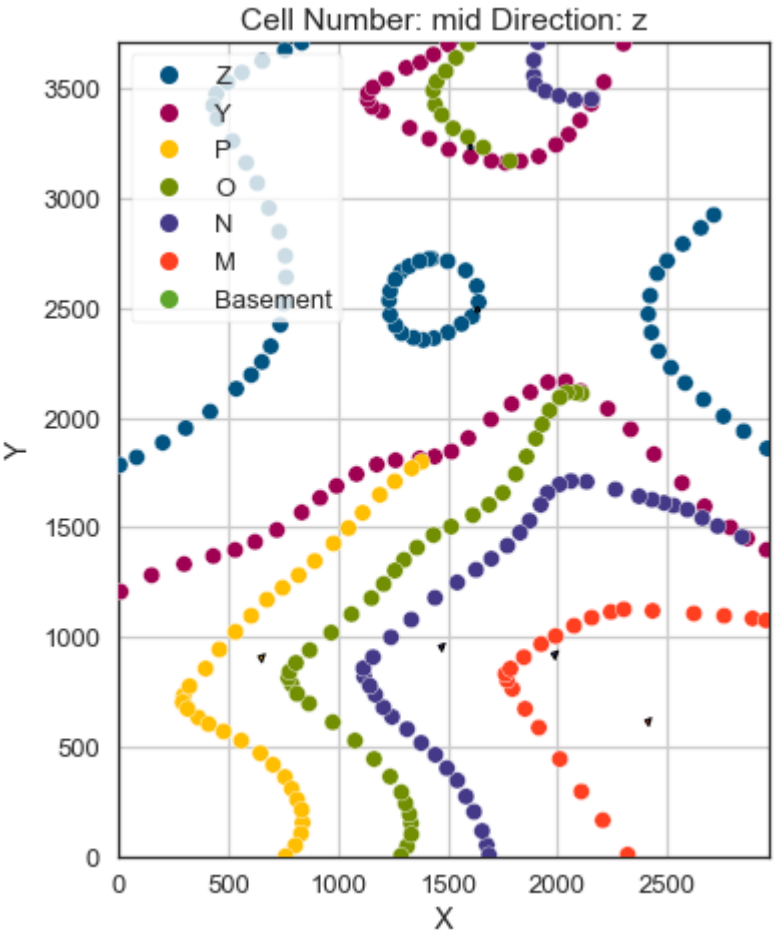
[28]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster4.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[28]:
```
Grid Object. Values:
array([[  14.815     ,   18.575     ,    5.        ],
       [  14.815     ,   18.575     ,   15.        ],
       [  14.815     ,   18.575     ,   25.        ],
       ...,
       [2958.01178451, 3690.03360215,  446.43917847],
       [2958.01178451, 3700.02016129,  445.69622803],
       [2958.01178451, 3710.00672043,  444.99276733]])
```

Plotting Input Data

[29]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[30]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```

[30]:
```
<gempy.plot.vista.GemPyToVista at 0x1f8883ae880>
```

Setting the Interpolator

[31]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              4855.99
$C_o$            561442.71
drift equations     [3, 3]
```

[31]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1f88515ab50>
```

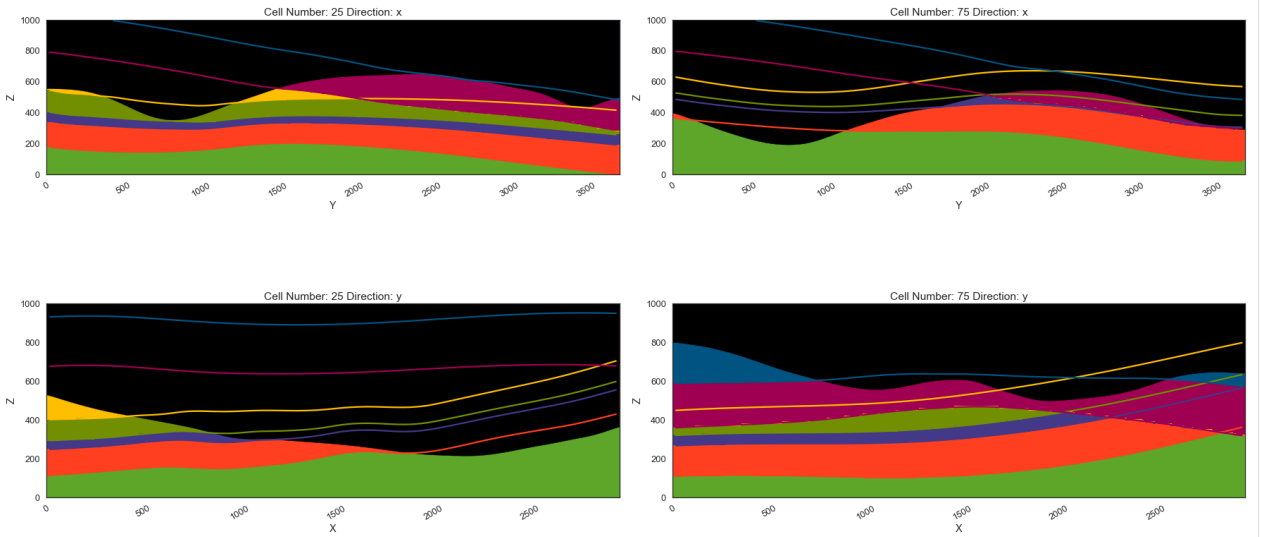
Computing Model

[32]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

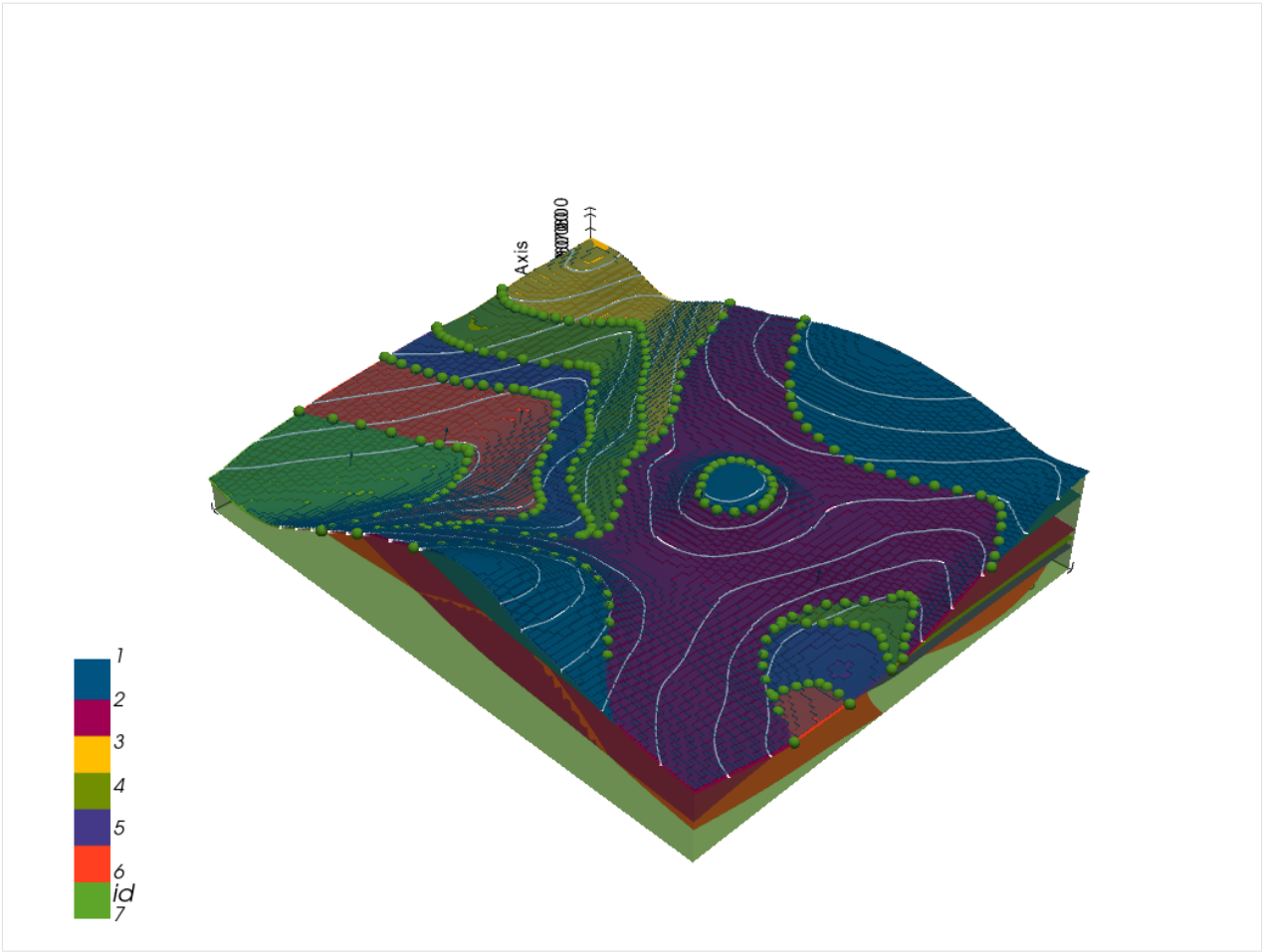
[33]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[33]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1f889906940>
```



Plotting 3D Model

[34]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



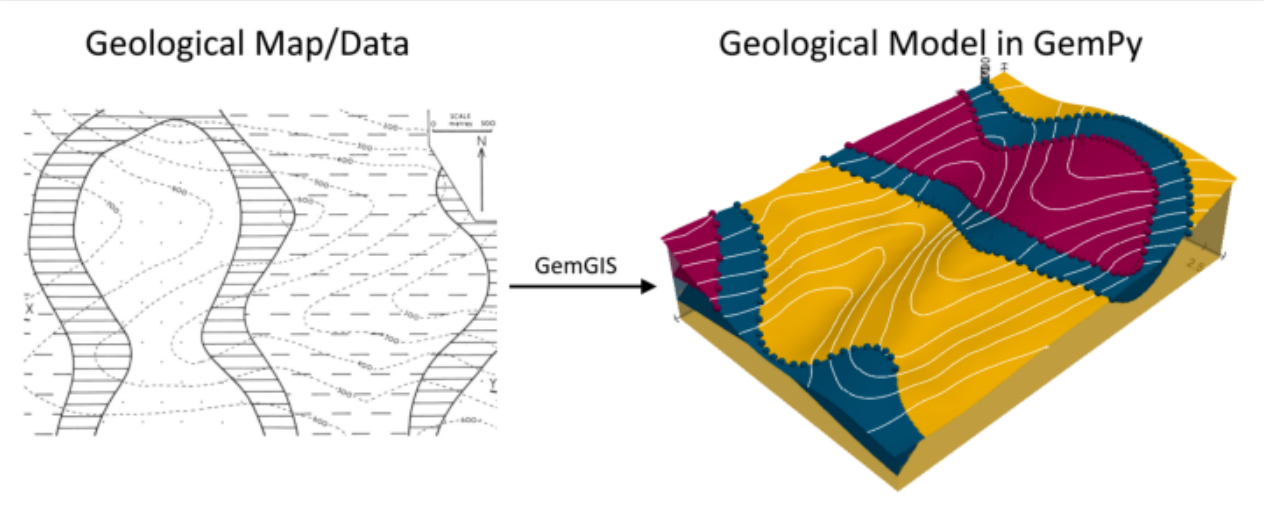
[ ]:

7.5 Example 5 - Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3942 m wide (W-E extent) and 2710 m high (N-S extent). The vertical extent varies between -200 m and 1000 m. The model represents two folded layers (purple and blue) above a basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example05.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.5.1 Licensing

Computational Geosciences and Reservoir Engineering, RWTH Aachen University, Authors: Alexander Juestel. For more information contact: alexander.juestel(at)rwth-aachen.de

This work is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/)

7.5.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.5.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

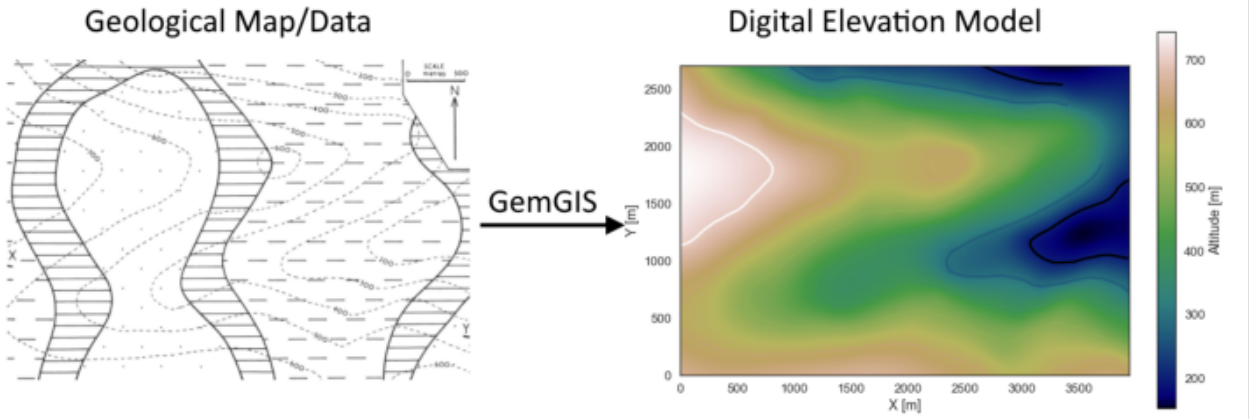
[4]:
```
file_path = 'data/example05/'
gg.download_gemgis_data.download_tutorial_data(filename="example05_folded_layers.zip",␣
↪dirpath=file_path)
```

```
Downloading file 'example05_folded_layers.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample05_folded_layers.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\example\data\example05'.
```

7.5.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example05.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo5.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	700	LINESTRING (1.537 2299.193, 29.191 2272.378, 7...
1	None	600	LINESTRING (81.145 2708.127, 130.586 2641.927,...
2	None	600	LINESTRING (6.565 403.682, 56.006 360.945, 103...
3	None	600	LINESTRING (3278.040 2.289, 3308.208 32.456, 3...
4	None	600	LINESTRING (2258.218 1964.001, 2375.536 1953.1...

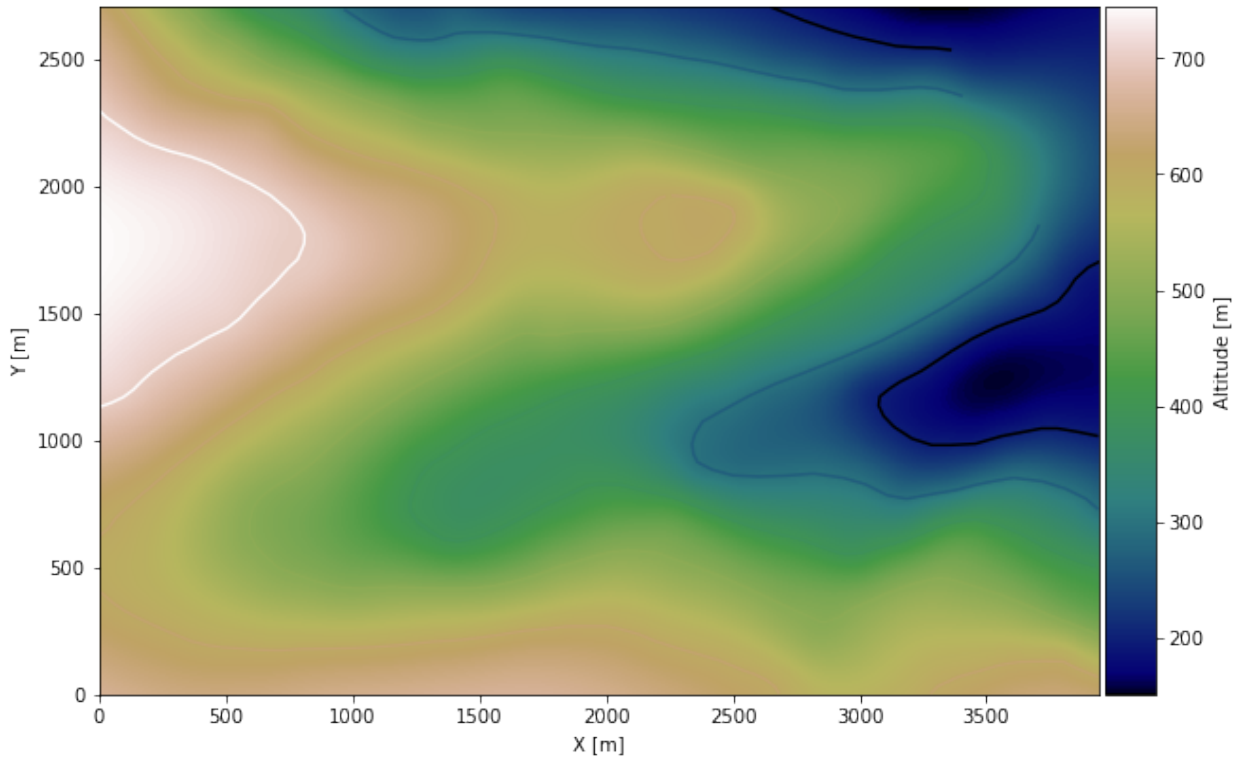
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable
fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 3942, 0, 2710], cmap='gist_earth
↪')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3942)
ax.set_ylim(0, 2710)
```

[8]:
```
(0.0, 2710.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster5.tif', extent = [0, 3942, 0, 2710], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

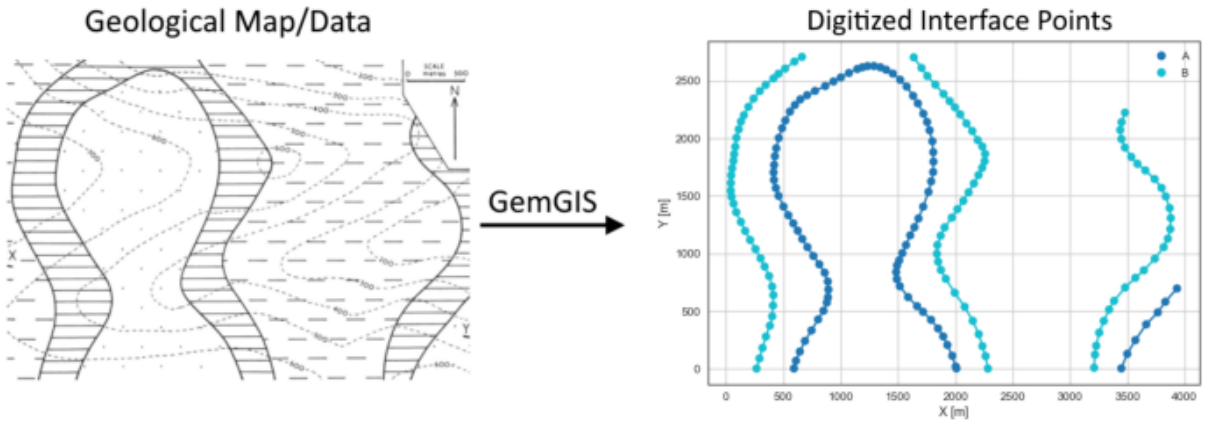
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster5.tif')
```

7.5.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example05.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces5.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	A	LINESTRING (591.475 2.289, 609.073 69.327, 643...
1	None	A	LINESTRING (3448.150 3.127, 3500.105 128.824, ...
2	None	B	LINESTRING (265.501 1.451, 290.641 89.439, 317...
3	None	B	LINESTRING (2284.196 1.451, 2264.084 109.550, ...
4	None	B	LINESTRING (3480.832 2222.937, 3450.664 2140.8...

Extracting Z coordinate from Digital Elevation Model

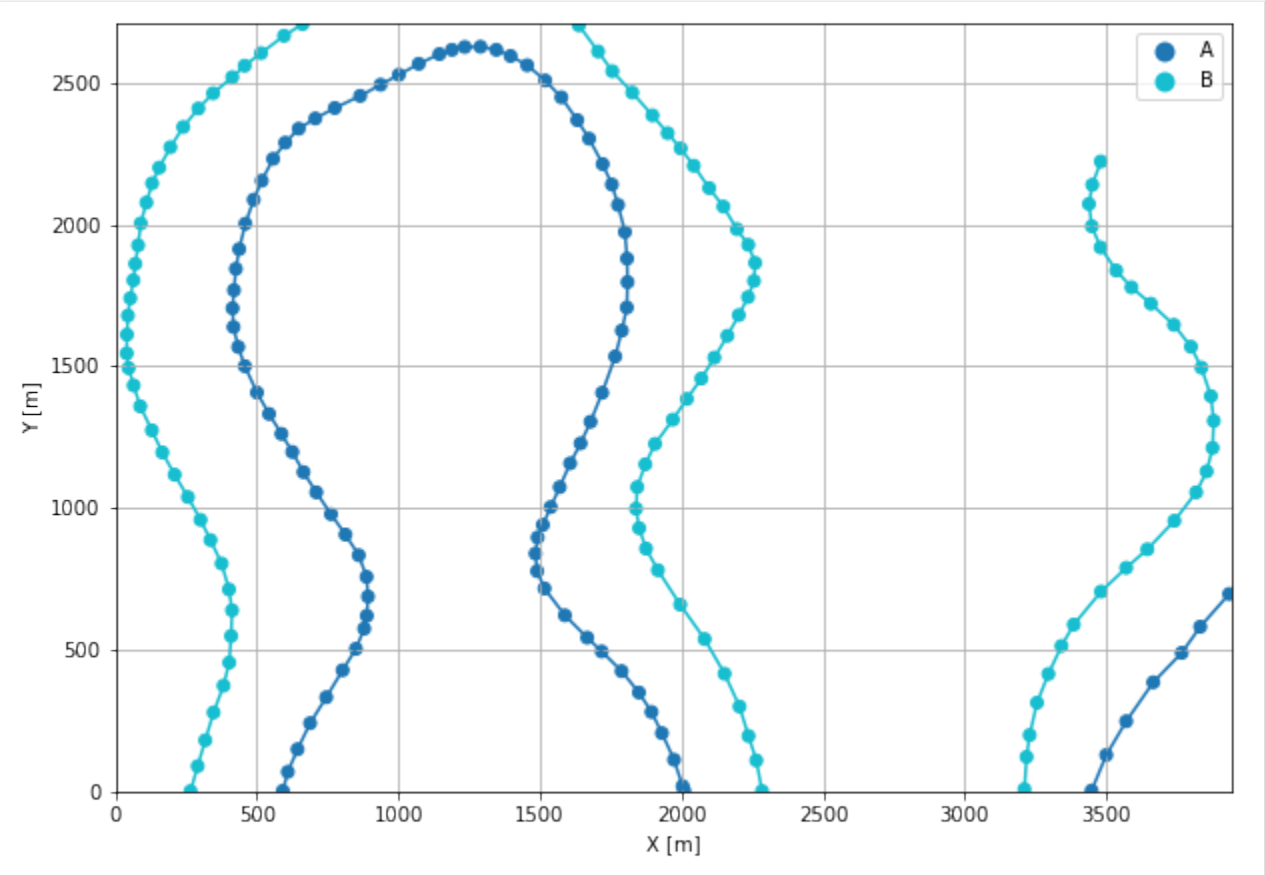
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords
```

[12]:
```
    formation                   geometry       X       Y      Z
94          B    POINT (412.148 639.154)  412.15  639.15 538.20
129         B   POINT (2204.588 299.772) 2204.59  299.77 578.98
120         B   POINT (345.109 2463.437)  345.11 2463.44 588.11
121         B   POINT (412.986 2520.420)  412.99 2520.42 547.54
122         B   POINT (456.561 2560.643)  456.56 2560.64 518.60
..        ...                       ...     ...     ...    ...
61          A  POINT (1643.141 1228.254) 1643.14 1228.25 436.22
62          A  POINT (1606.270 1157.864) 1606.27 1157.86 419.63
63          A  POINT (1569.399 1074.066) 1569.40 1074.07 402.09
64          A  POINT (1537.555 1003.675) 1537.56 1003.68 390.91
0           A      POINT (591.475 2.289)  591.48    2.29 641.21

[188 rows x 5 columns]
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3942)
ax.set_ylim(0, 2710)
```

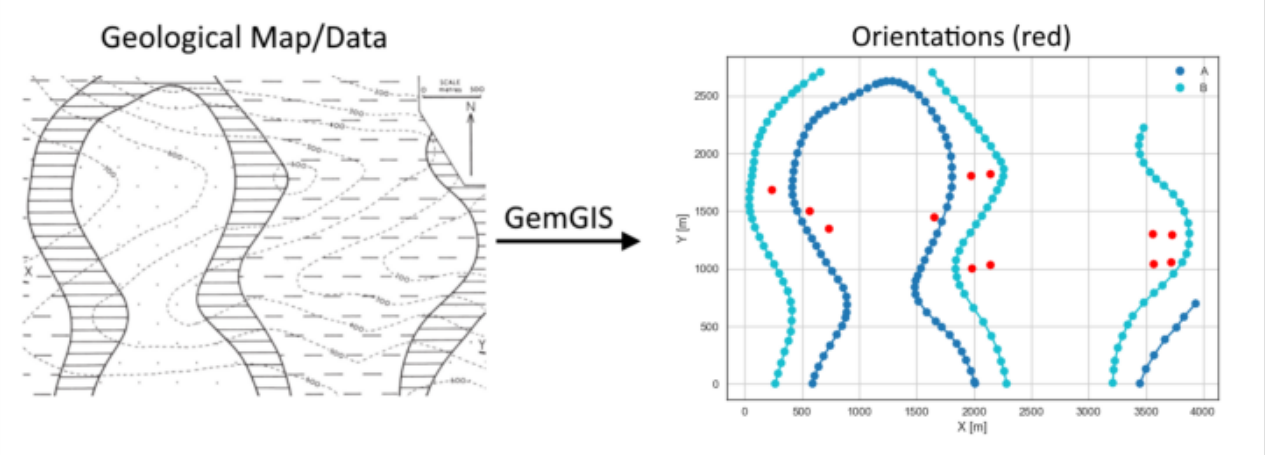
[13]:
```
(0.0, 2710.0)
```



7.5.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example05.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes5.shp')
strikes
```

[15]:

	id	formation	Z	geometry
0	2	B1	700	LINESTRING (149.860 2191.094, 154.888 1214.846)
1	1	B1	600	LINESTRING (319.551 2436.622, 321.855 916.106)
2	3	A1	700	LINESTRING (481.281 2067.910, 488.404 1433.140)
3	2	A1	600	LINESTRING (636.098 2326.846, 655.529 174.023)
4	1	A1	500	LINESTRING (807.413 2426.933, 822.287 456.422)
5	2	A2	500	LINESTRING (1725.629 2204.030, 1737.885 473.391)
6	1	A2	400	LINESTRING (1556.253 2469.041, 1567.356 648.529)
7	1	B2	400	LINESTRING (1890.816 1193.007, 1900.453 810.050)
8	2	B2	500	LINESTRING (2062.183 564.521, 2056.317 1437.278)
9	3	B2	600	LINESTRING (2216.791 1955.150, 2236.483 196.648)
10	2	B3	500	LINESTRING (2056.527 1437.278, 2055.479 2190.203)
11	3	B3	600	LINESTRING (2216.581 1954.312, 2220.352 1718.839)
12	1	B3	400	LINESTRING (1877.199 2407.659, 1890.816 1192.692)
13	1	B4	200	LINESTRING (3799.212 1567.165, 3810.106 1043.427)
14	2	B4	300	LINESTRING (3636.644 1737.275, 3642.509 847.340)
15	3	B4	400	LINESTRING (3469.048 1945.932, 3469.886 688.961)
16	3	A	600	LINESTRING (3487.483 96.509, 3487.902 1969.395)
17	2	A	500	LINESTRING (3647.118 355.445, 3637.063 1740.627)
18	1	A	400	LINESTRING (3808.639 548.600, 3790.413 1580.992)

Calculate Orientations for each formation

[16]:
```
orientations_a = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'A'].sort_values(by='Z', ascending=True).reset_index())
orientations_a
```

[16]:

	dip	azimuth	Z	geometry	polarity	formation
0	33.31	89.37	450.00	POINT (3720.808 1056.416)	1.00	A
1	33.83	89.86	550.00	POINT (3564.892 1040.494)	1.00	A

	X	Y
0	3720.81	1056.42
1	3564.89	1040.49

[17]:
```
orientations_a1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'A1'].sort_values(by='Z', ascending=True).reset_index())
orientations_a1
```

[17]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 30.57    89.52 550.00  POINT (730.331 1346.056)      1.00        A1 730.33
1 32.70    89.47 650.00  POINT (565.328 1500.480)      1.00        A1 565.33

        Y
0 1346.06
1 1500.48
```

[18]:
```
orientations_a2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'A2'].sort_values(by='Z', ascending=True).reset_index())
orientations_a2
```

[18]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 30.80   269.62 450.00  POINT (1646.781 1448.748)      1.00        A2

        X       Y
0 1646.78 1448.75
```

[19]:
```
orientations_b1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B1'].sort_values(by='Z', ascending=True).reset_index())
orientations_b1
```

[19]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 30.99    89.85 650.00  POINT (236.538 1689.667)      1.00        B1 236.54

        Y
0 1689.67
```

[20]:
```
orientations_b2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B2'].sort_values(by='Z', ascending=True).reset_index())
orientations_b2
```

[20]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 31.99   269.44 450.00  POINT (1977.443 1001.214)      1.00        B2
1 31.03   269.41 550.00  POINT (2142.944 1038.399)      1.00        B2

        X       Y
0 1977.44 1001.21
1 2142.94 1038.40
```

[21]:
```
orientations_b3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B3'].sort_values(by='Z', ascending=True).reset_index())
orientations_b3
```

[21]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 30.70   269.51 450.00  POINT (1970.005 1806.958)      1.00        B3
1 31.88   269.85 550.00  POINT (2137.235 1825.158)      1.00        B3

        X       Y
0 1970.01 1806.96
```

(continues on next page)

(continued from previous page)

```
1 2137.23 1825.16
```

[22]:
```
orientations_b4 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B4'].sort_values(by='Z', ascending=True).reset_index())
orientations_b4
```

[22]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 31.77    89.41 250.00  POINT (3722.118 1298.802)      1.00        B4
1 30.84    89.85 350.00  POINT (3554.522 1304.877)      1.00        B4

        X       Y
0 3722.12 1298.80
1 3554.52 1304.88
```

Merging Orientations

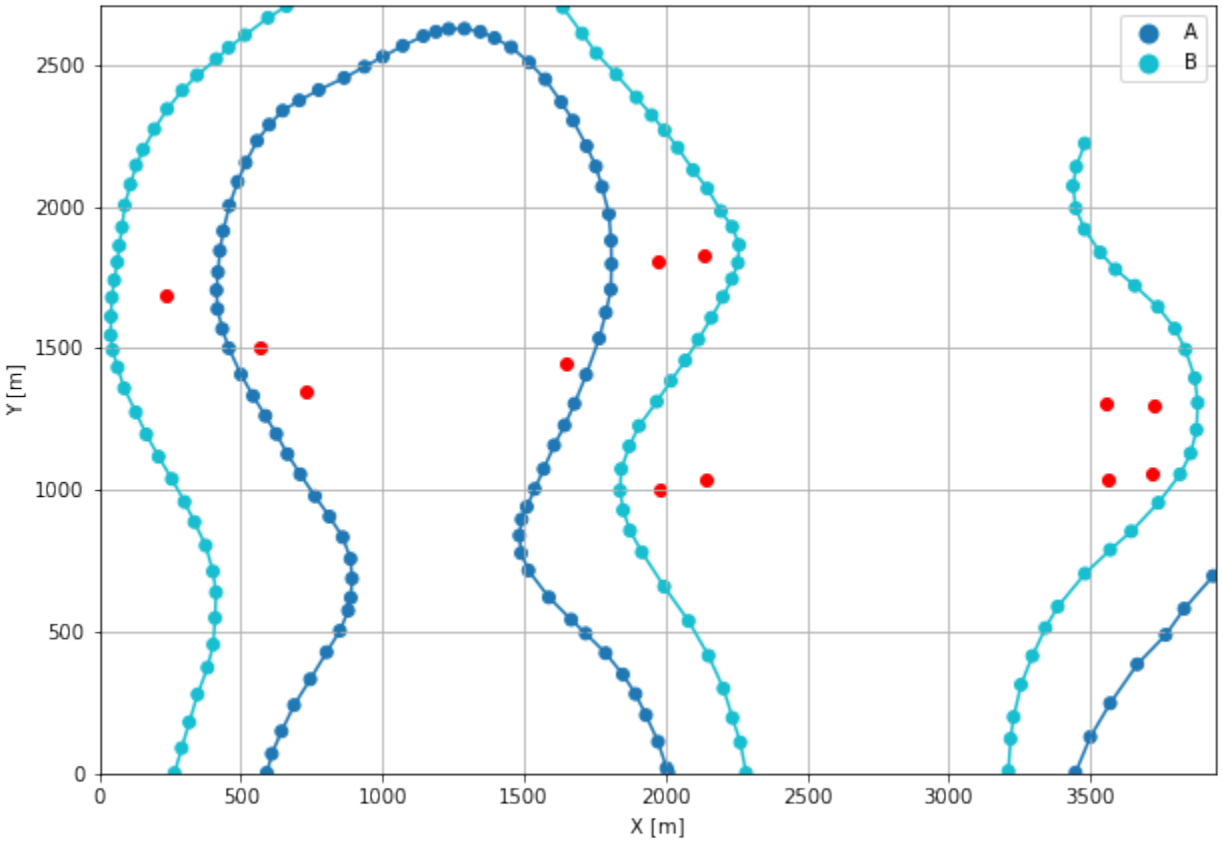
[23]:
```
import pandas as pd
orientations = pd.concat([orientations_a, orientations_a1, orientations_a2, orientations_
↪b1, orientations_b2, orientations_b3, orientations_b4]).reset_index()
orientations['formation'] = ['A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B', 'B', 'B', 'B']
orientations
```

[23]:
```
    index   dip  azimuth      Z                   geometry  polarity  \
0       0 33.31    89.37 450.00  POINT (3720.808 1056.416)      1.00
1       1 33.83    89.86 550.00  POINT (3564.892 1040.494)      1.00
2       0 30.57    89.52 550.00   POINT (730.331 1346.056)      1.00
3       1 32.70    89.47 650.00   POINT (565.328 1500.480)      1.00
4       0 30.80   269.62 450.00  POINT (1646.781 1448.748)      1.00
5       0 30.99    89.85 650.00   POINT (236.538 1689.667)      1.00
6       0 31.99   269.44 450.00  POINT (1977.443 1001.214)      1.00
7       1 31.03   269.41 550.00  POINT (2142.944 1038.399)      1.00
8       0 30.70   269.51 450.00  POINT (1970.005 1806.958)      1.00
9       1 31.88   269.85 550.00  POINT (2137.235 1825.158)      1.00
10      0 31.77    89.41 250.00  POINT (3722.118 1298.802)      1.00
11      1 30.84    89.85 350.00  POINT (3554.522 1304.877)      1.00

   formation       X       Y
0          A 3720.81 1056.42
1          A 3564.89 1040.49
2          A  730.33 1346.06
3          A  565.33 1500.48
4          A 1646.78 1448.75
5          B  236.54 1689.67
6          B 1977.44 1001.21
7          B 2142.94 1038.40
8          B 1970.01 1806.96
9          B 2137.23 1825.16
10         B 3722.12 1298.80
11         B 3554.52 1304.88
```

Plotting the Orientations

[24]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3942)
ax.set_ylim(0, 2710)
```

[24]:
```
(0.0, 2710.0)
```



7.5.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[25]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
```

(continues on next page)

(continued from previous page)

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[26]:
```
geo_model = gp.create_model('Model5')
geo_model
```

[26]:
```
Model5  2022-04-04 20:51
```

Initiate Data

[27]:
```
gp.init_data(geo_model, [0, 3942, 0, 2710, -200, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[27]:
```
Model5  2022-04-04 20:51
```

Model Surfaces

[28]:
```
geo_model.surfaces
```

[28]:

	surface	series	order_surfaces	color	id
0	B	Default series	1	#015482	1
1	A	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[29]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('A', 'B'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[29]:

	surface	series	order_surfaces	color	id
0	B	Strata1	1	#015482	1
1	A	Strata1	2	#9f0052	2
2	Basement	Strata1	3	#ffbe00	3

Showing the Number of Data Points

[30]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[30]:
```
    surface   series  order_surfaces    color  id  No. of Interfaces  No. of Orientations
0         B  Strata1               1  #015482   1                101                    7
1         A  Strata1               2  #9f0052   2                 87                    5
2  Basement  Strata1               3  #ffbe00   3                  0                    0
```

Loading Digital Elevation Model

[31]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster5.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[31]:
```
Grid Object. Values:
array([[  19.71      ,   13.55      , -194.        ],
       [  19.71      ,   13.55      , -182.        ],
       [  19.71      ,   13.55      , -170.        ],
       ...,
       [3937.01012658, 2685.        ,  182.10906982],
       [3937.01012658, 2695.        ,  181.02072144],
       [3937.01012658, 2705.        ,  179.95965576]])
```

Defining Custom Section

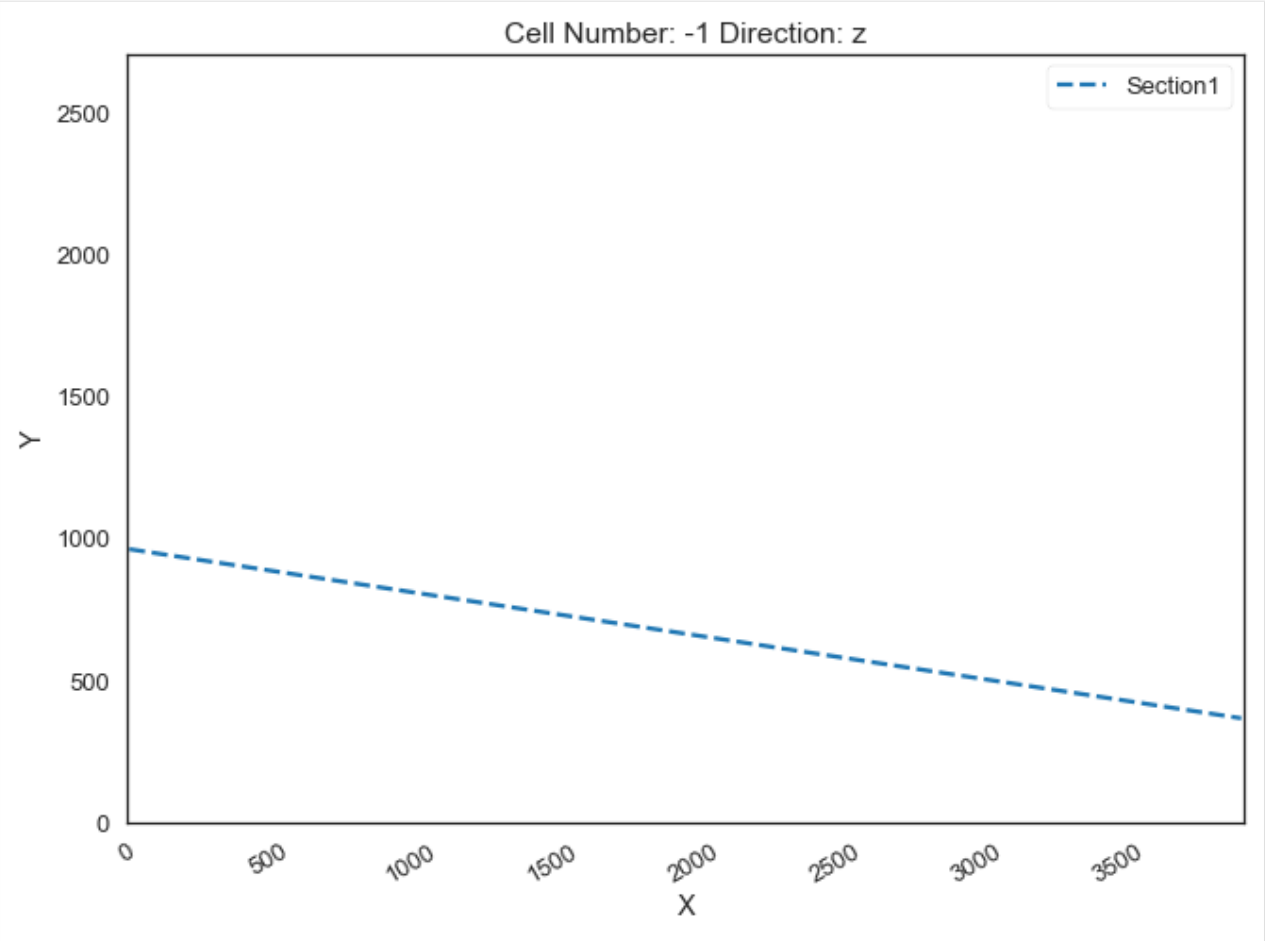
[32]:
```
custom_section = gpd.read_file(file_path + 'customsection5.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='name')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[32]:
```
                                          start                                       stop␣
↪resolution    dist
Section1  [6.565213969151387, 963.4523850507635]  [3931.665030709532, 366.8103975359992]␣
↪ [100, 80] 3970.19
```

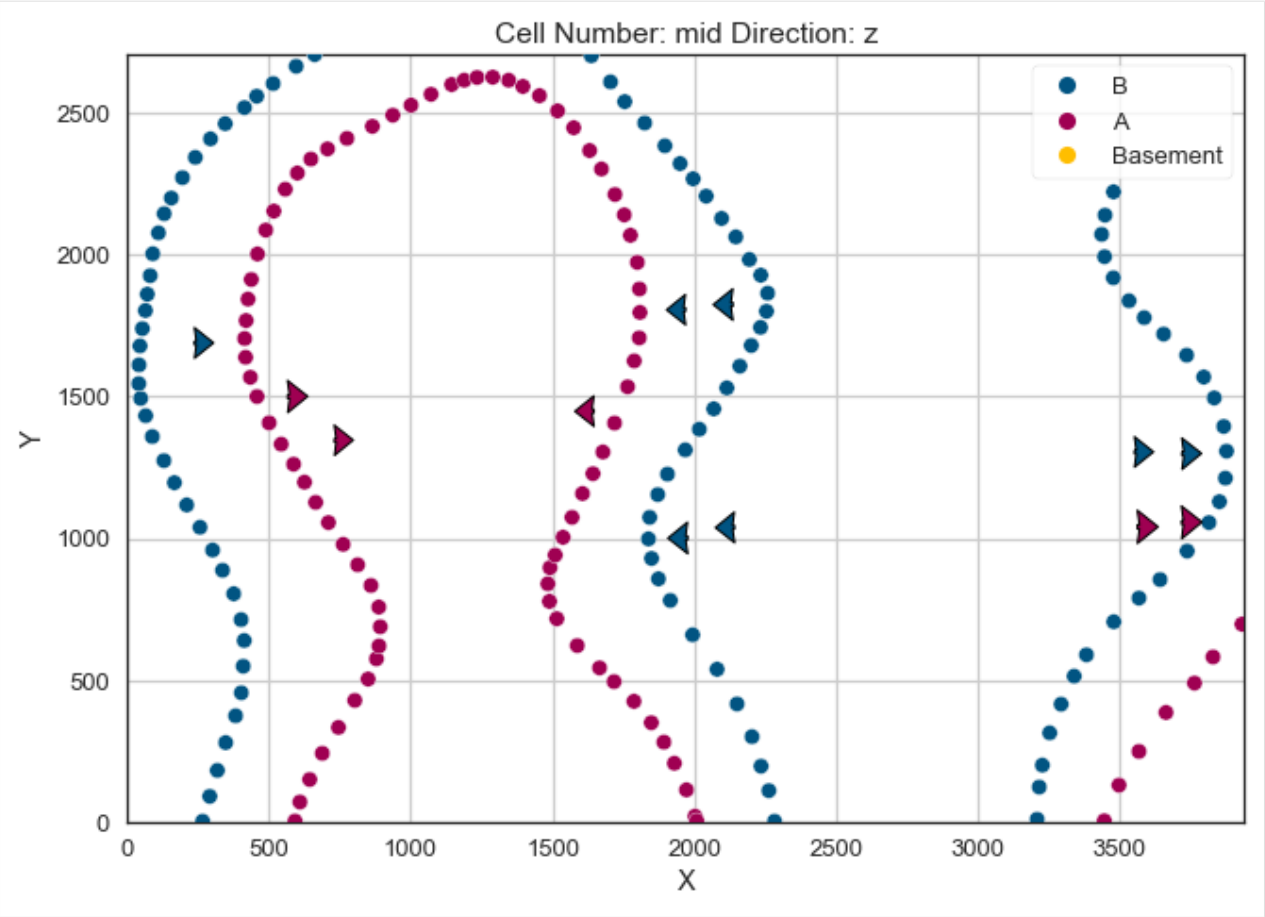
[33]:
```
gp.plot.plot_section_traces(geo_model)
```

[33]:
```
<gempy.plot.visualization_2d.Plot2D at 0x2388552d640>
```

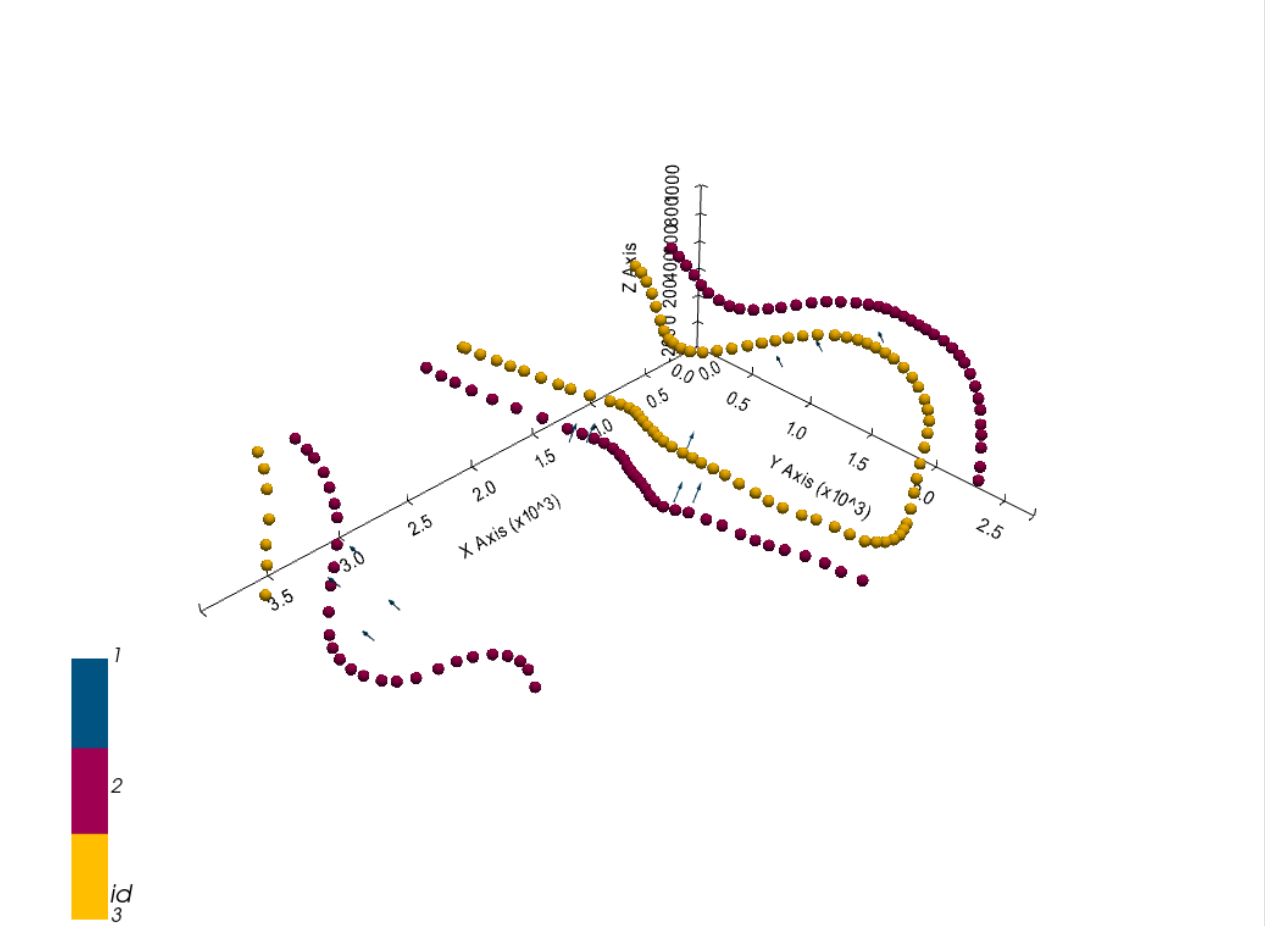


Plotting Input Data

[34]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[35]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[35]: <gempy.plot.vista.GemPyToVista at 0x238870ec850>

Setting the Interpolator

[36]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range            4931.88
$C_o$           579130.1
drift equations      [3]
```

[36]:
```
<gempy.core.interpolator.InterpolatorModel at 0x238855398e0>
```

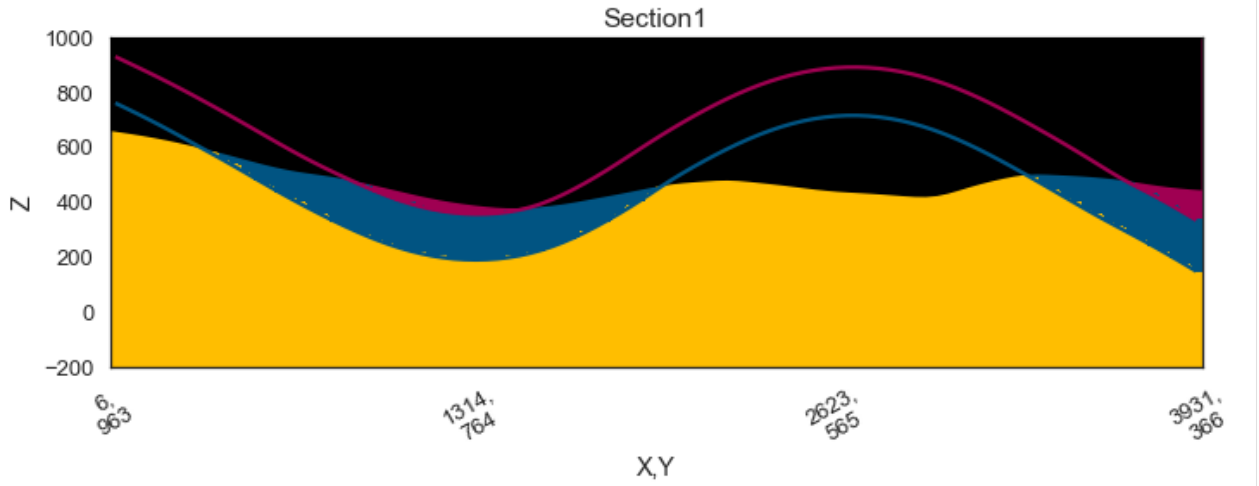
Computing Model

[37]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

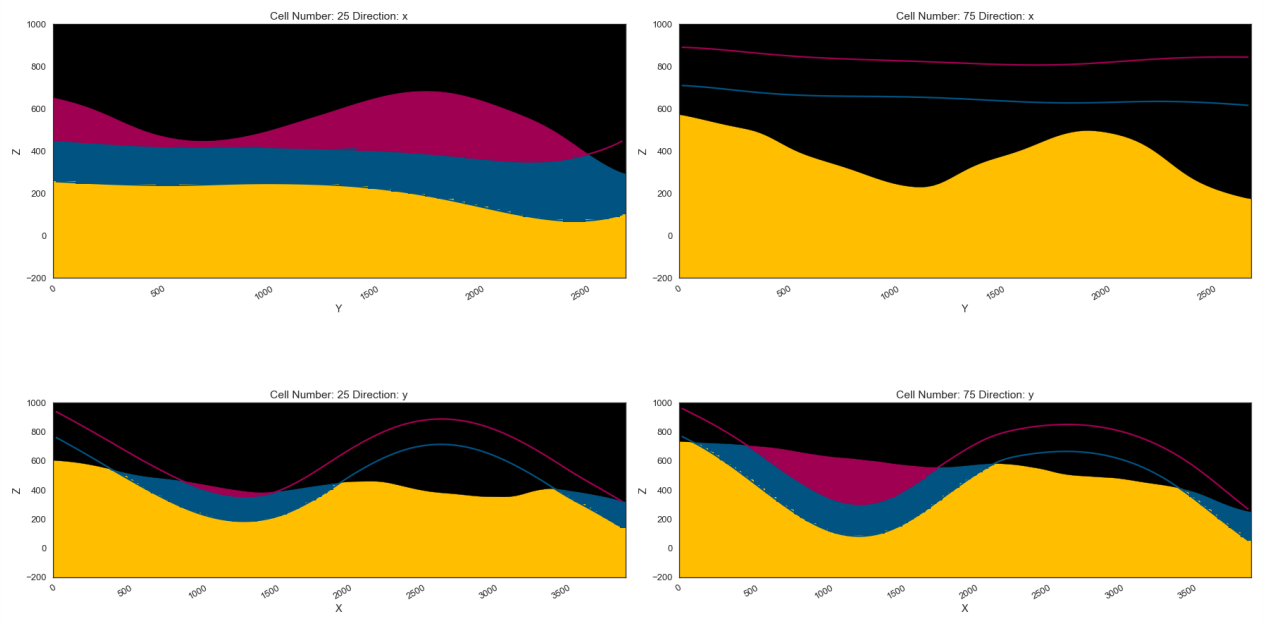
[38]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[38]:
```
<gempy.plot.visualization_2d.Plot2D at 0x2388ccabf40>
```



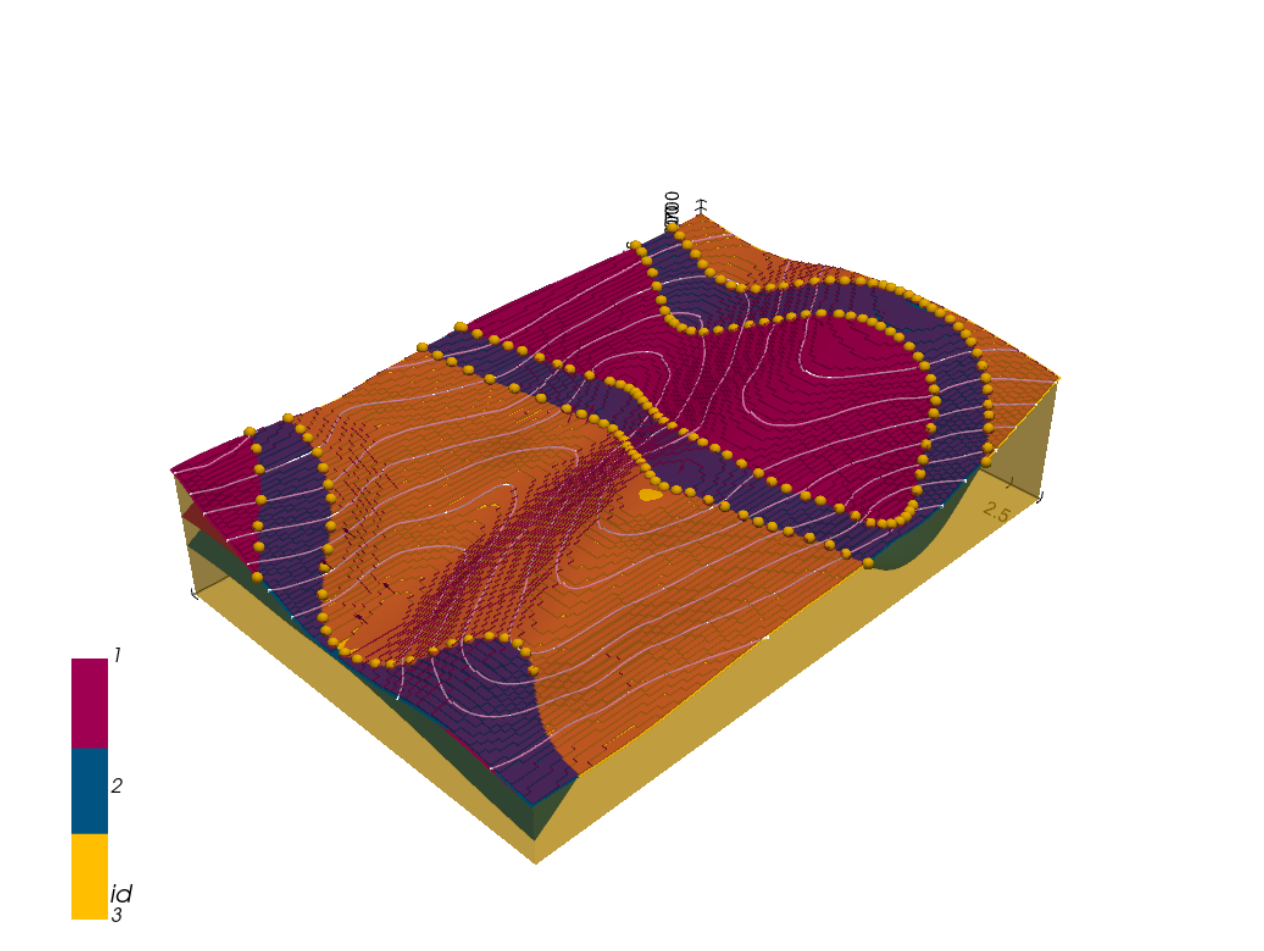
[39]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[39]:
```
<gempy.plot.visualization_2d.Plot2D at 0x2388cff4ee0>
```



Plotting 3D Model

[40]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



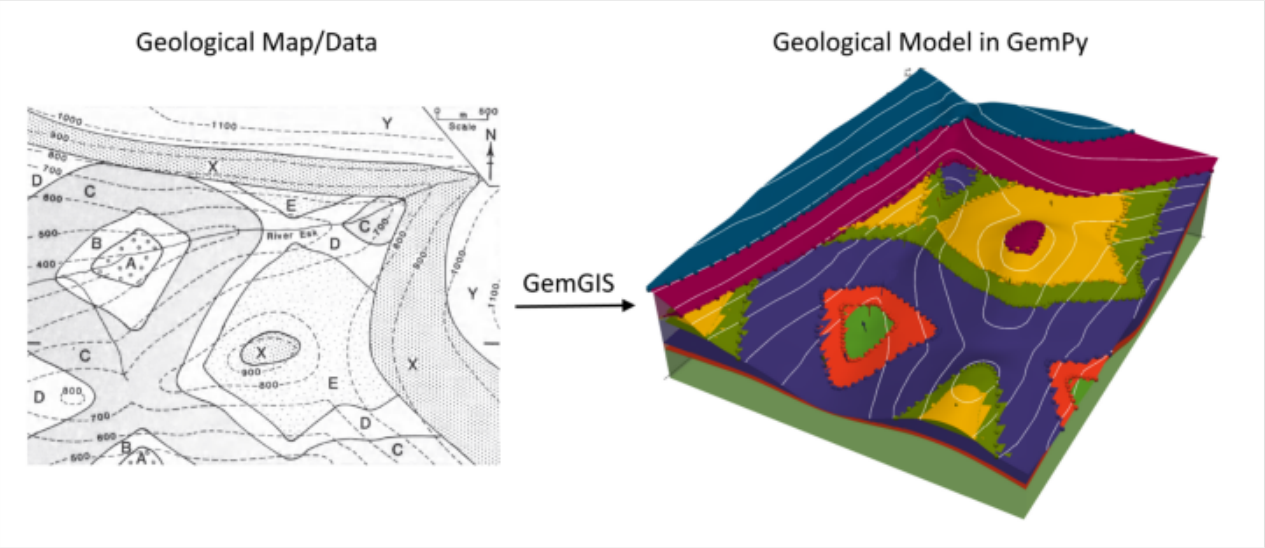
[ ]:

7.6 Example 6 - Folded Unconformable Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 4642 m wide (W-E extent) and 3519 m high (N-S extent). The vertical model extent varies from 0 m to 1500 m. The model represents folded layers (yellow to light green) which are separated to a second set of layers (blue and purple) by an unconformity. The light green layer also represents the basement. The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example06.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.6.1 Licensing

7.6.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.6.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example06/'
gg.download_gemgis_data.download_tutorial_data(filename="example06_folded_unconformable_
↪layers.zip", dirpath=file_path)
```

```
Downloading file 'example06_folded_unconformable_layers.zip' from 'https://rwth-aachen.
↪sciebo.de/s/AfXRsZywYDbUF34/download?path=%2Fexample06_folded_unconformable_layers.zip
↪' to 'C:\Users\ale93371\Documents\gemgis\docs\getting_started\example\data\example06'.
```

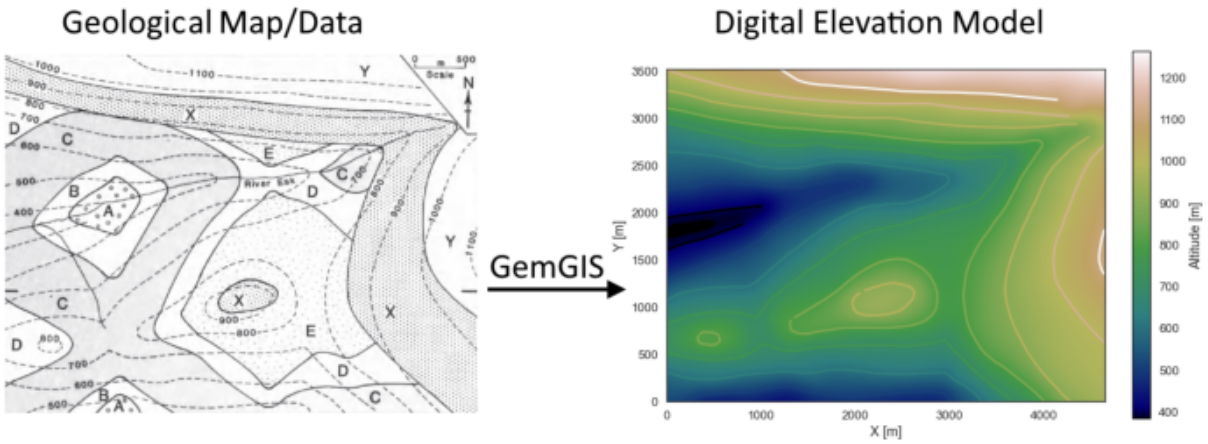
(continues on next page)

(continued from previous page)

7.6.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example06.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo6.shp')
topo.head()
```

[6]:
```
     id     Z                                           geometry
0  None  1100  LINESTRING (1206.918 3514.485, 1274.374 3468.7...
1  None  1000  LINESTRING (3.594 3507.957, 151.562 3458.998, ...
2  None   900  LINESTRING (2.506 3289.270, 195.081 3252.278, ...
3  None   800  LINESTRING (4.682 3102.135, 143.946 3077.111, ...
4  None  1100  LINESTRING (4639.548 1815.034, 4610.172 1701.8...
```

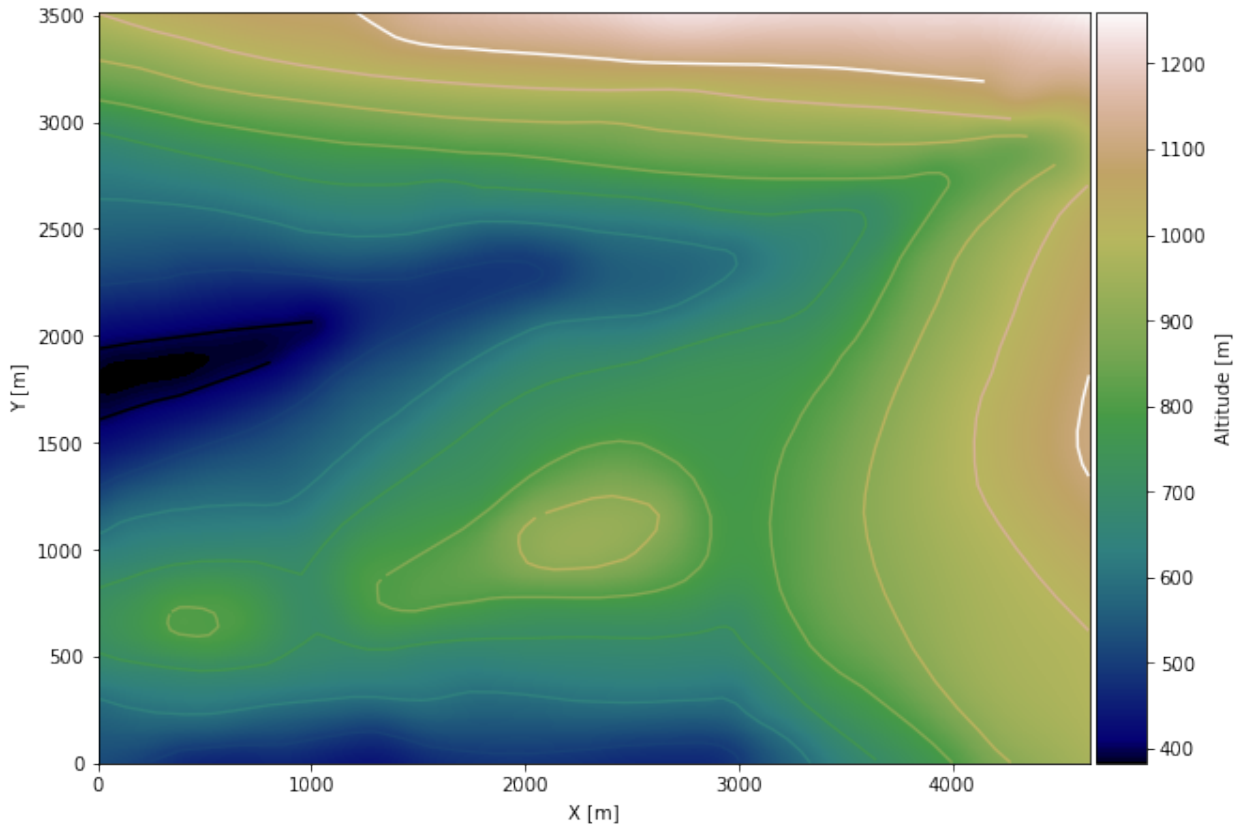
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable
fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 4642, 0, 3519], cmap='gist_earth
↪')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4642)
ax.set_ylim(0, 3519)
```

[8]:
```
(0.0, 3519.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster6.tif', extent = [0, 4642, 0, 3519], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

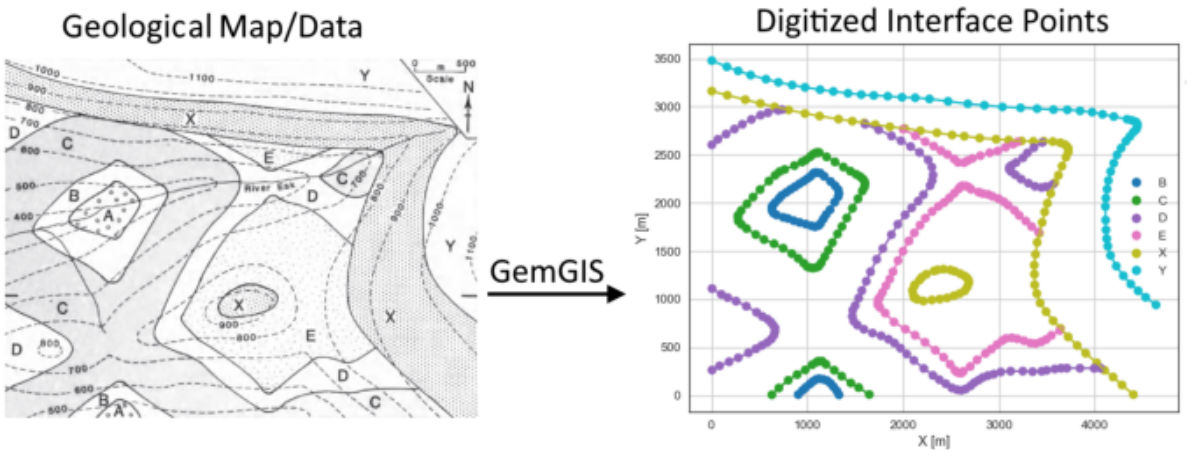
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster6.tif')
```

7.6.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example06.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces6.shp')
interfaces.head()
```

[11]:
```
     id formation                                           geometry
0  None         B  LINESTRING (906.087 7.328, 949.607 50.304, 984...
1  None         C  LINESTRING (631.368 5.152, 704.264 75.871, 802...
2  None         D  LINESTRING (6.858 261.919, 85.194 305.439, 185...
```

(continues on next page)

(continued from previous page)

```
3  None         B  LINESTRING (1097.031 2316.057, 1135.111 2310.6...
4  None         C  LINESTRING (1125.319 2522.776, 1202.566 2478.1...
```

Extracting Z coordinate from Digital Elevation Model

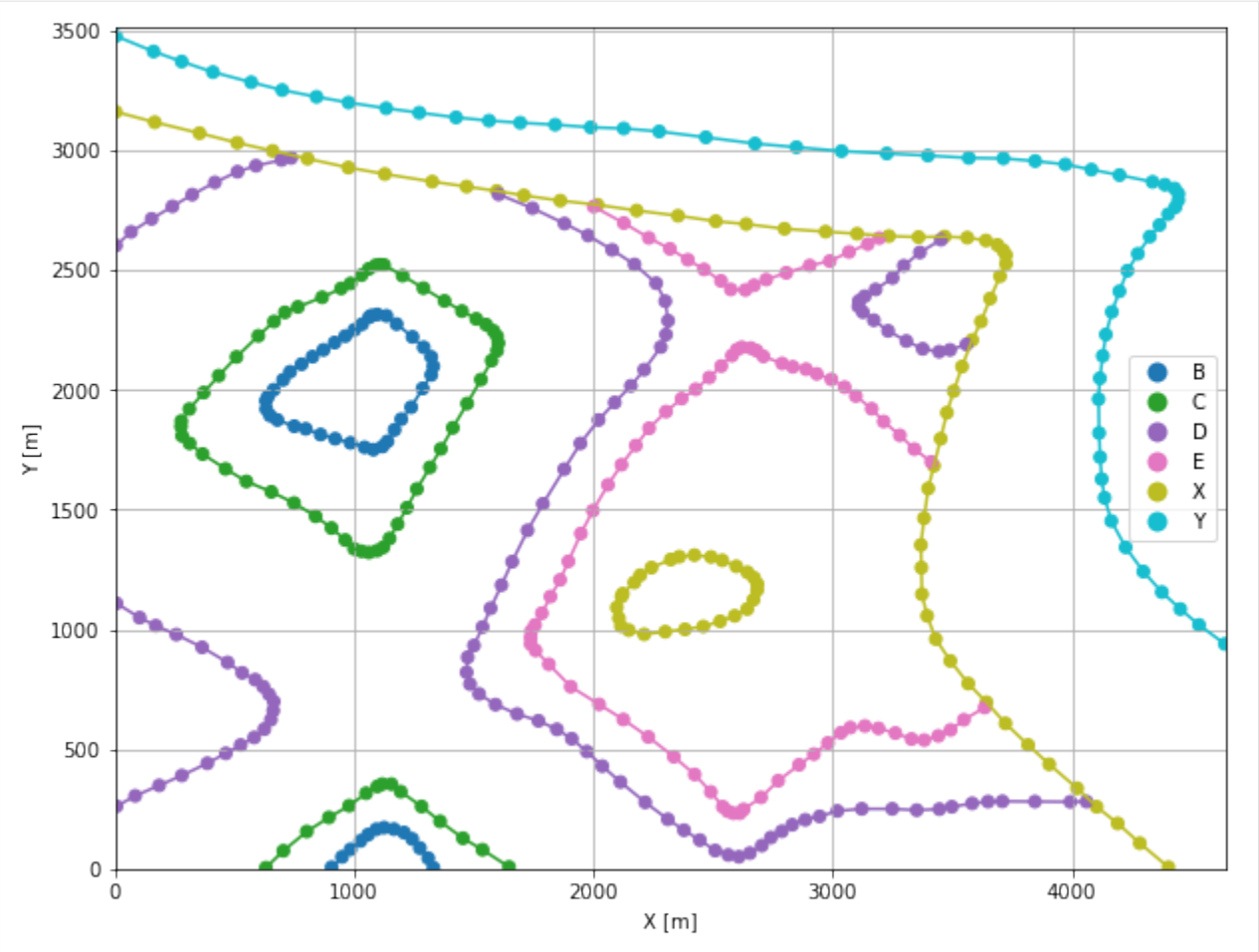
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

[12]:
```
    formation                  geometry       X       Y       Z
453         Y   POINT (4639.004 939.741) 4639.00  939.74 1040.37
411         Y  POINT (1988.644 3097.783) 1988.64 3097.78  954.71
423         Y  POINT (3972.606 2942.743) 3972.61 2942.74  928.61
422         Y  POINT (3846.399 2956.887) 3846.40 2956.89  936.75
421         Y  POINT (3713.663 2967.767) 3713.66 2967.77  938.66
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4642)
ax.set_ylim(0, 3519)
```

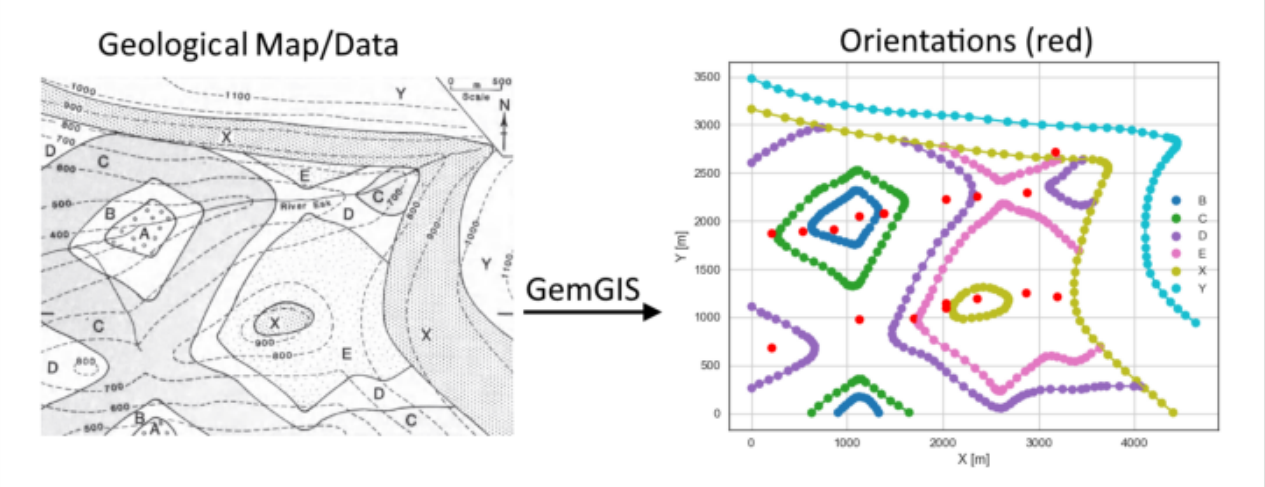
[13]:
```
(0.0, 3519.0)
```



7.6.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for GemPy. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example06.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes6.shp')
strikes.head()
```

[15]:

	id	formation	Z	geometry
0	1	B	500	LINESTRING (1048.887 2285.049, 1046.711 1758.459)
1	2	B	500	LINESTRING (1196.855 2262.201, 1196.855 1875.962)
2	1	C1	500	LINESTRING (1529.782 2283.961, 1539.573 2067.450)
3	2	C	500	LINESTRING (685.496 2305.721, 697.464 1551.739)
4	1	C	400	LINESTRING (383.849 1996.186, 378.953 1721.467)

Calculate Orientations for each formation

[16]:
```
orientations_b = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B'].sort_values(by='Z', ascending=True).reset_index())
orientations_b
```

[16]:

	dip	azimuth	Z	geometry	polarity	formation	X	Y
0	0.00	0.00	500.00	POINT (1122.327 2045.418)	1.00	B	1122.33	2045.42

[17]:
```
orientations_b1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B1'].sort_values(by='Z', ascending=True).reset_index())
orientations_b1
```

[17]:

	dip	azimuth	Z	geometry	polarity	formation	X	Y
0	0.00	0.00	500.00	POINT (1123.143 973.537)	1.00	B1	1123.14	973.54

[18]:
```
orientations_c = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C'].sort_values(by='Z', ascending=True).reset_index())
orientations_c
```

[18]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 18.07   269.32 450.00  POINT (536.440 1893.778)      1.00         C 536.44
1 16.77   269.96 550.00  POINT (861.343 1916.762)      1.00         C 861.34

        Y
0 1893.78
1 1916.76
```

[19]:
```
orientations_c1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C1'].sort_values(by='Z', ascending=True).reset_index())
orientations_c1
```

[19]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 18.26    90.30 550.00  POINT (1379.502 2078.602)      1.00        C1

        X       Y
0 1379.50 2078.60
```

[20]:
```
orientations_c2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C1'].sort_values(by='Z', ascending=True).reset_index())
orientations_c2
```

[20]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 18.26    90.30 550.00  POINT (1379.502 2078.602)      1.00        C1

        X       Y
0 1379.50 2078.60
```

[21]:
```
orientations_d = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D'].sort_values(by='Z', ascending=True).reset_index())
orientations_d
```

[21]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 17.00   270.73 650.00  POINT (211.673 1877.322)      1.00         D 211.67

        Y
0 1877.32
```

[22]:
```
orientations_d1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D1'].sort_values(by='Z', ascending=True).reset_index())
orientations_d1
```

[22]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 16.79   268.97 650.00  POINT (212.217 684.878)      1.00        D1 212.22

       Y
0 684.88
```

[23]:
```
orientations_d2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D2'].sort_values(by='Z', ascending=True).reset_index())
orientations_d2
```

[23]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 18.25    90.76 650.00  POINT (2034.068 2228.337)      1.00        D2
```

(continues on next page)

(continued from previous page)

```
        X       Y
0 2034.07 2228.34
```

[24]:
```
orientations_d3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D3'].sort_values(by='Z', ascending=True).reset_index())
orientations_d3
```

[24]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 17.12    90.05 650.00  POINT (2027.540 1148.092)      1.00        D3
1 17.07    90.49 750.00   POINT (1695.973 984.077)      1.00        D3

        X       Y
0 2027.54 1148.09
1 1695.97  984.08
```

[25]:
```
orientations_e = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'E'].sort_values(by='Z', ascending=True).reset_index())
orientations_e
```

[25]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 17.03    90.29 650.00  POINT (2358.563 1194.060)      1.00         E
1 18.62    90.97 750.00  POINT (2033.796 1095.868)      1.00         E

        X       Y
0 2358.56 1194.06
1 2033.80 1095.87
```

[26]:
```
orientations_e1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'E1'].sort_values(by='Z', ascending=True).reset_index())
orientations_e1
```

[26]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 17.56   270.33 650.00  POINT (2862.034 1255.124)      1.00        E1
1 17.18   270.30 750.00  POINT (3188.841 1215.140)      1.00        E1

        X       Y
0 2862.03 1255.12
1 3188.84 1215.14
```

[27]:
```
orientations_e2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'E2'].sort_values(by='Z', ascending=True).reset_index())
orientations_e2
```

[27]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 16.52    88.10 650.00  POINT (2354.483 2258.121)      1.00        E2

        X       Y
0 2354.48 2258.12
```

[28]:
```
orientations_e3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'E3'].sort_values(by='Z', ascending=True).reset_index())
orientations_e3
```

[28]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 16.77   271.06 650.00  POINT (2872.642 2298.377)      1.00        E3

        X       Y
0 2872.64 2298.38
```

[29]:
```
orientations_y = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'Y'].sort_values(by='Z', ascending=True).reset_index())
orientations_y
```

[29]:
```
   dip  azimuth      Z                   geometry  polarity formation       X  \
0 5.08    23.65 950.00  POINT (3167.489 2716.440)      1.00         Y 3167.49

        Y
0 2716.44
```

Merging Orientations

[30]:
```
import pandas as pd
orientations = pd.concat([orientations_b, orientations_b1, orientations_c, orientations_
↪c1, orientations_c2, orientations_d, orientations_d1, orientations_d2, orientations_d3,
↪ orientations_e, orientations_e1, orientations_e2, orientations_e3, orientations_y]).
↪reset_index()
orientations['formation'] = ['B', 'B', 'C', 'C', 'C', 'C', 'D', 'D', 'D', 'D', 'D', 'E',
↪'E', 'E', 'E', 'E', 'E', 'Y']
orientations
```

[30]:
```
    index   dip  azimuth      Z                   geometry  polarity  \
0       0  0.00     0.00 500.00  POINT (1122.327 2045.418)      1.00
1       0  0.00     0.00 500.00   POINT (1123.143 973.537)      1.00
2       0 18.07   269.32 450.00   POINT (536.440 1893.778)      1.00
3       1 16.77   269.96 550.00   POINT (861.343 1916.762)      1.00
4       0 18.26    90.30 550.00  POINT (1379.502 2078.602)      1.00
5       0 18.26    90.30 550.00  POINT (1379.502 2078.602)      1.00
6       0 17.00   270.73 650.00   POINT (211.673 1877.322)      1.00
7       0 16.79   268.97 650.00    POINT (212.217 684.878)      1.00
8       0 18.25    90.76 650.00  POINT (2034.068 2228.337)      1.00
9       0 17.12    90.05 650.00  POINT (2027.540 1148.092)      1.00
10      1 17.07    90.49 750.00   POINT (1695.973 984.077)      1.00
11      0 17.03    90.29 650.00  POINT (2358.563 1194.060)      1.00
12      1 18.62    90.97 750.00  POINT (2033.796 1095.868)      1.00
13      0 17.56   270.33 650.00  POINT (2862.034 1255.124)      1.00
14      1 17.18   270.30 750.00  POINT (3188.841 1215.140)      1.00
15      0 16.52    88.10 650.00  POINT (2354.483 2258.121)      1.00
16      0 16.77   271.06 650.00  POINT (2872.642 2298.377)      1.00
17      0  5.08    23.65 950.00  POINT (3167.489 2716.440)      1.00

   formation       X       Y
0          B 1122.33 2045.42
1          B 1123.14  973.54
2          C  536.44 1893.78
3          C  861.34 1916.76
```

(continues on next page)

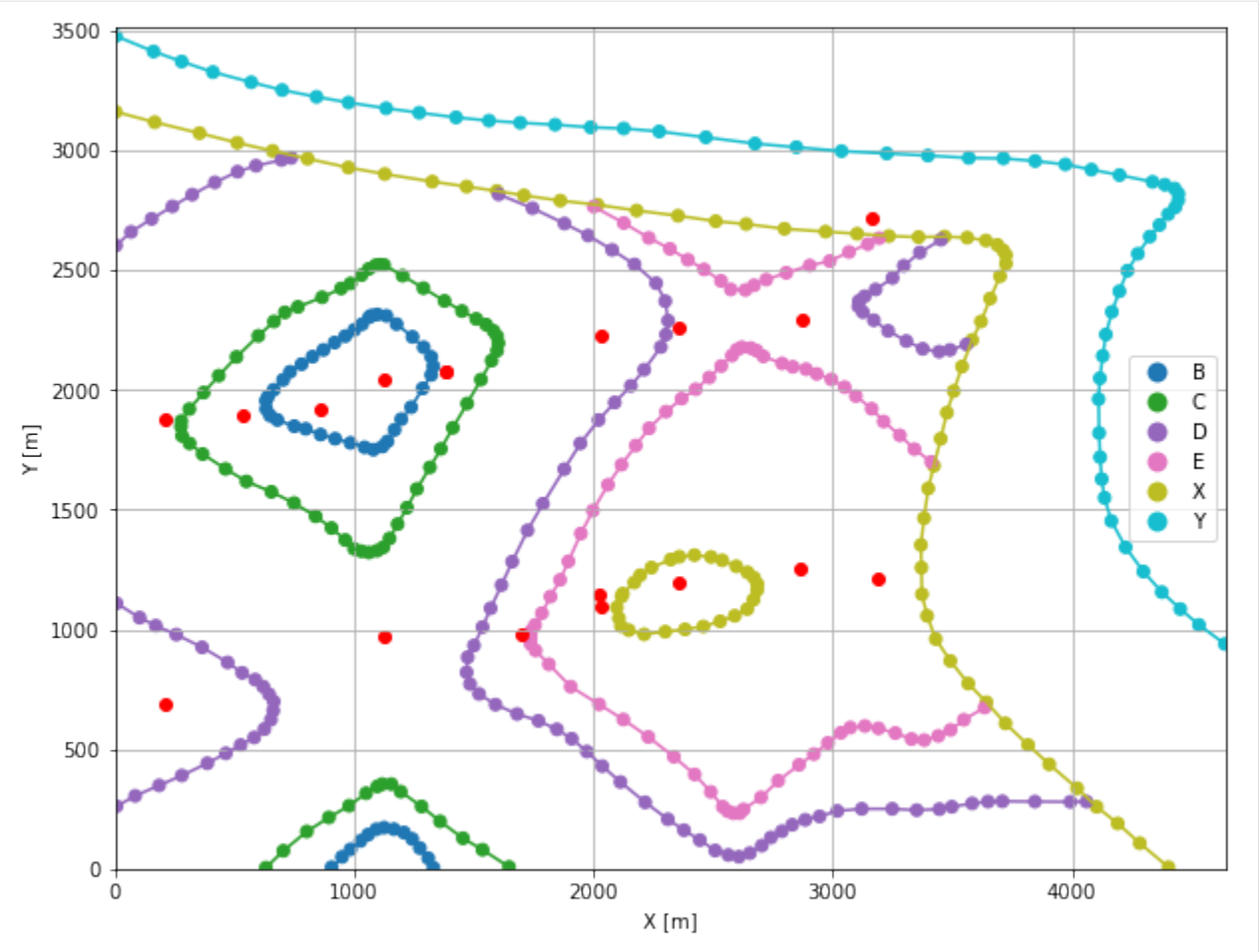
(continued from previous page)

```
4          C 1379.50 2078.60
5          C 1379.50 2078.60
6          D  211.67 1877.32
7          D  212.22  684.88
8          D 2034.07 2228.34
9          D 2027.54 1148.09
10         D 1695.97  984.08
11         E 2358.56 1194.06
12         E 2033.80 1095.87
13         E 2862.03 1255.12
14         E 3188.84 1215.14
15         E 2354.48 2258.12
16         E 2872.64 2298.38
17         Y 3167.49 2716.44
```

Plotting the Orientations

[31]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4642)
ax.set_ylim(0, 3519)
```

[31]:
```
(0.0, 3519.0)
```



7.6.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[32]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[33]:
```
geo_model = gp.create_model('Model6')
geo_model
```

[33]:
```
Model6  2022-04-05 08:26
```

Initiate Data

[34]:
```
gp.init_data(geo_model, [0, 4642, 0, 3519, 0, 1500], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[34]:
```
Model6  2022-04-05 08:26
```

Model Surfaces

[35]:
```
geo_model.surfaces
```

[35]:

	surface	series	order_surfaces	color	id
0	Y	Default series	1	#015482	1
1	X	Default series	2	#9f0052	2
2	E	Default series	3	#ffbe00	3
3	D	Default series	4	#728f02	4
4	C	Default series	5	#443988	5
5	B	Default series	6	#ff3f20	6

Mapping the Stack to Surfaces

[36]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Y', 'X'),
                          'Strata2': ('E', 'D', 'C', 'B'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('A')
```

[36]:

	surface	series	order_surfaces	color	id
0	Y	Strata1	1	#015482	1
1	X	Strata1	2	#9f0052	2
2	E	Strata2	1	#ffbe00	3
3	D	Strata2	2	#728f02	4
4	C	Strata2	3	#443988	5
5	B	Strata2	4	#ff3f20	6
6	A	Strata2	5	#5DA629	7

Showing the Number of Data Points

[37]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[37]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Y	Strata1	1	#015482	1	57	1
1	X	Strata1	2	#9f0052	2	84	0
2	E	Strata2	1	#ffbe00	3	84	6
3	D	Strata2	2	#728f02	4	110	5
4	C	Strata2	3	#443988	5	68	4
5	B	Strata2	4	#ff3f20	6	51	2
6	A	Strata2	5	#5DA629	7	0	0

Loading Digital Elevation Model

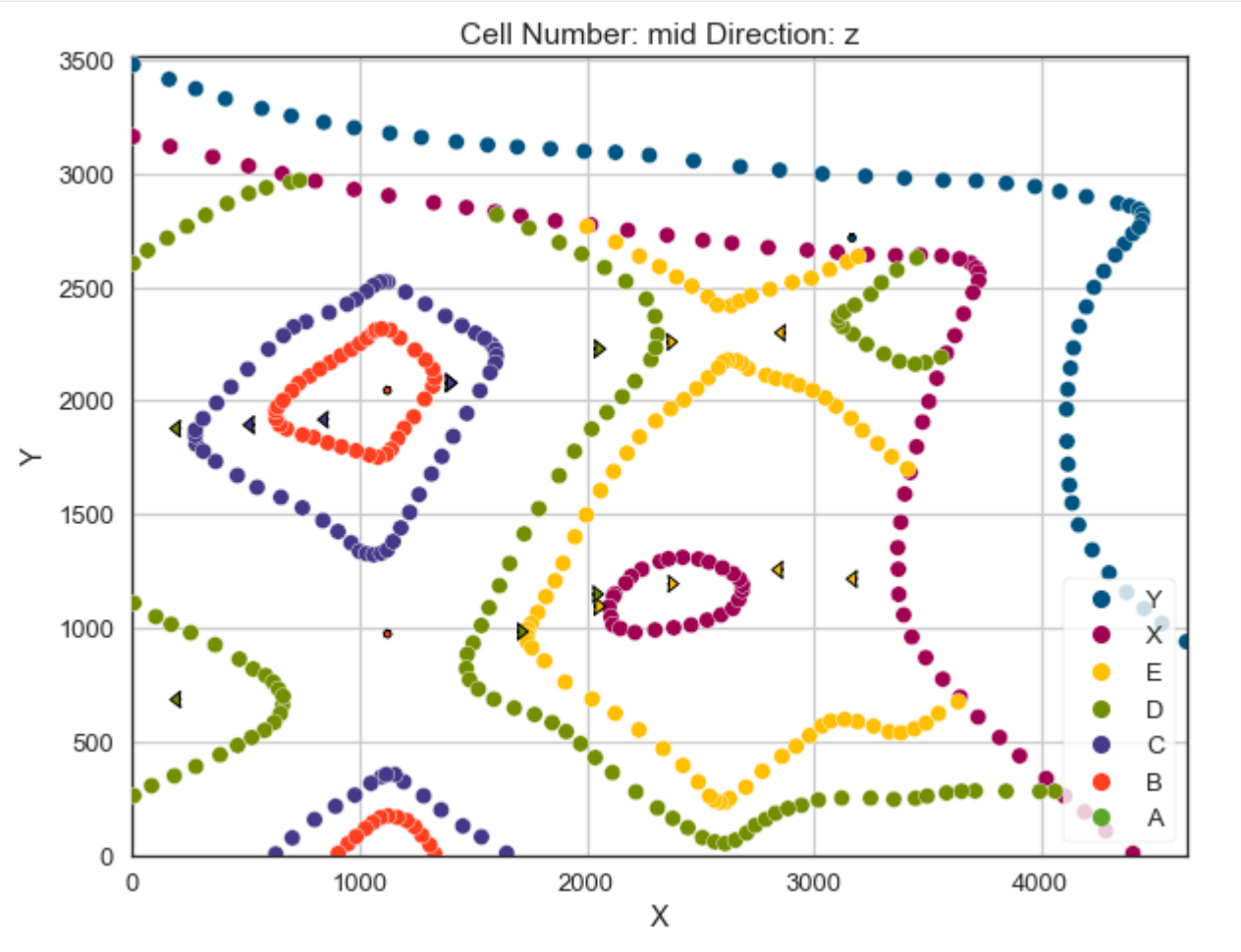
[38]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster6.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

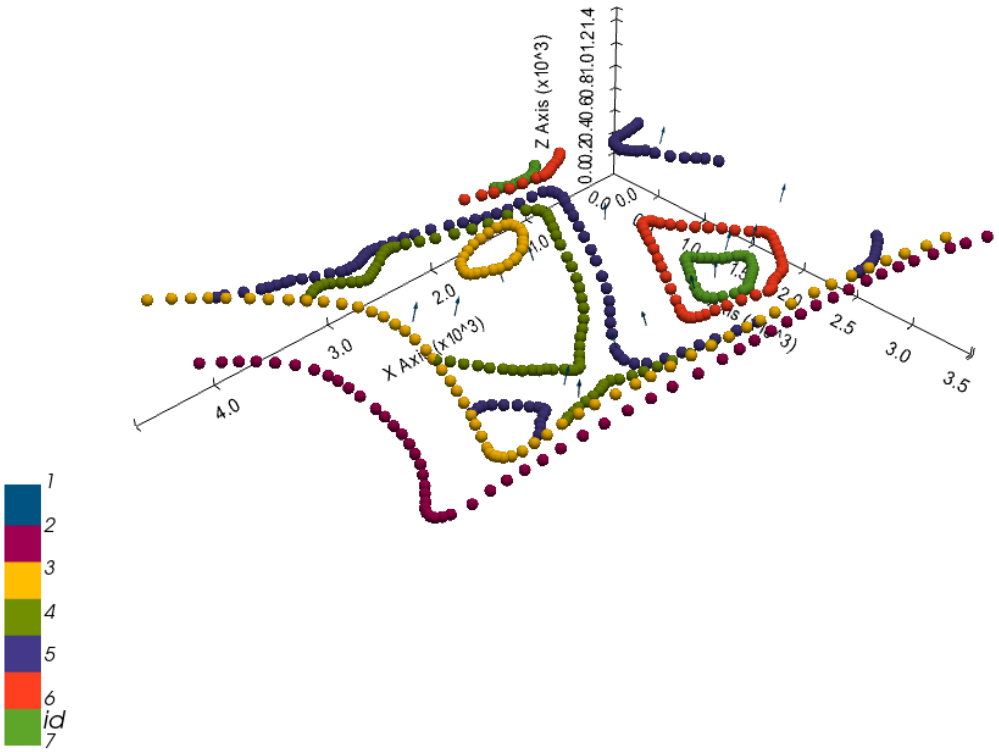
[38]:
```
Grid Object. Values:
array([[  23.21      ,   17.595     ,    7.5       ],
       [  23.21      ,   17.595     ,   22.5       ],
       [  23.21      ,   17.595     ,   37.5       ],
       ...,
       [4636.99784483, 3494.00710227, 1253.95422363],
       [4636.99784483, 3504.00426136, 1257.55334473],
       [4636.99784483, 3514.00142045, 1261.13220215]])
```

Plotting Input Data

[39]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[40]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[40]:
```
<gempy.plot.vista.GemPyToVista at 0x15aa7dcd2e0>
```

Setting the Interpolator

[41]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              6015.11
$C_o$            861464.88
drift equations     [3, 3]
```

[41]:
```
<gempy.core.interpolator.InterpolatorModel at 0x15aa61133d0>
```

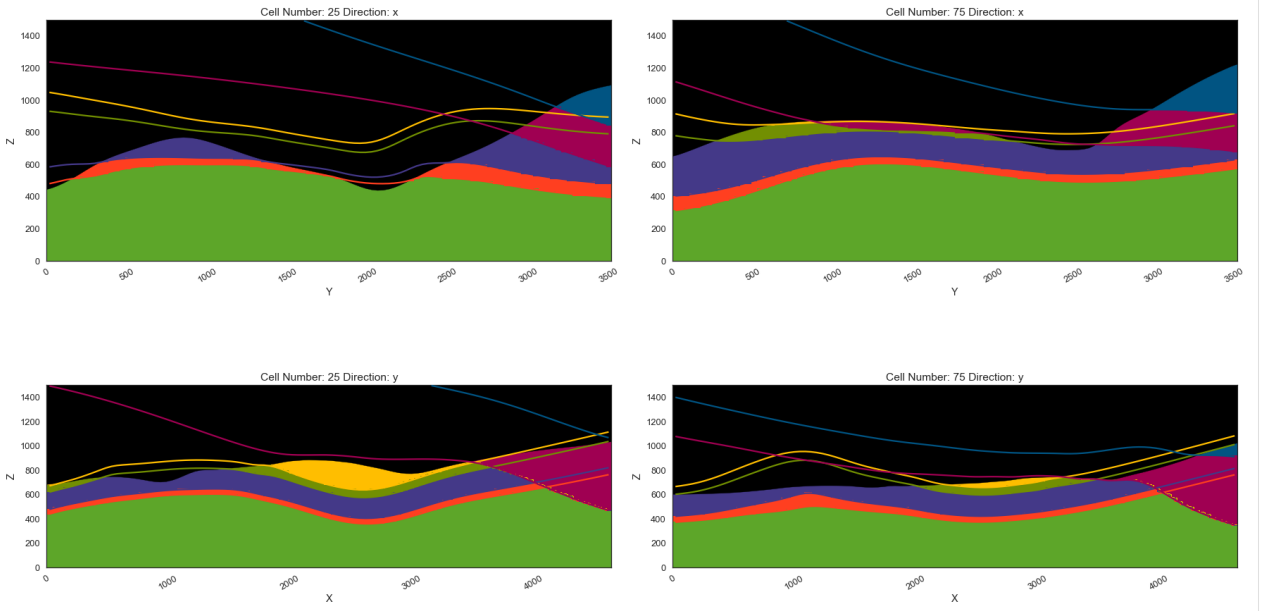
Computing Model

[42]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

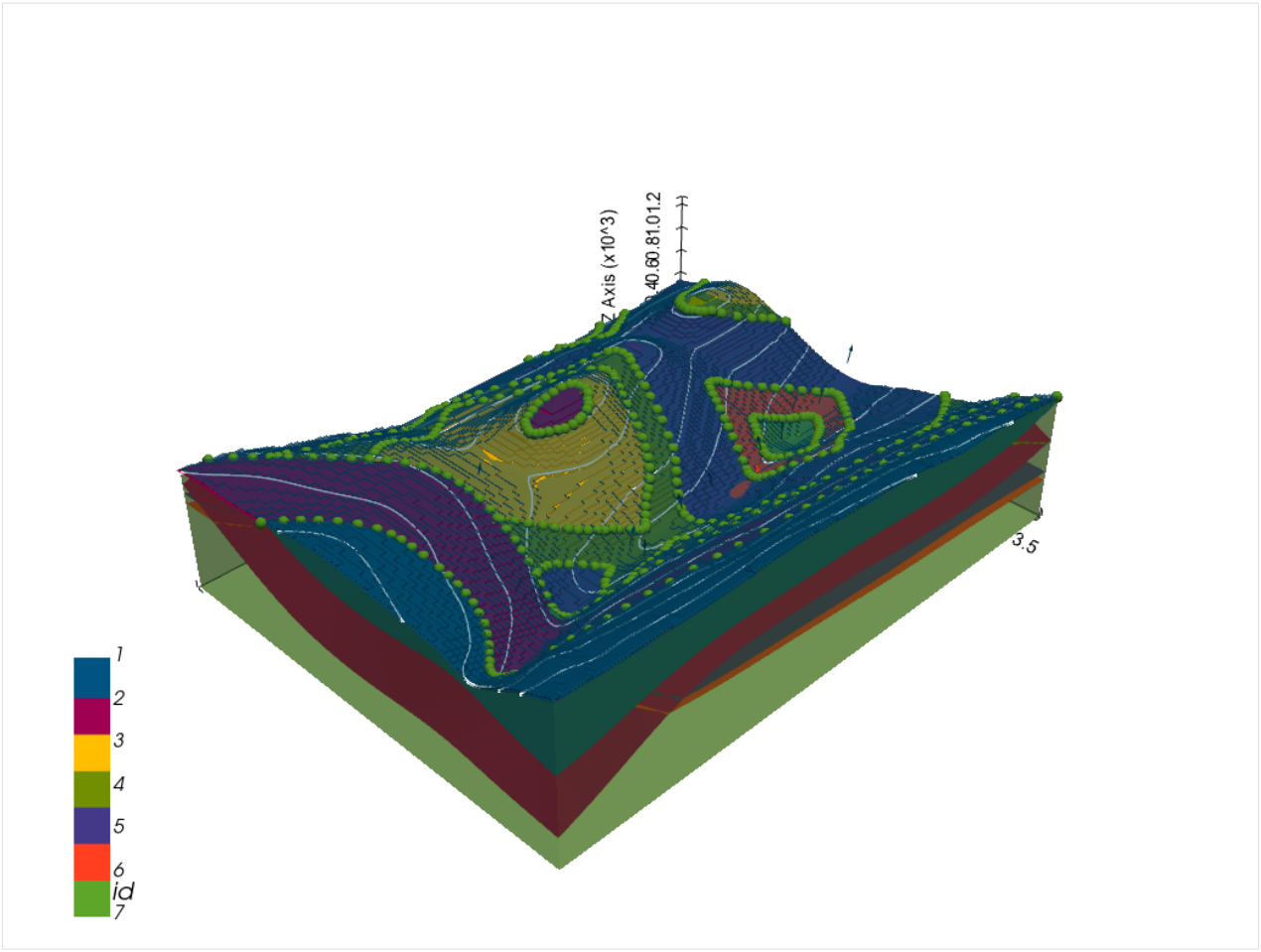
[43]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[43]:
```
<gempy.plot.visualization_2d.Plot2D at 0x15aae678910>
```



Plotting 3D Model

[44]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



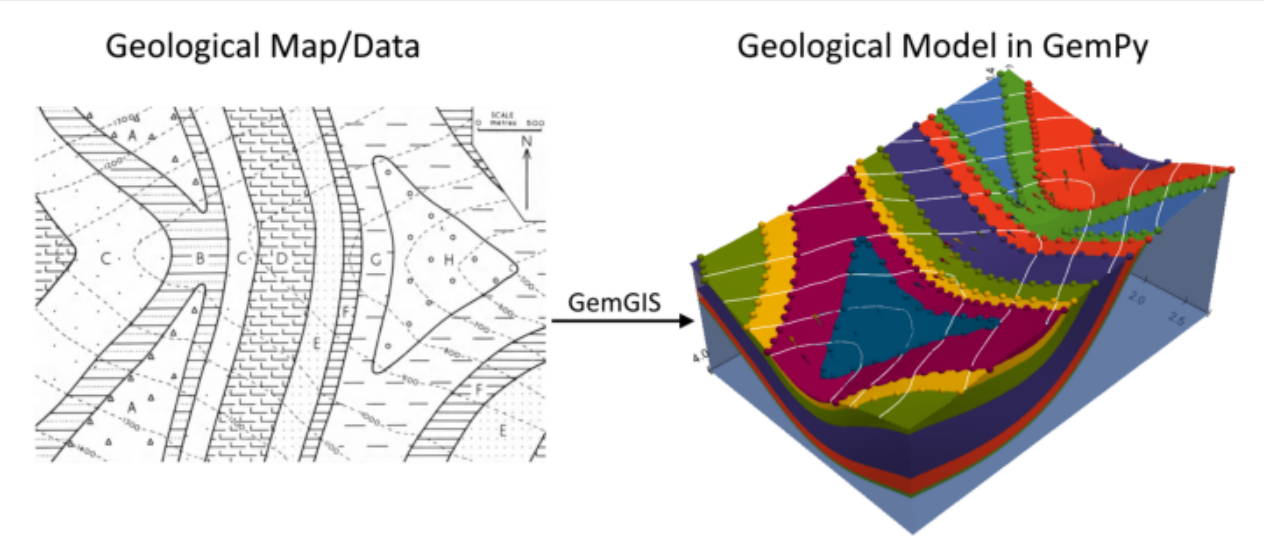
[ ]:

7.7 Example 7 - Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 4016 m wide (W-E extent) and 2790 m high (N-S extent). The vertical model extent varies from -250 m to 1500 m. The model represents folded layers (blue to light green) above a basement unit (light blue). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example07.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.7.1 Licensing

7.7.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.7.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

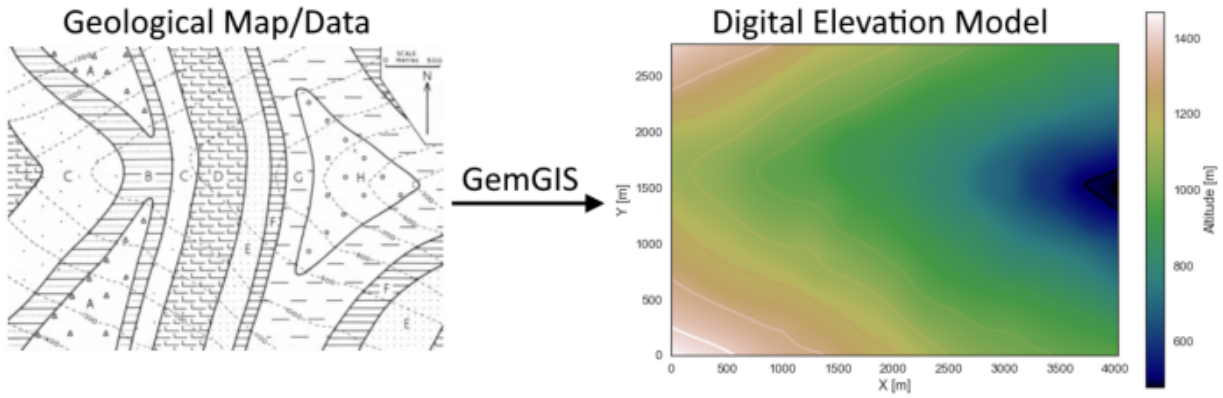
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example07/'
gg.download_gemgis_data.download_tutorial_data(filename="example07_folded_layers.zip",␣
↪dirpath=file_path)
```

7.7.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example07.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo7.shp')
topo.head()
```

[6]:
```
     id     Z                                           geometry
0  None  1400  LINESTRING (3.284 251.204, 70.568 216.700, 129...
1  None  1300  LINESTRING (1.559 681.649, 61.079 639.381, 127...
2  None  1200  LINESTRING (4.146 1199.219, 94.721 1109.507, 1...
3  None  1300  LINESTRING (4.578 2374.964, 84.370 2414.213, 1...
4  None  1200  LINESTRING (4.146 2108.847, 71.430 2114.885, 1...
```

Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

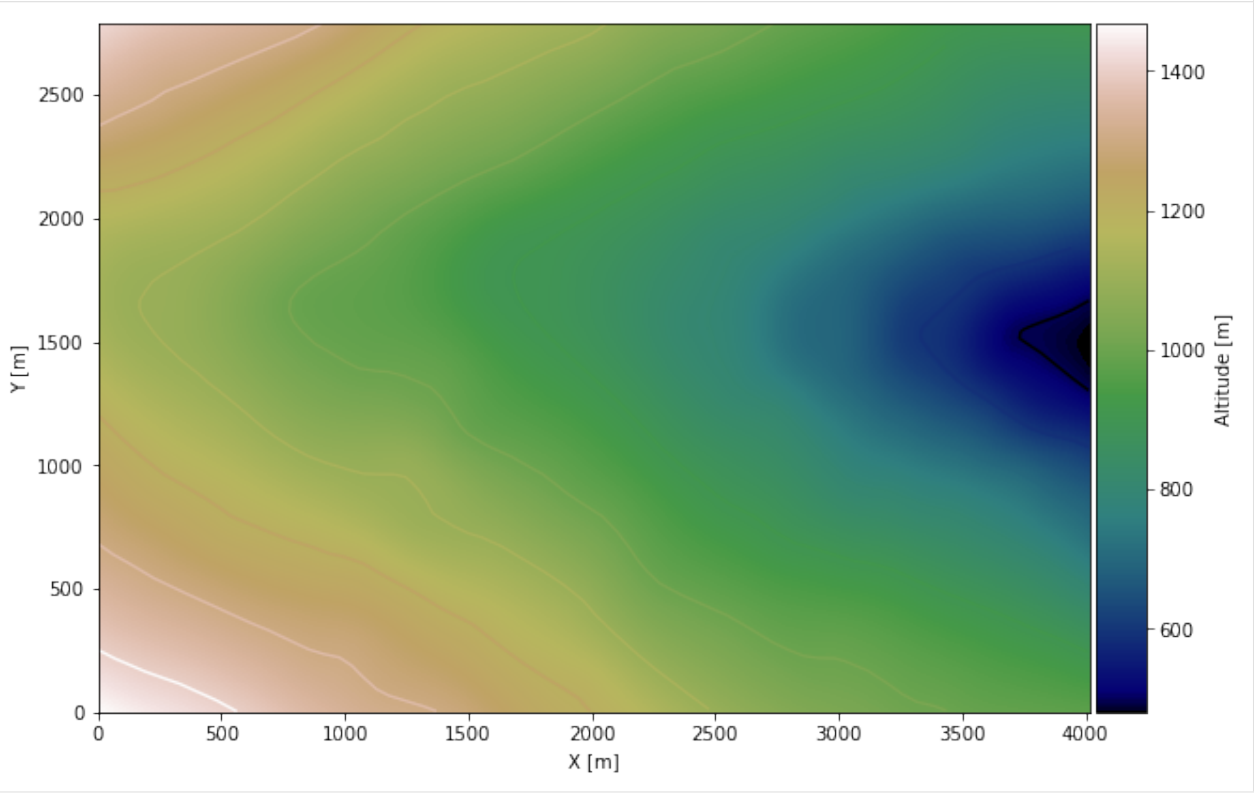
[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable
fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 4016, 0, 2790], cmap='gist_earth
↪')
```

(continues on next page)

(continued from previous page)

```
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4016)
ax.set_ylim(0, 2790)
```

[8]: (0.0, 2790.0)



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster7.tif', extent = [0, 4016, 0, 2790], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

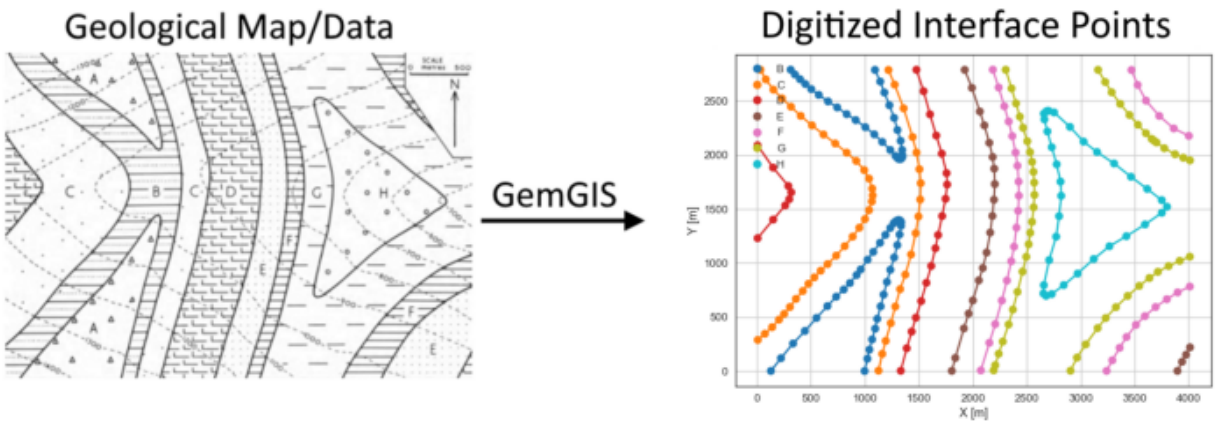
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster7.tif')
```

7.7.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example07.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces7.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	H	LINESTRING (2701.546 2406.881, 2753.302 2392.2...
1	None	G	LINESTRING (3160.457 2785.569, 3226.878 2673.4...
2	None	G	LINESTRING (2905.123 4.496, 2965.506 125.262, ...
3	None	F	LINESTRING (3239.818 2.771, 3292.437 100.246, ...
4	None	E	LINESTRING (3896.268 2.771, 3938.536 86.445, 3...

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

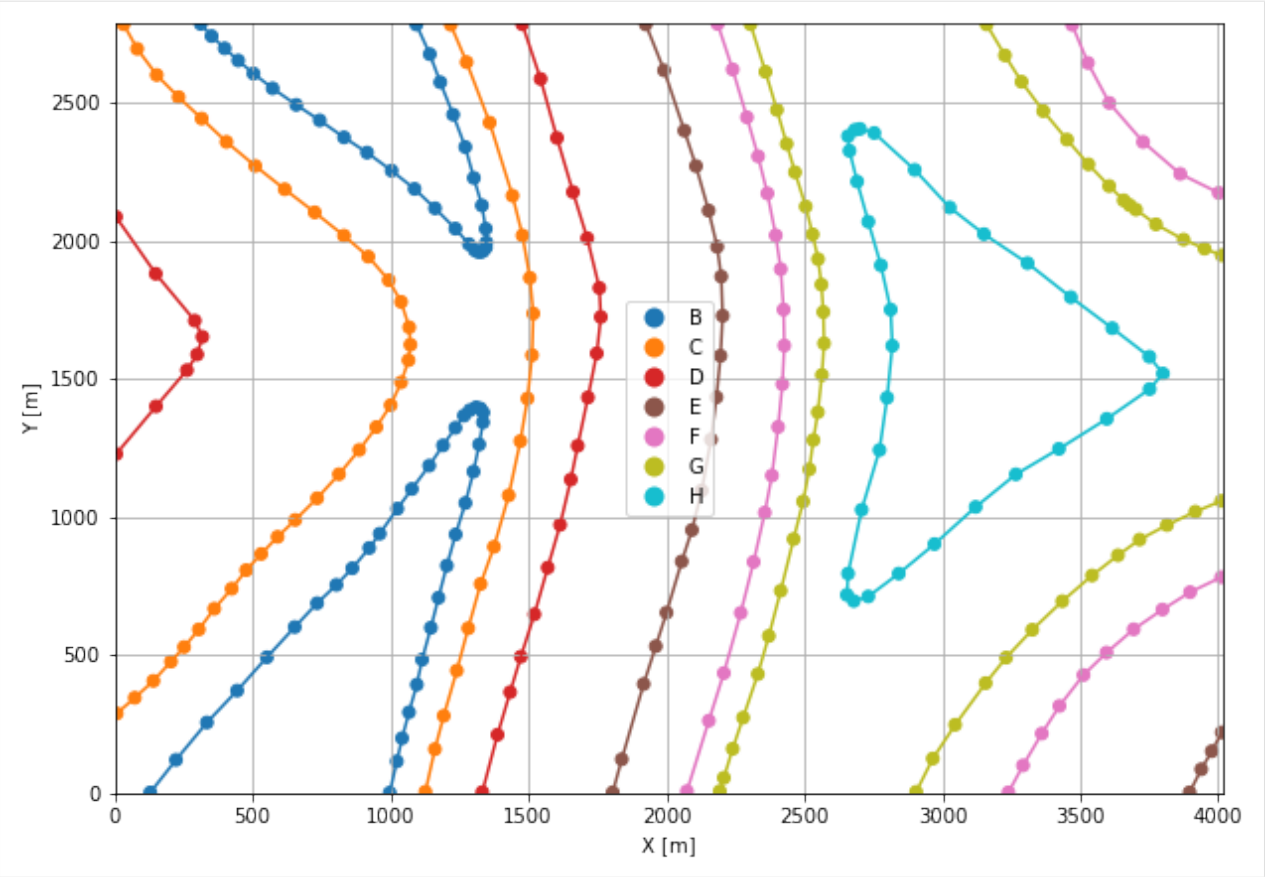
[12]:

	formation	geometry	X	Y	Z
0	H	POINT (2701.546 2406.881)	2701.55	2406.88	911.61
17	H	POINT (2731.737 712.704)	2731.74	712.70	904.72
1	H	POINT (2753.302 2392.216)	2753.30	2392.22	904.77
31	H	POINT (2683.431 2403.430)	2683.43	2403.43	913.30
29	H	POINT (2663.590 2327.520)	2663.59	2327.52	895.20

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4016)
ax.set_ylim(0, 2790)
```

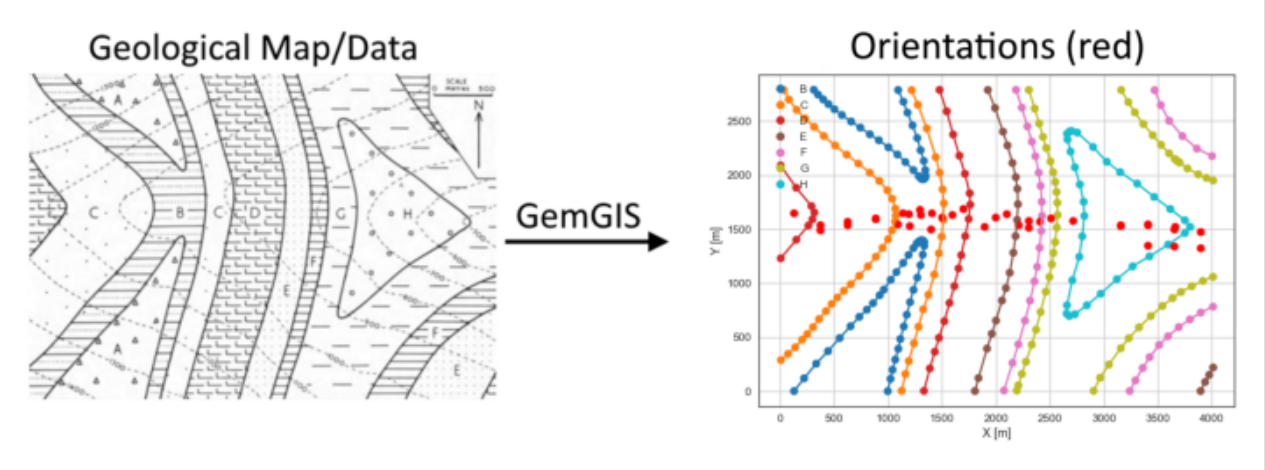
[13]:
```
(0.0, 2790.0)
```



7.7.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example07.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes7.shp')
strikes.head()
```

[15]:
```
   id formation    Z                                          geometry
0   1         H  500  LINESTRING (3775.502 1556.773, 3777.227 1482.157)
1   2         H  600  LINESTRING (3536.557 1741.804, 3533.970 1312.221)
2   3         H  700  LINESTRING (3281.223 1939.343, 3278.635 1159.538)
3   4         H  800   LINESTRING (3034.515 2115.317, 3025.026 947.335)
4   2        H1  900   LINESTRING (2660.140 2341.322, 2657.552 791.202)
```

Calculate Orientations for each formation

[16]:
```
orientations_b = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B'].sort_values(by='Z', ascending=True).reset_index())
orientations_b
```

[16]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 48.08    90.40 1050.00  POINT (1289.120 1684.656)      1.00         B
1 44.77    90.25 1150.00  POINT (1191.860 1645.299)      1.00         B
2 45.45    90.15 1250.00  POINT (1090.395 1549.117)      1.00         B

        X       Y
0 1289.12 1684.66
1 1191.86 1645.30
2 1090.40 1549.12
```

[17]:
```
orientations_b1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B1'].sort_values(by='Z', ascending=True).reset_index())
orientations_b1
```

[17]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 21.67    90.24 1050.00  POINT (1131.154 1653.170)      1.00        B1
1 22.38    90.19 1150.00   POINT (879.917 1593.218)      1.00        B1
2 21.07    90.17 1250.00   POINT (625.877 1541.677)      1.00        B1
3 22.22    90.21 1350.00   POINT (372.268 1490.136)      1.00        B1

        X       Y
```

(continues on next page)

(continued from previous page)

```
0 1131.15 1653.17
1  879.92 1593.22
2  625.88 1541.68
3  372.27 1490.14
```

[18]:
```
orientations_c = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C'].sort_values(by='Z', ascending=True).reset_index())
orientations_c
```

[18]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 44.89    90.05 1050.00  POINT (1400.182 1647.671)      1.00         C
1 46.27    89.68 1150.00  POINT (1298.016 1630.742)      1.00         C
2 45.82    89.73 1250.00  POINT (1194.502 1529.169)      1.00         C

        X       Y
0 1400.18 1647.67
1 1298.02 1630.74
2 1194.50 1529.17
```

[19]:
```
orientations_c1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C1'].sort_values(by='Z', ascending=True).reset_index())
orientations_c1
```

[19]:
```
    dip  azimuth       Z                   geometry  polarity formation      X  \
0 22.00    90.03 1050.00  POINT (880.887 1605.834)      1.00        C1 880.89
1 21.02    90.00 1150.00  POINT (625.984 1571.006)      1.00        C1 625.98
2 22.47    90.04 1250.00  POINT (373.777 1535.531)      1.00        C1 373.78

        Y
0 1605.83
1 1571.01
2 1535.53
```

[20]:
```
orientations_d = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D'].sort_values(by='Z', ascending=True).reset_index())
orientations_d
```

[20]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 43.21    90.59  950.00  POINT (1694.388 1687.459)      1.00         D
1 47.09    90.51 1050.00  POINT (1593.246 1635.810)      1.00         D
2 46.90    90.42 1150.00  POINT (1498.520 1611.873)      1.00         D
3 43.42    90.28 1250.00  POINT (1395.761 1496.605)      1.00         D

        X       Y
0 1694.39 1687.46
1 1593.25 1635.81
2 1498.52 1611.87
3 1395.76 1496.61
```

[21]:
```
orientations_d1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D1'].sort_values(by='Z', ascending=True).reset_index())
orientations_d1
```

[21]:
```
    dip  azimuth       Z                  geometry  polarity formation      X  \
0 22.43    90.07 1150.00  POINT (124.913 1649.935)      1.00        D1 124.91

        Y
0 1649.94
```

[22]:
```
orientations_e = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'E'].sort_values(by='Z', ascending=True).reset_index())
orientations_e
```

[22]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 43.76    90.34  950.00  POINT (2099.332 1639.422)      1.00         E
1 45.01    90.27 1050.00  POINT (1996.034 1608.314)      1.00         E
2 43.87    90.13 1150.00  POINT (1891.549 1524.425)      1.00         E

        X       Y
0 2099.33 1639.42
1 1996.03 1608.31
2 1891.55 1524.42
```

[23]:
```
orientations_f = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'F'].sort_values(by='Z', ascending=True).reset_index())
orientations_f
```

[23]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.87    90.06 750.00  POINT (3892.926 1329.042)      1.00         F
1 22.62    90.02 850.00  POINT (3652.148 1342.844)      1.00         F
2 21.94    90.16 950.00  POINT (3404.146 1346.510)      1.00         F

        X       Y
0 3892.93 1329.04
1 3652.15 1342.84
2 3404.15 1346.51
```

[24]:
```
orientations_f1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'F1'].sort_values(by='Z', ascending=True).reset_index())
orientations_f1
```

[24]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 45.08    90.35  950.00  POINT (2303.988 1581.088)      1.00        F1
1 47.80    90.16 1050.00  POINT (2203.061 1538.981)      1.00        F1

        X       Y
0 2303.99 1581.09
1 2203.06 1538.98
```

[25]:
```
orientations_g = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'G'].sort_values(by='Z', ascending=True).reset_index())
orientations_g
```

[25]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 23.14    89.99 650.00  POINT (3893.249 1475.040)      1.00         G
1 22.18    90.01 750.00  POINT (3652.148 1500.919)      1.00         G
2 22.32    89.97 850.00  POINT (3405.979 1548.255)      1.00         G
```

(continues on next page)

(continued from previous page)

```
3 22.15    90.18 950.00  POINT (3156.252 1530.894)      1.00         G

        X       Y
0 3893.25 1475.04
1 3652.15 1500.92
2 3405.98 1548.25
3 3156.25 1530.89
```

[26]:
```
orientations_g1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'G1'].sort_values(by='Z', ascending=True).reset_index())
orientations_g1
```

[26]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 47.83    90.14  850.00  POINT (2507.888 1600.335)      1.00         G1
1 44.48    90.31  950.00  POINT (2406.315 1571.006)      1.00         G1
2 46.29    90.39 1050.00  POINT (2303.880 1516.014)      1.00         G1

        X       Y
0 2507.89 1600.34
1 2406.32 1571.01
2 2303.88 1516.01
```

[27]:
```
orientations_h = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'H'].sort_values(by='Z', ascending=True).reset_index())
orientations_h
```

[27]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.62    90.30 550.00  POINT (3655.814 1523.239)      1.00         H
1 21.43    90.23 650.00  POINT (3407.596 1538.227)      1.00         H
2 21.95    90.38 750.00  POINT (3154.850 1540.383)      1.00         H

        X       Y
0 3655.81 1523.24
1 3407.60 1538.23
2 3154.85 1540.38
```

[28]:
```
orientations_h1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'H1'].sort_values(by='Z', ascending=True).reset_index())
orientations_h1
```

[28]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 42.80    90.12 850.00  POINT (2713.191 1576.829)      1.00        H1

        X       Y
0 2713.19 1576.83
```

Merging Orientations

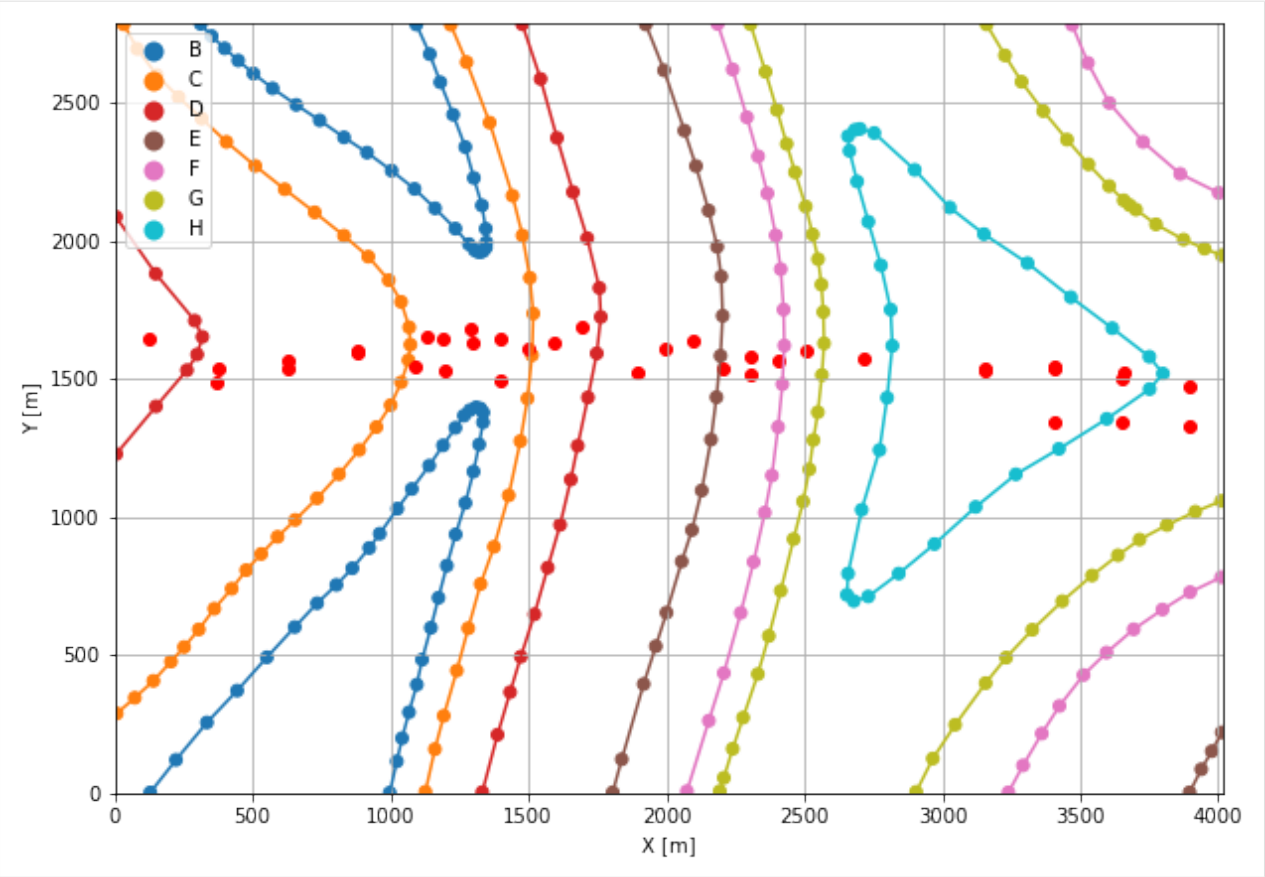
[29]:
```
import pandas as pd
orientations = pd.concat([orientations_b, orientations_b1, orientations_c, orientations_
↪c1, orientations_d, orientations_d1, orientations_e, orientations_f, orientations_f1,␣
↪orientations_g, orientations_g1, orientations_h, orientations_h1]).reset_index()
orientations['formation'] = ['B', 'B', 'B', 'B', 'B', 'B', 'B', 'C', 'C', 'C', 'C', 'C',
↪'C', 'D', 'D', 'D', 'D', 'D',
 'E', 'E', 'E', 'F', 'F', 'F', 'F', 'F', 'G', 'G', 'G', 'G', 'G', 'G', 'G', 'H', 'H', 'H
↪', 'H']
orientations.head()
```

[29]:
```
   index   dip  azimuth       Z                   geometry  polarity  \
0      0 48.08    90.40 1050.00  POINT (1289.120 1684.656)      1.00
1      1 44.77    90.25 1150.00  POINT (1191.860 1645.299)      1.00
2      2 45.45    90.15 1250.00  POINT (1090.395 1549.117)      1.00
3      0 21.67    90.24 1050.00  POINT (1131.154 1653.170)      1.00
4      1 22.38    90.19 1150.00   POINT (879.917 1593.218)      1.00

  formation       X       Y
0         B 1289.12 1684.66
1         B 1191.86 1645.30
2         B 1090.40 1549.12
3         B 1131.15 1653.17
4         B  879.92 1593.22
```

Plotting the Orientations

[30]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4016)
ax.set_ylim(0, 2790)
```

[30]:
```
(0.0, 2790.0)
```



7.7.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[31]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[32]:
```
geo_model = gp.create_model('Model7')
geo_model
```

[32]:
```
Model7  2022-04-05 09:52
```

Initiate Data

[33]:
```
gp.init_data(geo_model, [0, 4016, 0, 2790, -250, 1500], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z']!=0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[33]:
```
Model7  2022-04-05 09:52
```

Model Surfaces

[34]:
```
geo_model.surfaces
```

[34]:

	surface	series	order_surfaces	color	id
0	H	Default series	1	#015482	1
1	G	Default series	2	#9f0052	2
2	F	Default series	3	#ffbe00	3
3	E	Default series	4	#728f02	4
4	D	Default series	5	#443988	5
5	C	Default series	6	#ff3f20	6
6	B	Default series	7	#5DA629	7

Mapping the Stack to Surfaces

[35]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('H', 'G', 'F', 'E', 'D', 'C', 'B')
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('A')
```

[35]:

	surface	series	order_surfaces	color	id
0	H	Strata1	1	#015482	1
1	G	Strata1	2	#9f0052	2
2	F	Strata1	3	#ffbe00	3
3	E	Strata1	4	#728f02	4
4	D	Strata1	5	#443988	5
5	C	Strata1	6	#ff3f20	6
6	B	Strata1	7	#5DA629	7
7	A	Strata1	8	#4878d0	8

Showing the Number of Data Points

[36]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[36]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	H	Strata1	1	#015482	1	32	4
1	G	Strata1	2	#9f0052	2	51	7
2	F	Strata1	3	#ffbe00	3	34	5
3	E	Strata1	4	#728f02	4	23	3
4	D	Strata1	5	#443988	5	26	5
5	C	Strata1	6	#ff3f20	6	52	6
6	B	Strata1	7	#5DA629	7	64	7
7	A	Strata1	8	#4878d0	8	0	0

Loading Digital Elevation Model

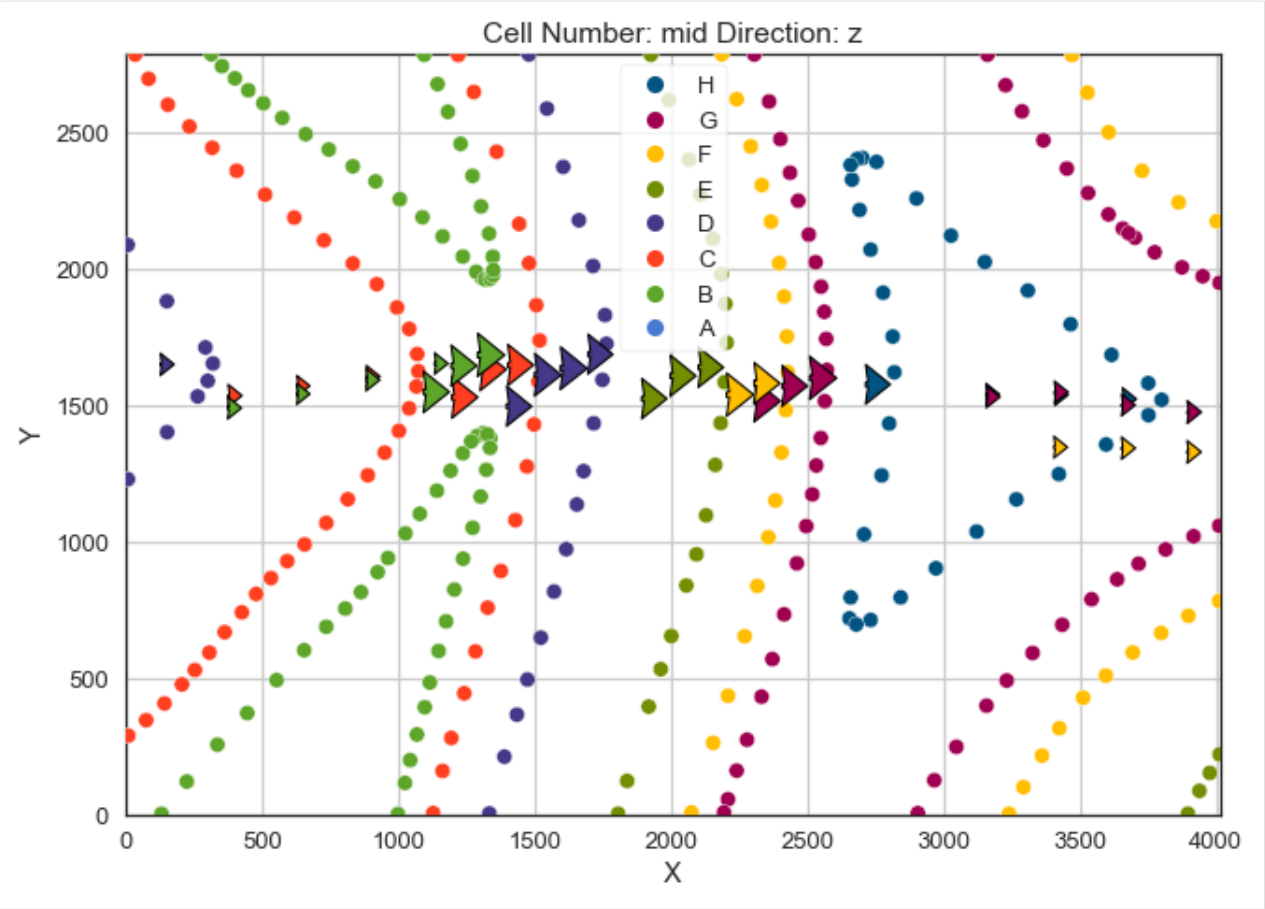
[37]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster7.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

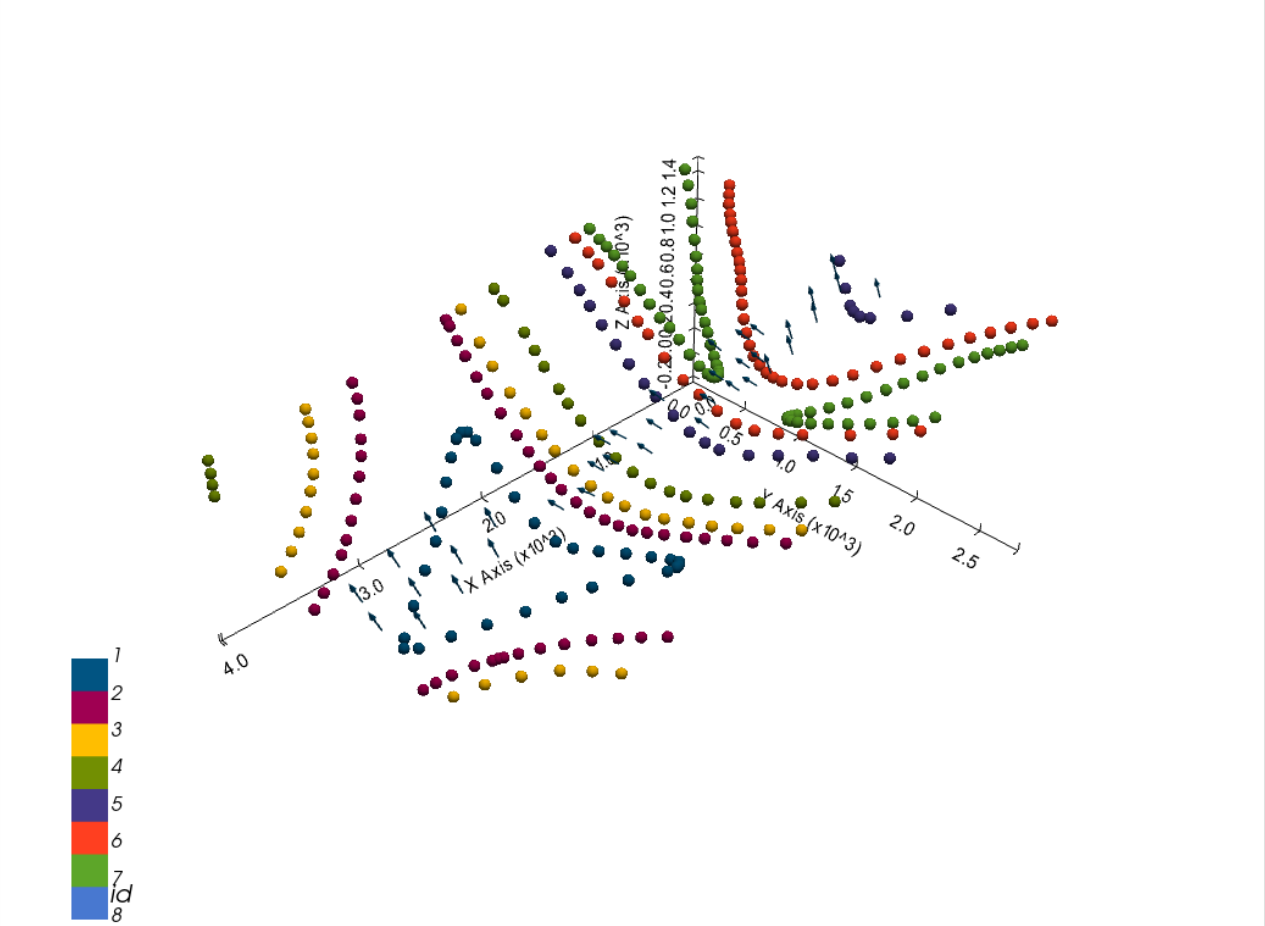
[37]:
```
Grid Object. Values:
array([[  20.08      ,   13.95      , -241.25      ],
       [  20.08      ,   13.95      , -223.75      ],
       [  20.08      ,   13.95      , -206.25      ],
       ...,
       [4011.00497512, 2765.        ,  898.98010254],
       [4011.00497512, 2775.        ,  902.10076904],
       [4011.00497512, 2785.        ,  905.22875977]])
```

Plotting Input Data

[38]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[39]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[39]:
```
<gempy.plot.vista.GemPyToVista at 0x1fa0b79d5b0>
```

Setting the Interpolator

[40]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              5193.73
$C_o$            642258.48
drift equations       [3]
```

[40]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1fa06690850>
```

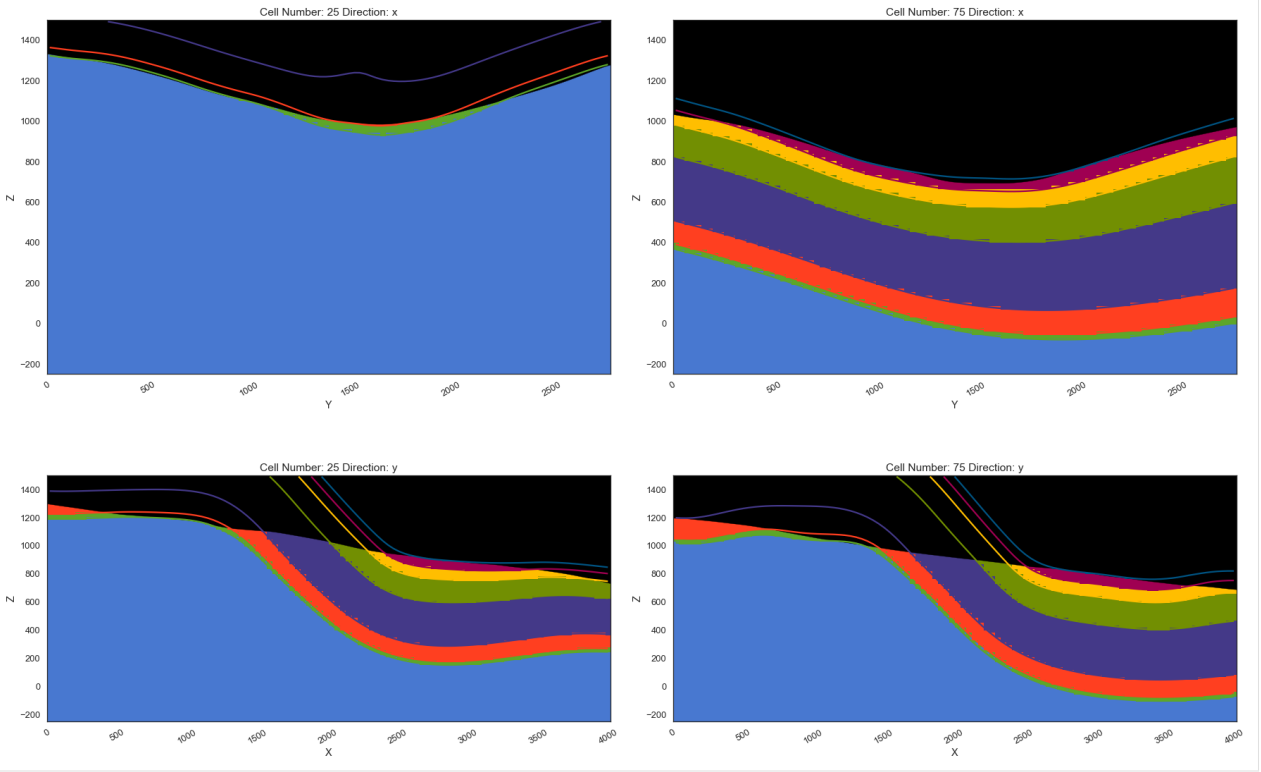
Computing Model

[41]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

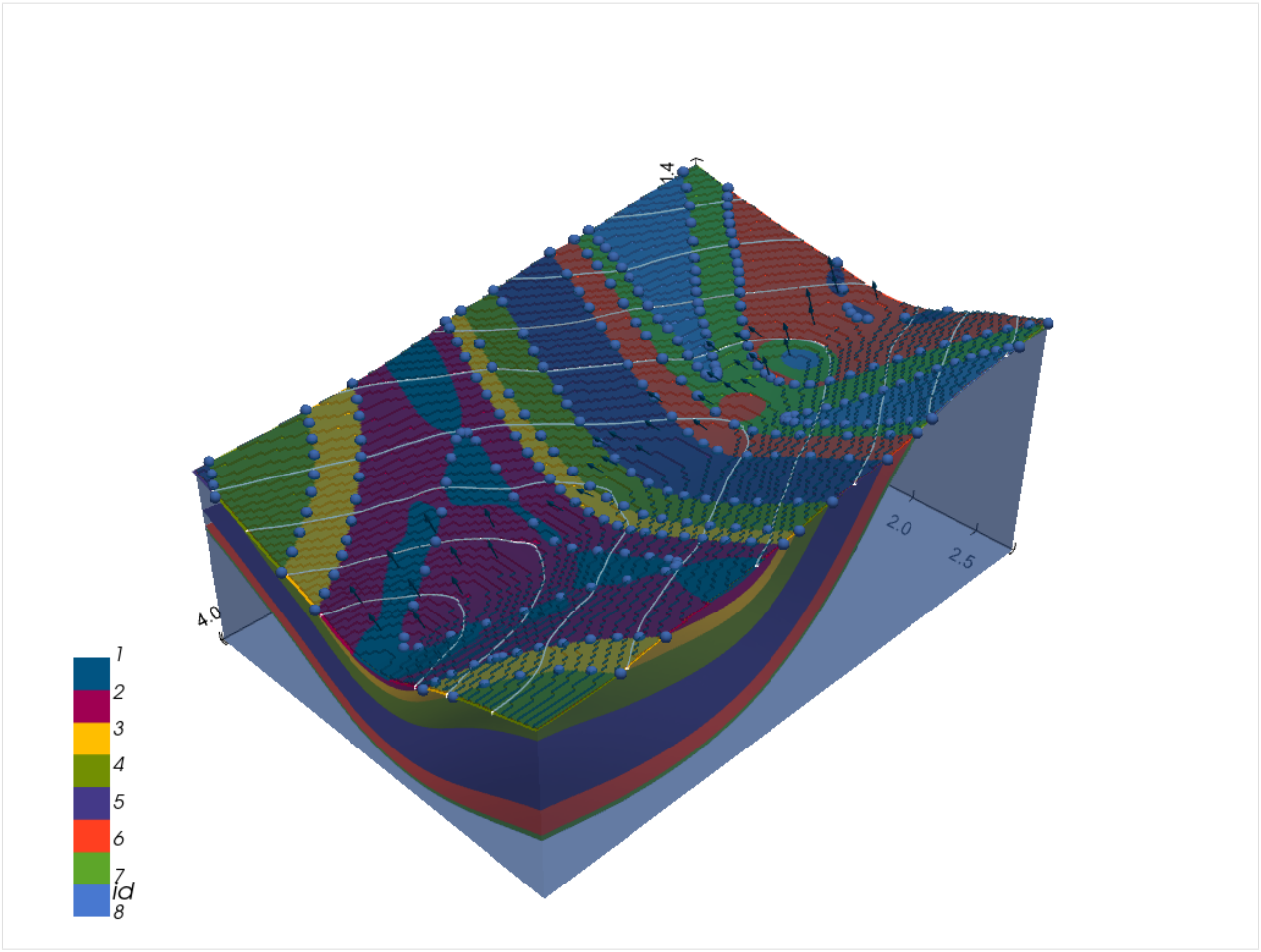
[42]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[42]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1fa09c4a8e0>
```



Plotting 3D Model

[43]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



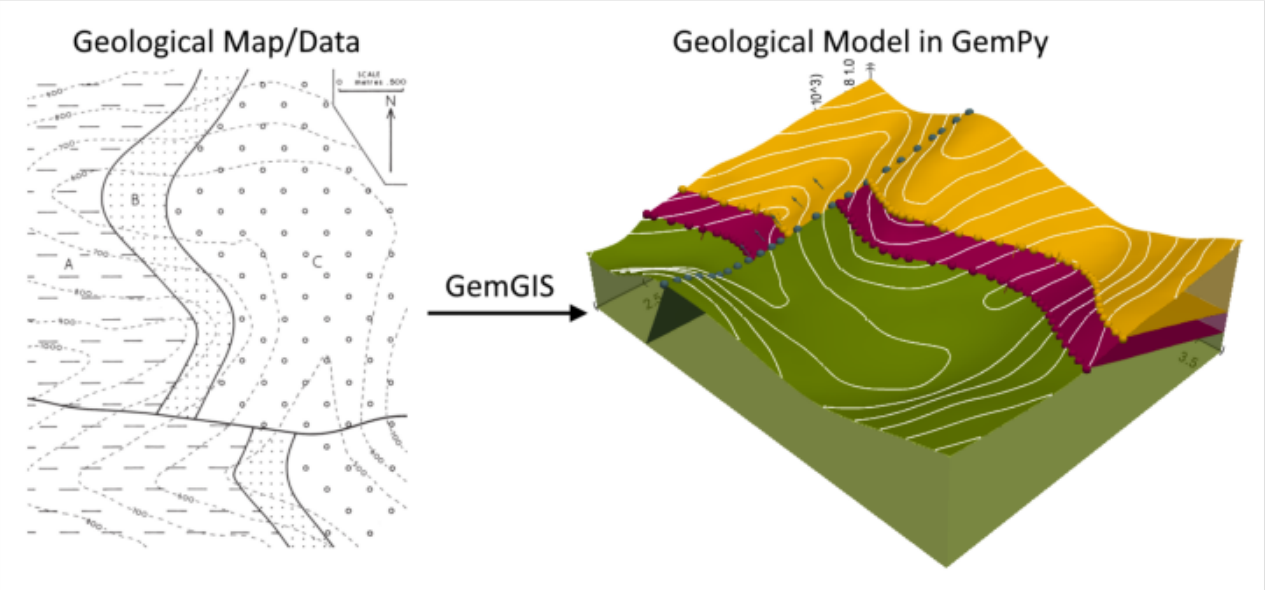
[ ]:

7.8 Example 8 - Faulted Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 2957 m wide (W-E extent) and 3715 m high (N-S extent). The vertical extent varies between 0 m and 1250 m. The model represents three faulted layers, red and yellow above a green basement. The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example08.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.8.1 Licensing

7.8.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.8.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

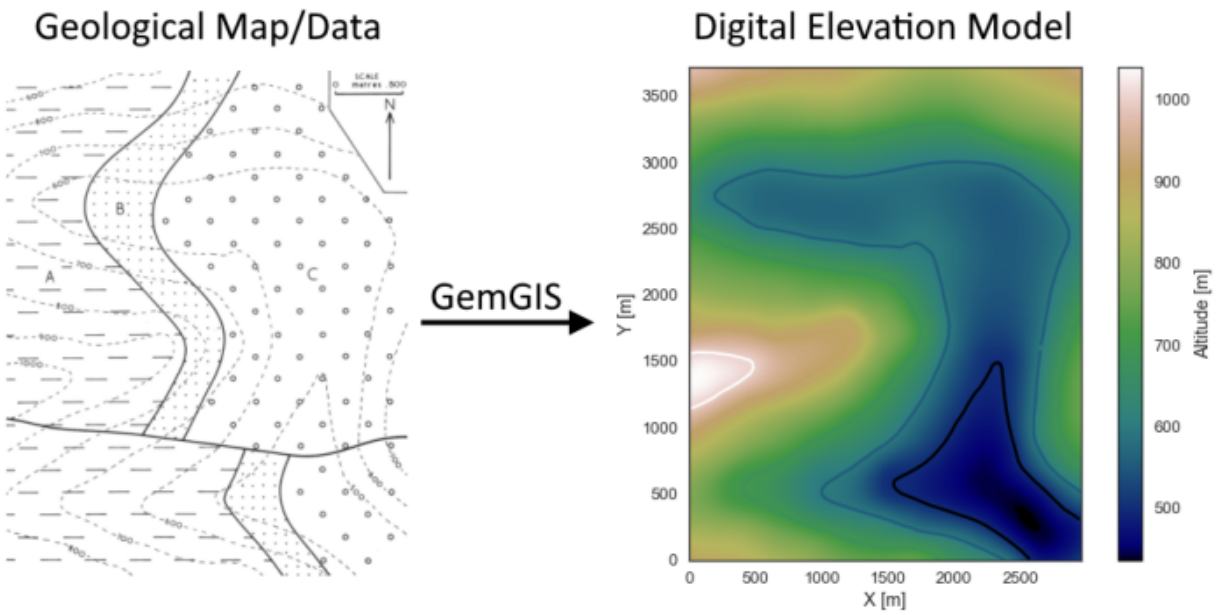
[4]:
```
file_path = 'data/example08/'
gg.download_gemgis_data.download_tutorial_data(filename="example08_faulted_layers.zip",␣
↪dirpath=file_path)
```

```
Downloading file 'example08_faulted_layers.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample08_faulted_layers.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\example\data\example08'.
```

7.8.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example08.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo8.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	800	LINESTRING (8.360 333.390, 94.504 303.527, 184...
1	None	700	LINESTRING (2954.497 1855.274, 2918.891 1755.3...
2	None	900	LINESTRING (3.765 3486.274, 124.368 3510.394, ...
3	None	800	LINESTRING (3.191 3280.676, 105.416 3314.559, ...
4	None	700	LINESTRING (3.191 2934.376, 53.729 2986.062, 1...

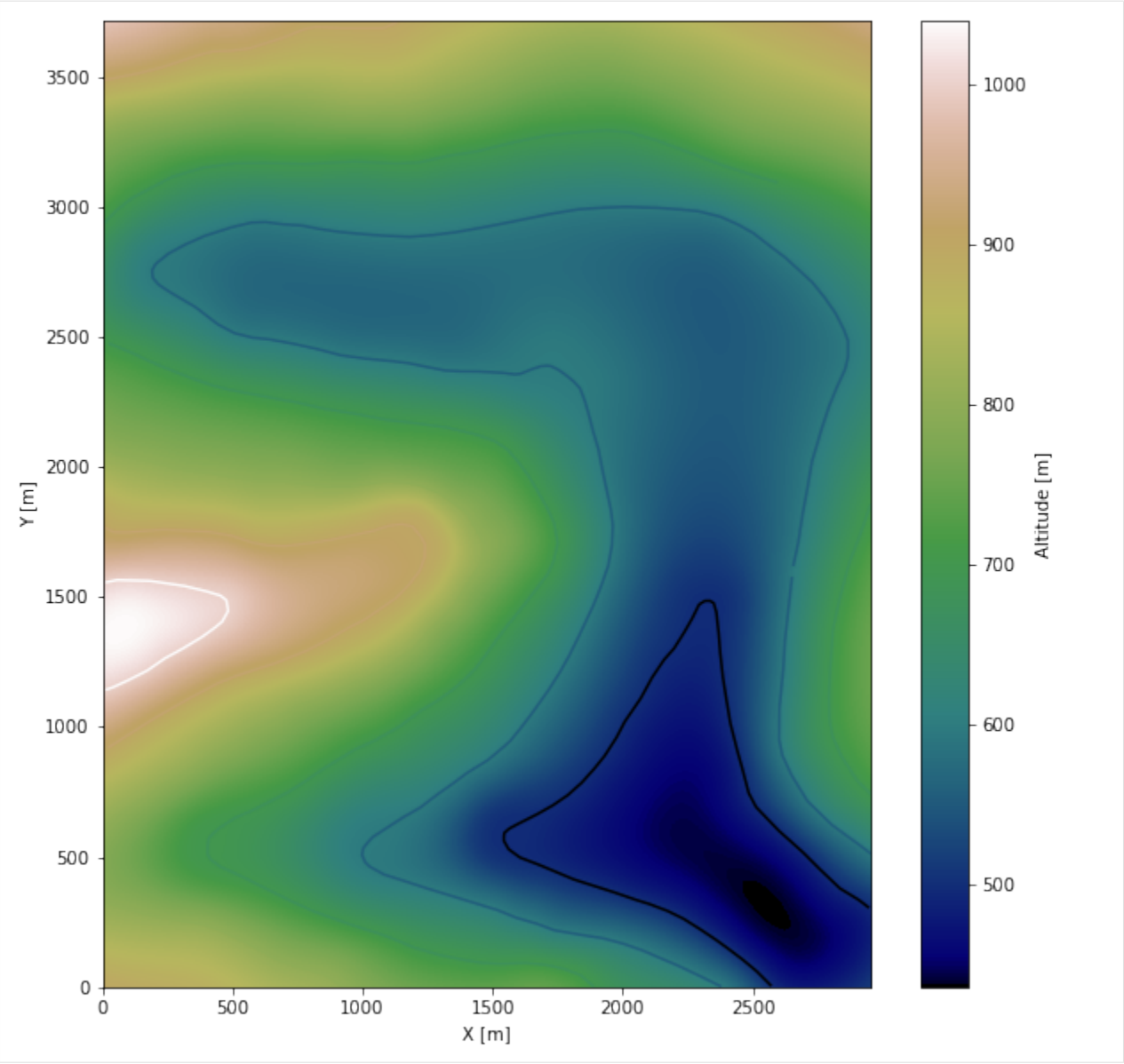
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 2957, 0, 3715], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2957)
ax.set_ylim(0, 3715)
```

[8]:
```
(0.0, 3715.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster8.tif', extent = [0, 2957, 0, 3715], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

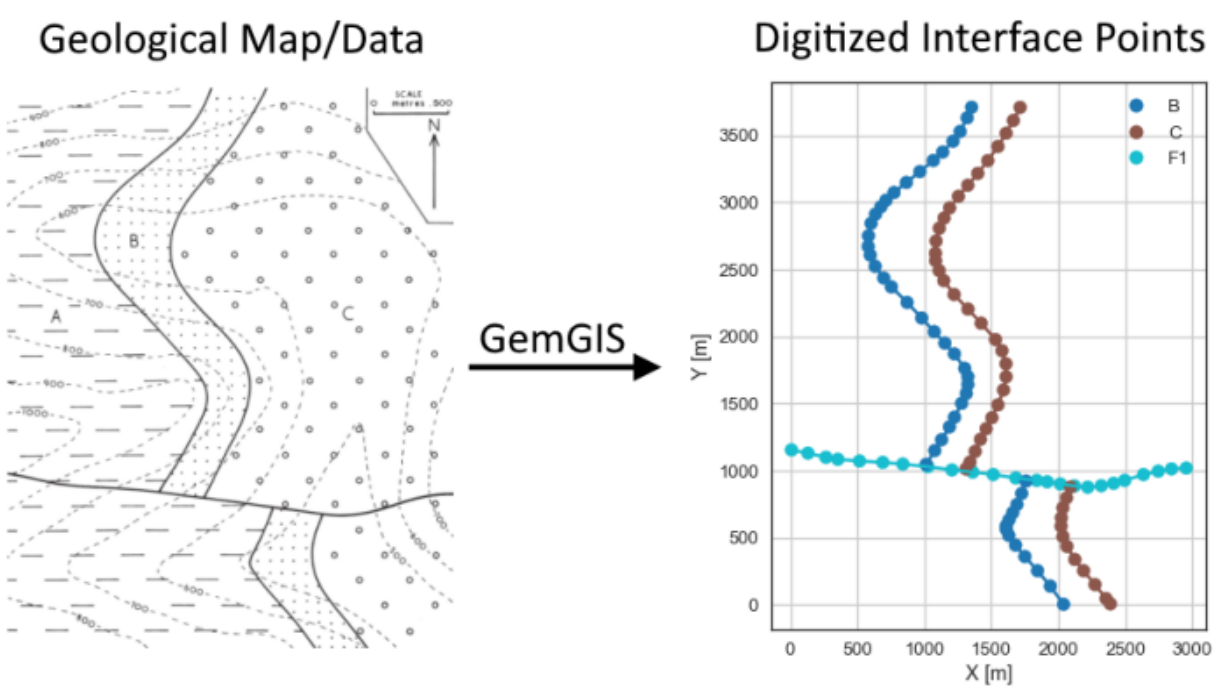
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster8.tif')
```

7.8.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example08.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces8.shp')
interfaces.head()
```

[11]:
```
     id formation                                           geometry
0  None         C  LINESTRING (1714.018 3708.526, 1663.480 3610.8...
1  None         B  LINESTRING (1352.212 3709.675, 1318.903 3630.4...
2  None         C  LINESTRING (2089.608 876.100, 2059.744 795.698...
```

(continues on next page)

(continued from previous page)

```
3  None         B  LINESTRING (1757.665 920.895, 1725.504 829.007...
4  None        F1  LINESTRING (8.360 1153.485, 131.259 1129.364, ...
```

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

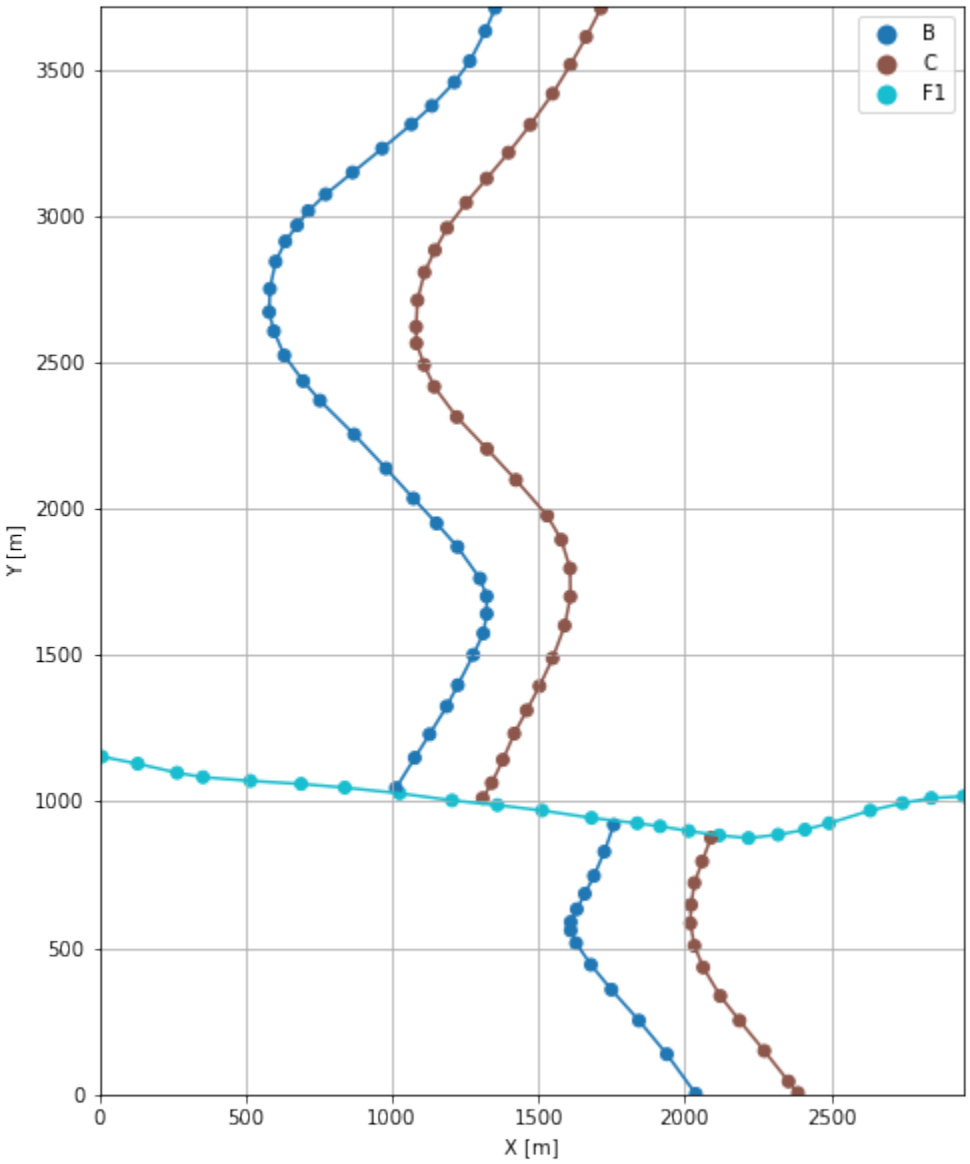
[12]:

	formation	geometry	X	Y	Z
113	F1	POINT (2953.348 1017.951)	2953.35	1017.95	745.77
101	F1	POINT (1681.858 944.441)	1681.86	944.44	567.06
90	F1	POINT (8.360 1153.485)	8.36	1153.48	1003.28
91	F1	POINT (131.259 1129.364)	131.26	1129.36	968.30
92	F1	POINT (265.644 1098.352)	265.64	1098.35	909.05

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2957)
ax.set_ylim(0, 3715)
```

[13]:
```
(0.0, 3715.0)
```



7.8.6 Orientations from Strike Lines

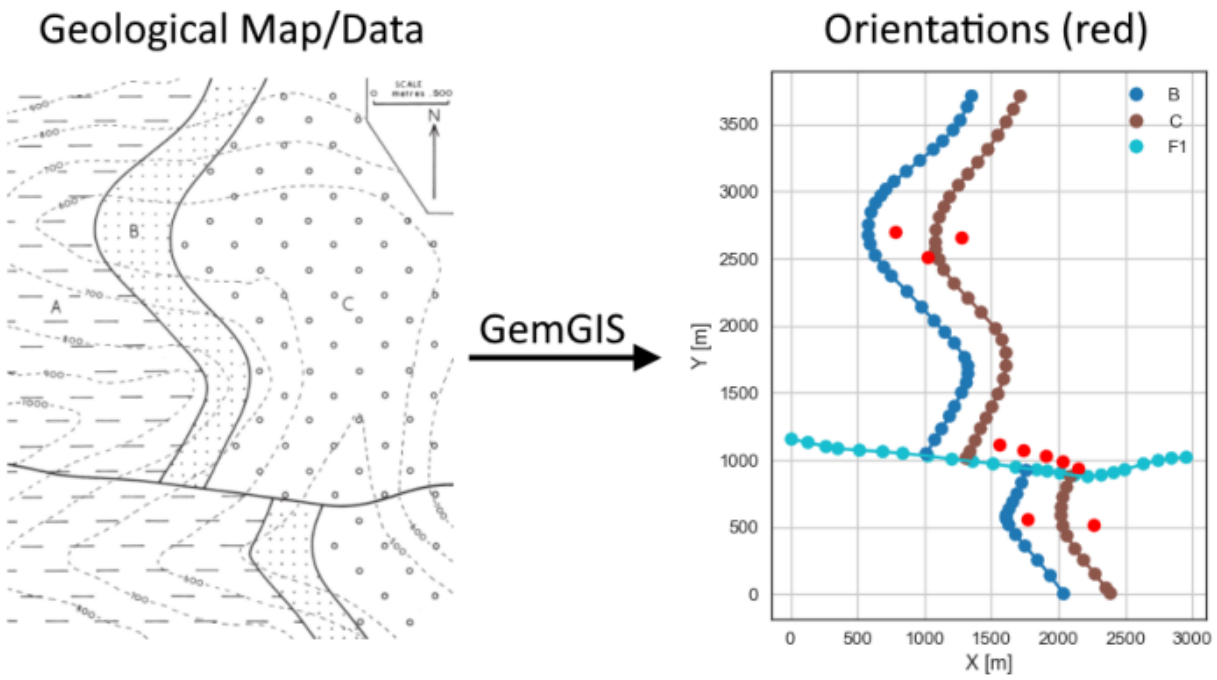
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/orientations_example08.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes8.shp')
strikes.head()
```

[15]:

	id	formation	Z	geometry
0	1	B1	500	LINESTRING (1645.103 652.699, 1651.420 479.261)
1	1	C1	500	LINESTRING (2151.632 301.230, 2144.740 877.823)
2	1	C	600	LINESTRING (1148.337 2886.709, 1154.654 2403.152)
3	2	C	700	LINESTRING (1390.115 3204.869, 1390.690 2132.659)
4	1	B	600	LINESTRING (654.443 2938.970, 658.463 2486.999)

Calculate Orientations for each formation

[16]:
```
orientations_f1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'F1'].sort_values(by='Z', ascending=True).reset_index())
orientations_f1
```

[16]:

	dip	azimuth	Z	geometry	polarity	formation	\
0	64.17	180.06	550.00	POINT (2142.443 936.831)	1.00	F1	
1	65.03	180.15	650.00	POINT (2034.906 985.072)	1.00	F1	
2	65.86	180.16	750.00	POINT (1907.556 1031.877)	1.00	F1	
3	70.31	180.05	850.00	POINT (1732.539 1074.232)	1.00	F1	
4	66.70	180.12	950.00	POINT (1559.676 1118.740)	1.00	F1	

(continues on next page)

(continued from previous page)

```
        X       Y
0 2142.44  936.83
1 2034.91  985.07
2 1907.56 1031.88
3 1732.54 1074.23
4 1559.68 1118.74
```

[17]:
```
orientations_c = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C'].sort_values(by='Z', ascending=True).reset_index())
orientations_c
```

[17]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.97   269.85 650.00  POINT (1270.949 2656.847)      1.00         C

        X       Y
0 1270.95 2656.85
```

[18]:
```
orientations_c1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C1'].sort_values(by='Z', ascending=True).reset_index())
orientations_c1
```

[18]:
```
    dip  azimuth      Z                   geometry  polarity formation       X  \
0 23.26   269.27 550.00  POINT (2265.629 516.591)      1.00        C1 2265.63

       Y
0 516.59
```

[19]:
```
orientations_b = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B'].sort_values(by='Z', ascending=True).reset_index())
orientations_b
```

[19]:
```
    dip  azimuth      Z                    geometry  polarity formation  \
0 22.35   269.26 650.00   POINT (778.778 2702.073)      1.00         B
1 22.74   269.74 750.00  POINT (1023.571 2506.812)      1.00         B

        X       Y
0  778.78 2702.07
1 1023.57 2506.81
```

[20]:
```
orientations_b1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B1'].sort_values(by='Z', ascending=True).reset_index())
orientations_b1
```

[20]:
```
    dip  azimuth      Z                   geometry  polarity formation       X  \
0 22.32   269.37 550.00  POINT (1771.304 559.376)      1.00        B1 1771.30

       Y
0 559.38
```

Merging Orientations

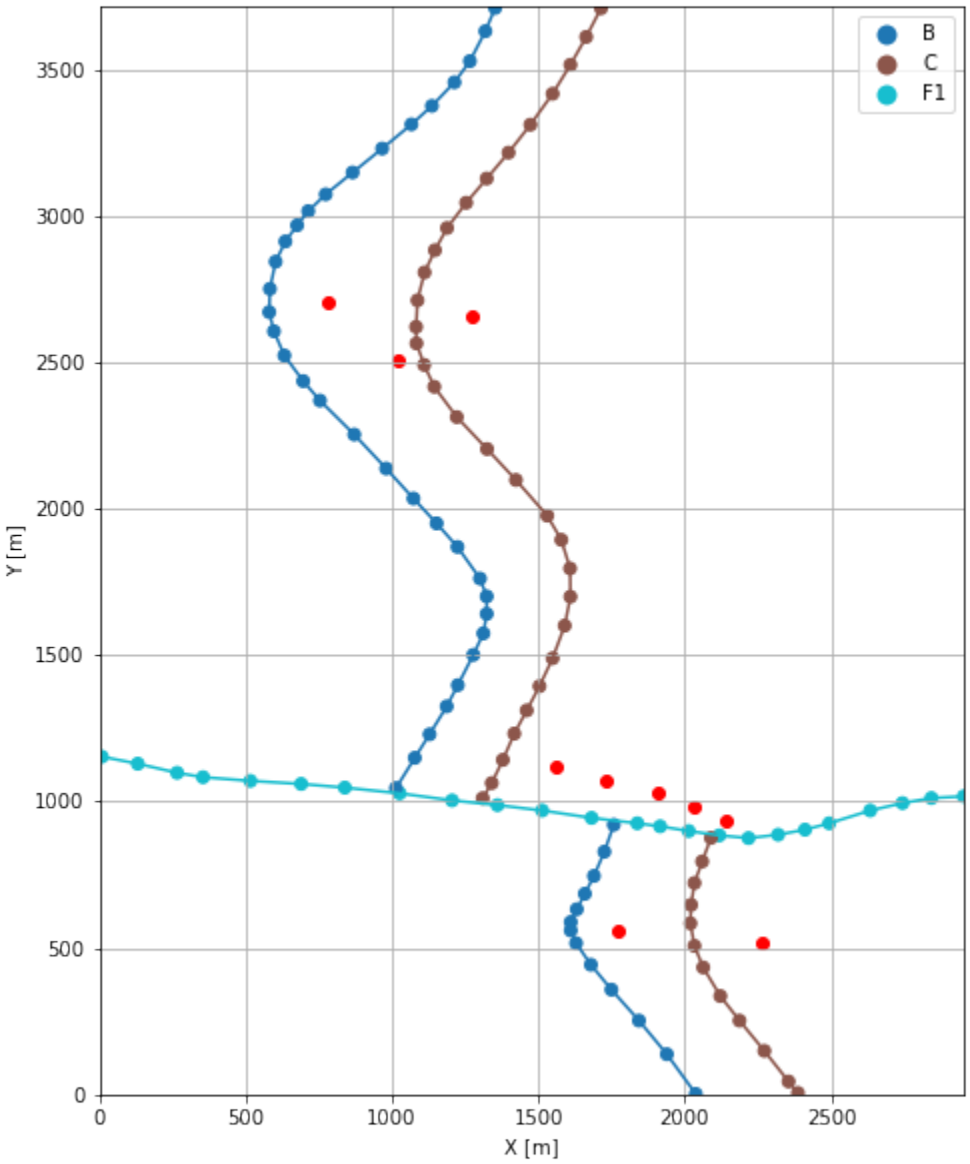
[21]:
```
import pandas as pd
orientations = pd.concat([orientations_f1, orientations_c, orientations_c1, orientations_
↪b, orientations_b1]).reset_index()
orientations['formation'] = ['F1', 'F1', 'F1', 'F1', 'F1', 'C', 'C', 'B', 'B', 'B']
orientations = orientations[orientations['formation'].isin(['F1', 'C', 'B'])]
orientations
```

[21]:
```
   index   dip  azimuth      Z                   geometry  polarity formation  \
0      0 64.17   180.06 550.00   POINT (2142.443 936.831)      1.00        F1
1      1 65.03   180.15 650.00   POINT (2034.906 985.072)      1.00        F1
2      2 65.86   180.16 750.00  POINT (1907.556 1031.877)      1.00        F1
3      3 70.31   180.05 850.00  POINT (1732.539 1074.232)      1.00        F1
4      4 66.70   180.12 950.00  POINT (1559.676 1118.740)      1.00        F1
5      0 22.97   269.85 650.00  POINT (1270.949 2656.847)      1.00         C
6      0 23.26   269.27 550.00   POINT (2265.629 516.591)      1.00         C
7      0 22.35   269.26 650.00   POINT (778.778 2702.073)      1.00         B
8      1 22.74   269.74 750.00  POINT (1023.571 2506.812)      1.00         B
9      0 22.32   269.37 550.00   POINT (1771.304 559.376)      1.00         B

        X       Y
0 2142.44  936.83
1 2034.91  985.07
2 1907.56 1031.88
3 1732.54 1074.23
4 1559.68 1118.74
5 1270.95 2656.85
6 2265.63  516.59
7  778.78 2702.07
8 1023.57 2506.81
9 1771.30  559.38
```

Plotting the Orientations

[22]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2957)
ax.set_ylim(0, 3715)
```

[22]:
```
(0.0, 3715.0)
```



7.8.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[23]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[24]:
```
geo_model = gp.create_model('Model8')
geo_model
```

[24]:
```
Model8  2022-04-05 09:57
```

Initiate Data

[25]:
```
gp.init_data(geo_model, [0, 2957, 0, 3715, 0, 1250], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[25]:
```
Model8  2022-04-05 09:57
```

Model Surfaces

[26]:
```
geo_model.surfaces
```

[26]:

	surface	series	order_surfaces	color	id
0	F1	Default series	1	#015482	1
1	C	Default series	2	#9f0052	2
2	B	Default series	3	#ffbe00	3

Mapping the Stack to Surfaces

[27]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('F1'),
                          'Strata1': ('C', 'B'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('A')
geo_model.set_is_fault(['Fault1'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[27]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault1	1	Fault	True	True	False
Strata1	2	Erosion	True	False	False

Showing the Number of Data Points

[28]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[28]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	F1	Fault1	1	#527682	1	24	5
1	C	Strata1	1	#9f0052	2	43	2
2	B	Strata1	2	#ffbe00	3	47	3
3	A	Strata1	3	#728f02	4	0	0

Loading Digital Elevation Model

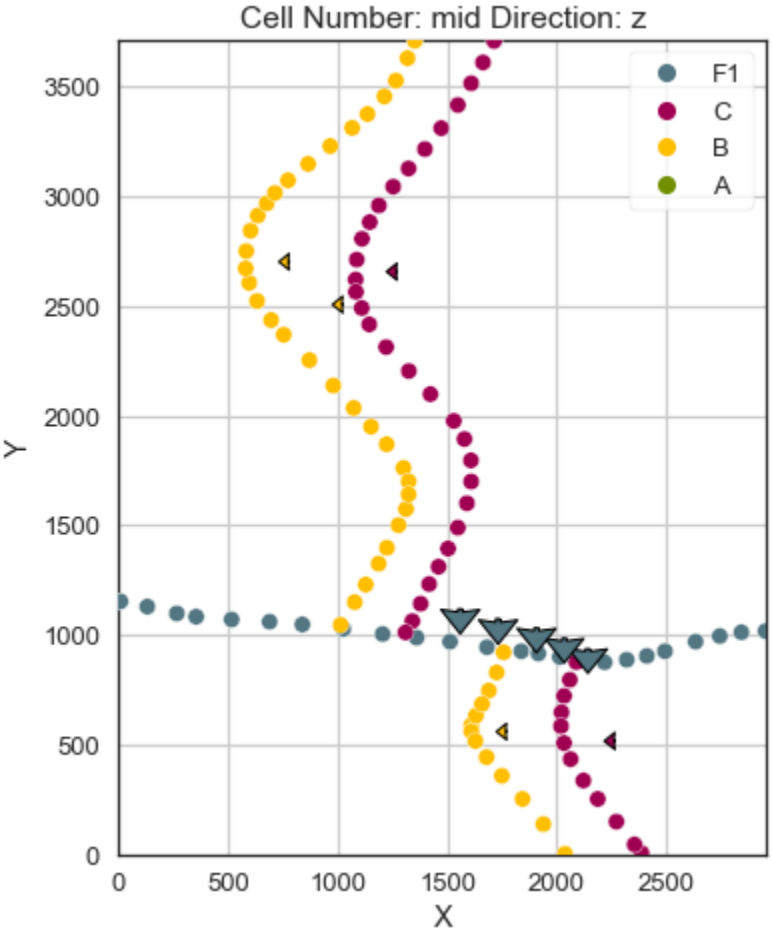
[29]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster8.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

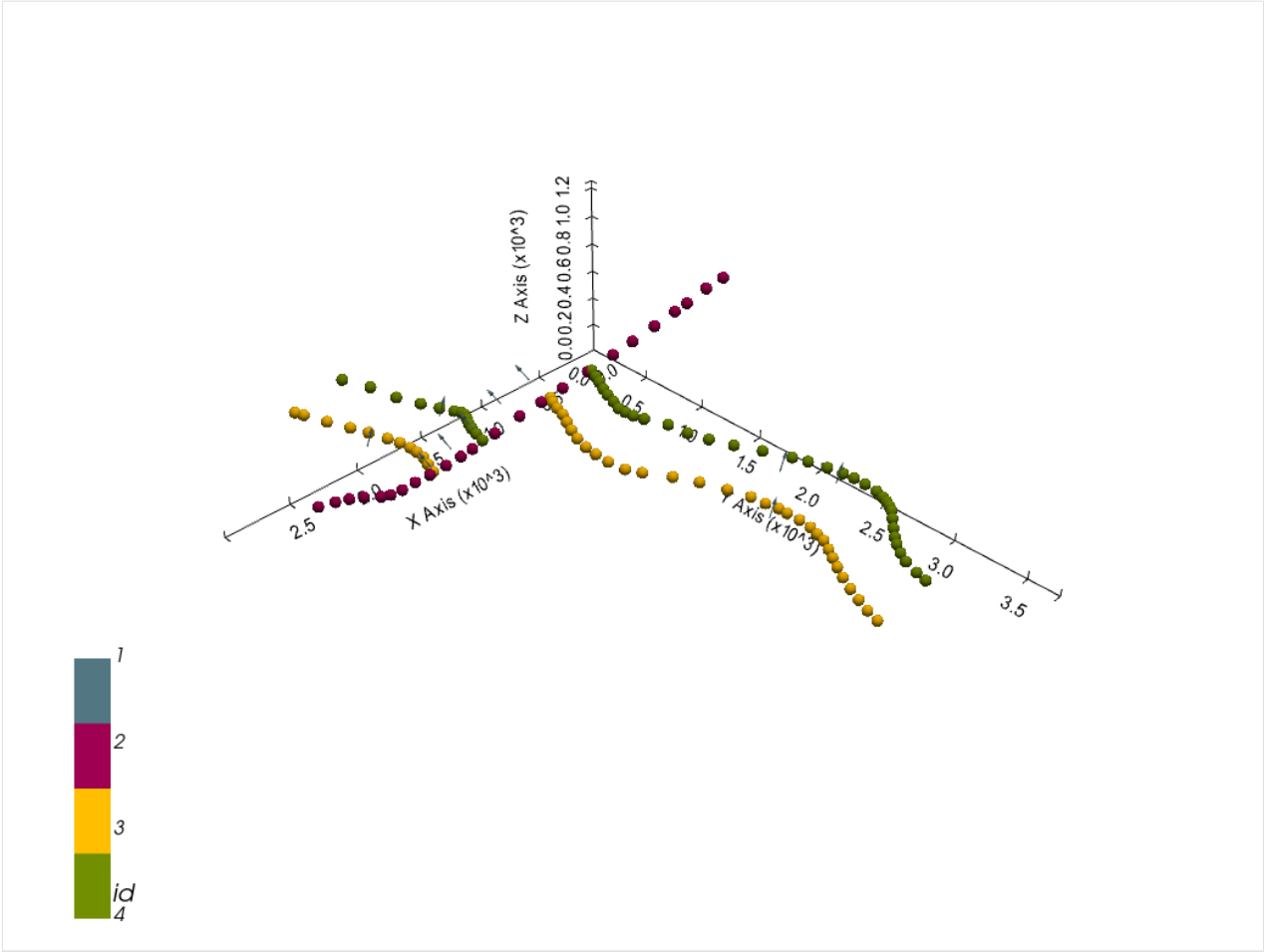
[29]:
```
Grid Object. Values:
array([[  14.785     ,   18.575     ,    6.25      ],
       [  14.785     ,   18.575     ,   18.75      ],
       [  14.785     ,   18.575     ,   31.25      ],
       ...,
       [2952.00506757, 3690.03360215,  931.20074463],
       [2952.00506757, 3700.02016129,  933.94940186],
       [2952.00506757, 3710.00672043,  936.69573975]])
```

Plotting Input Data

[30]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[31]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[31]:
```
<gempy.plot.vista.GemPyToVista at 0x26187de3df0>
```

Setting the Interpolator

[32]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                    values
range              4909.95
$C_o$            573989.86
drift equations     [3, 3]
```

[32]:
```
<gempy.core.interpolator.InterpolatorModel at 0x26186170bb0>
```

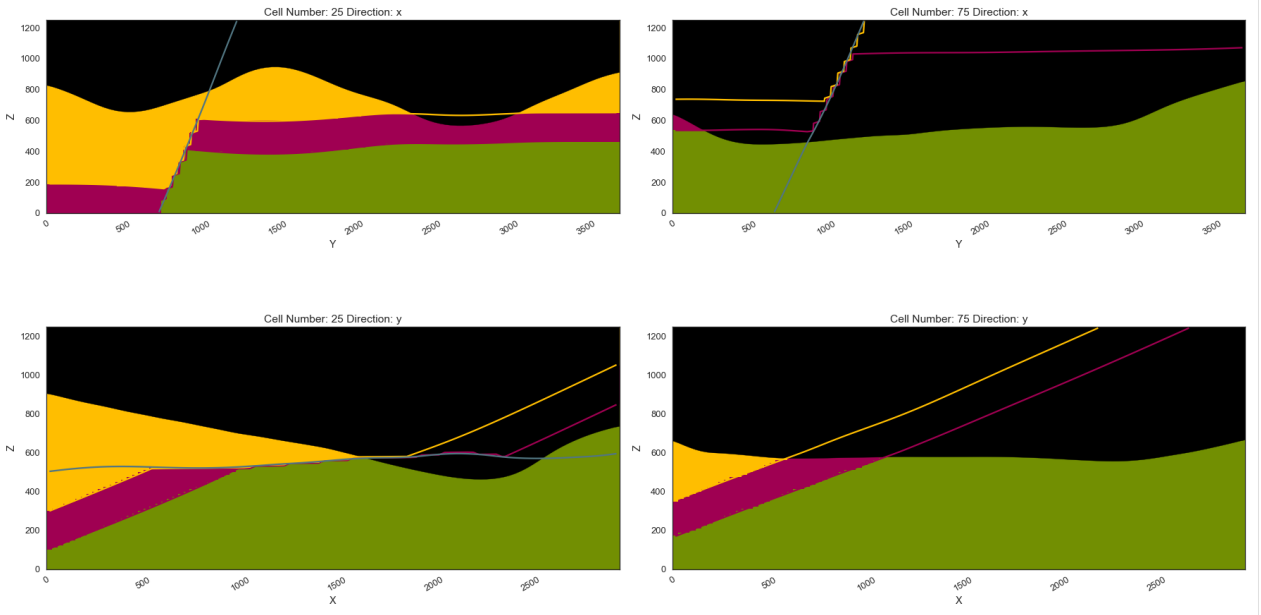
Computing Model

[33]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

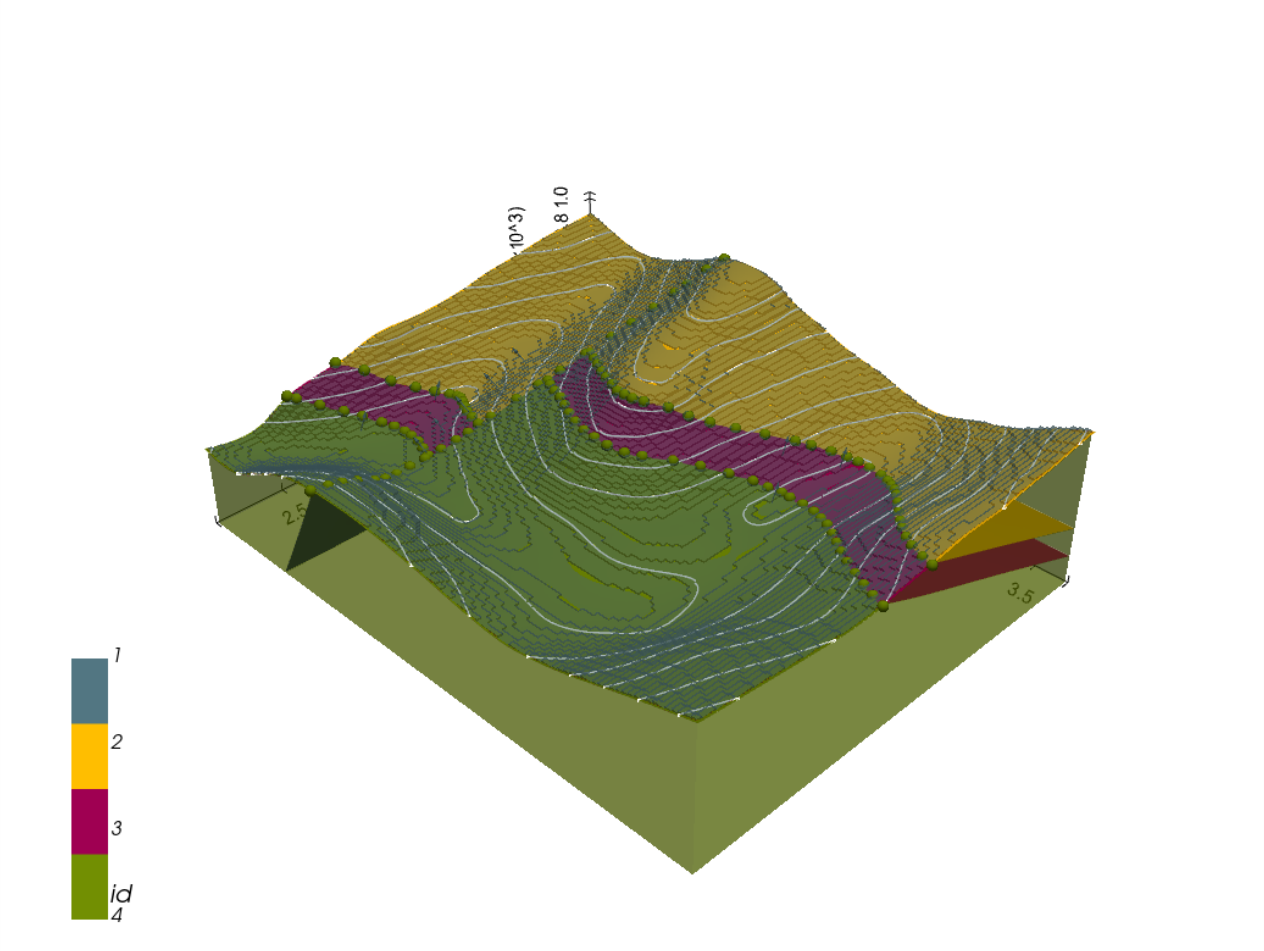
[34]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[34]:
```
<gempy.plot.visualization_2d.Plot2D at 0x261896b5c40>
```



Plotting 3D Model

[35]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



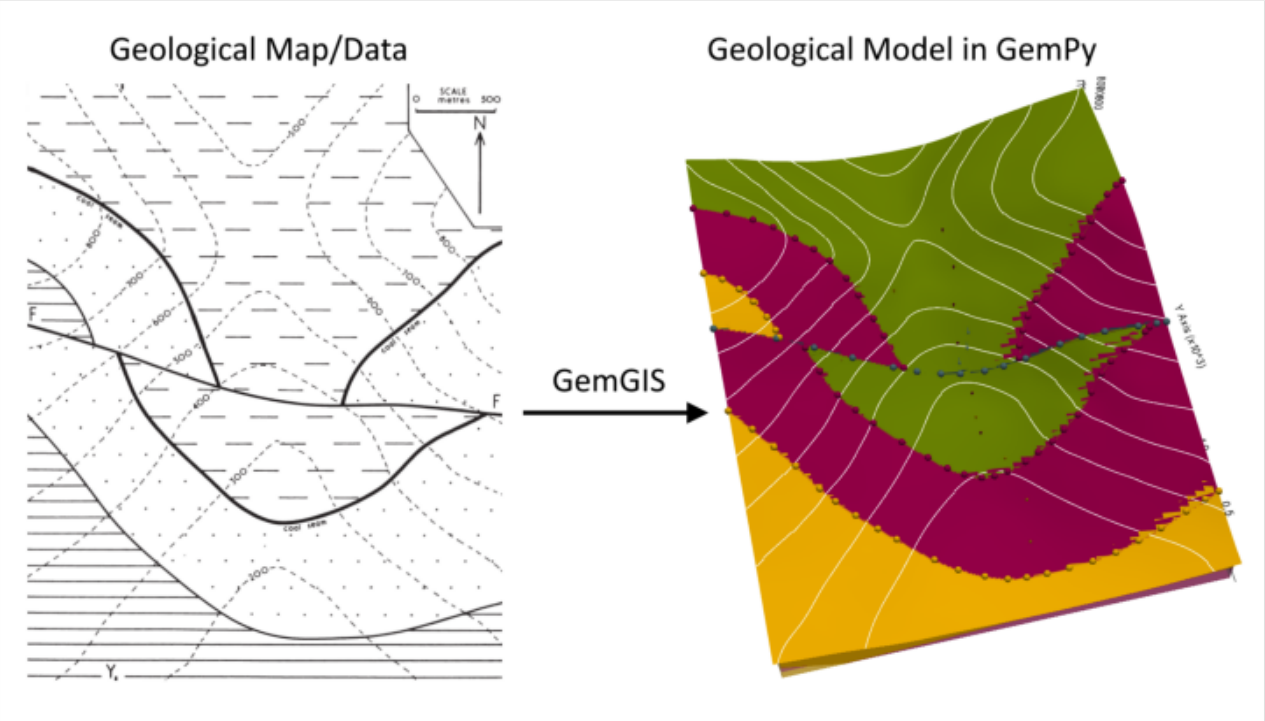
[ ]:

7.9 Example 9 - Faulted Layers

This example will show how to convert the geological map below using GemGIS to a GemPy model. This example is based on digitized data. The area is 2977 m wide (W-E extent) and 3731 m high (N-S extent). The vertical model extent varies from 0 m to 1000 m. The model represents two faulted layers (green and red) above a yellow basement. The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the GemPy model. The contour lines were also digitized and will be interpolated with GemGIS to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example09.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.9.1 Licensing

7.9.2 Import GemGIS

If you have installed `GemGIS` via pip and conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.9.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

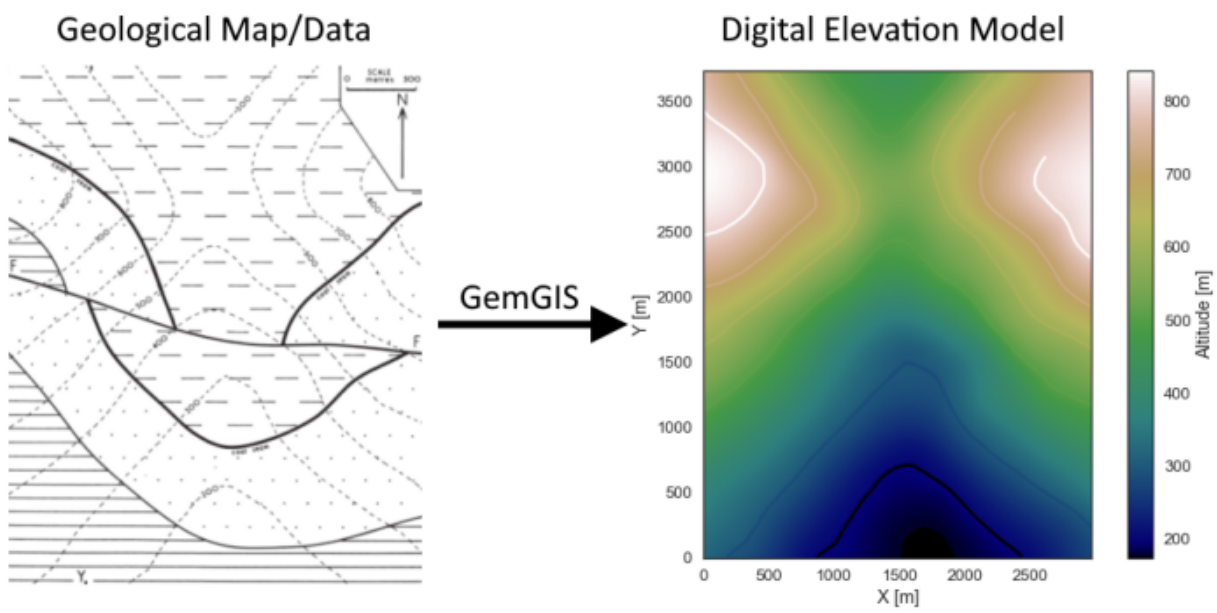
[4]:
```
file_path = 'data/example09/'
gg.download_gemgis_data.download_tutorial_data(filename="example09_faulted_layers.zip",␣
↪dirpath=file_path)
```

```
Downloading file 'example09_faulted_layers.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample09_faulted_layers.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\example\data\example09'.
```

7.9.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example09.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo9.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	200	LINESTRING (867.721 3.054, 905.213 37.662, 962...
1	None	300	LINESTRING (180.750 7.091, 266.116 75.731, 321...
2	None	400	LINESTRING (3.094 646.765, 63.082 692.332, 171...
3	None	500	LINESTRING (3.671 1124.357, 90.768 1186.652, 2...
4	None	600	LINESTRING (4.825 1626.175, 82.693 1677.510, 1...

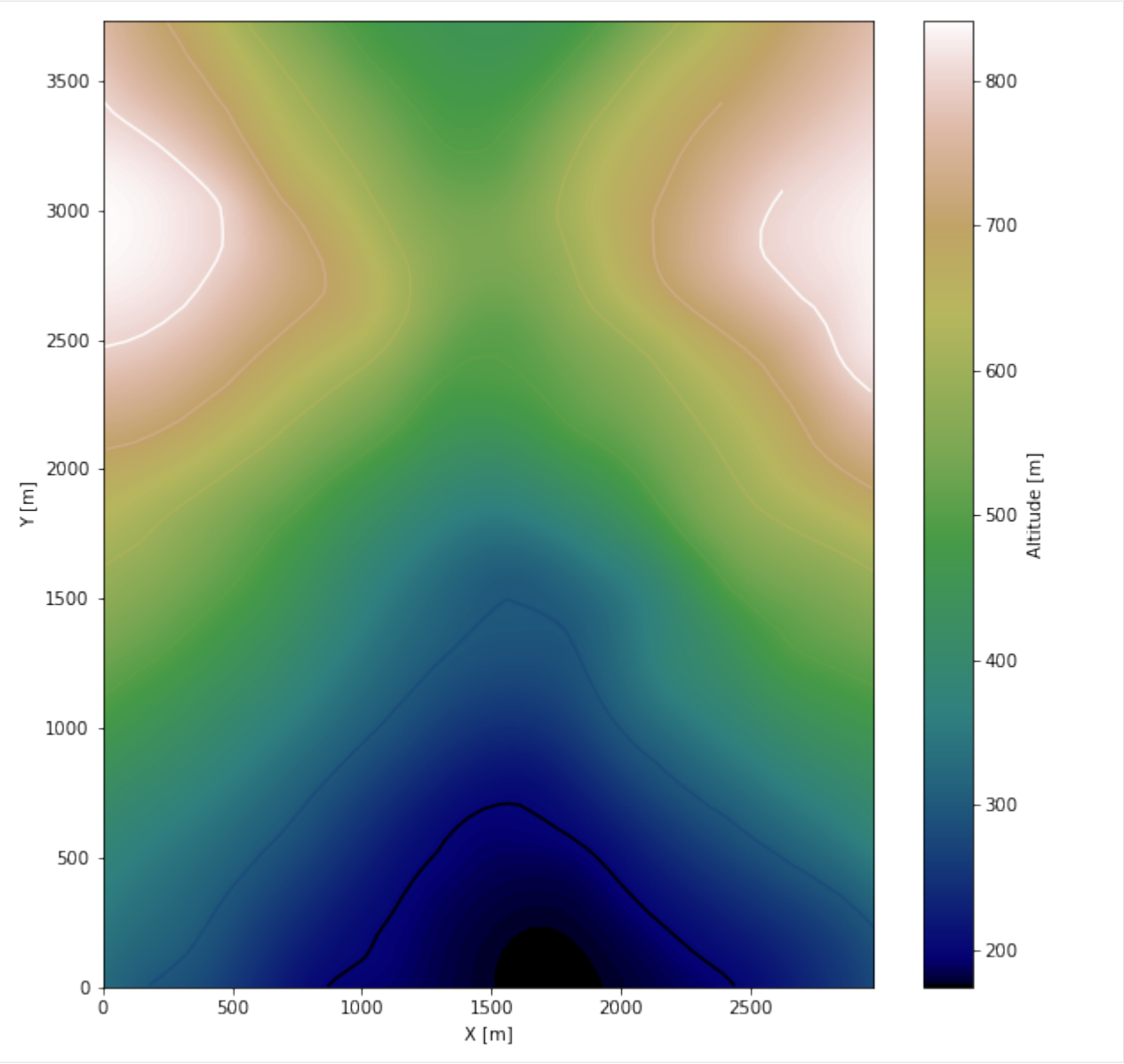
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 2977, 0, 3731], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2977)
ax.set_ylim(0, 3731)
```

[8]:
```
(0.0, 3731.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster9.tif', extent = [0, 2977, 0, 3731], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

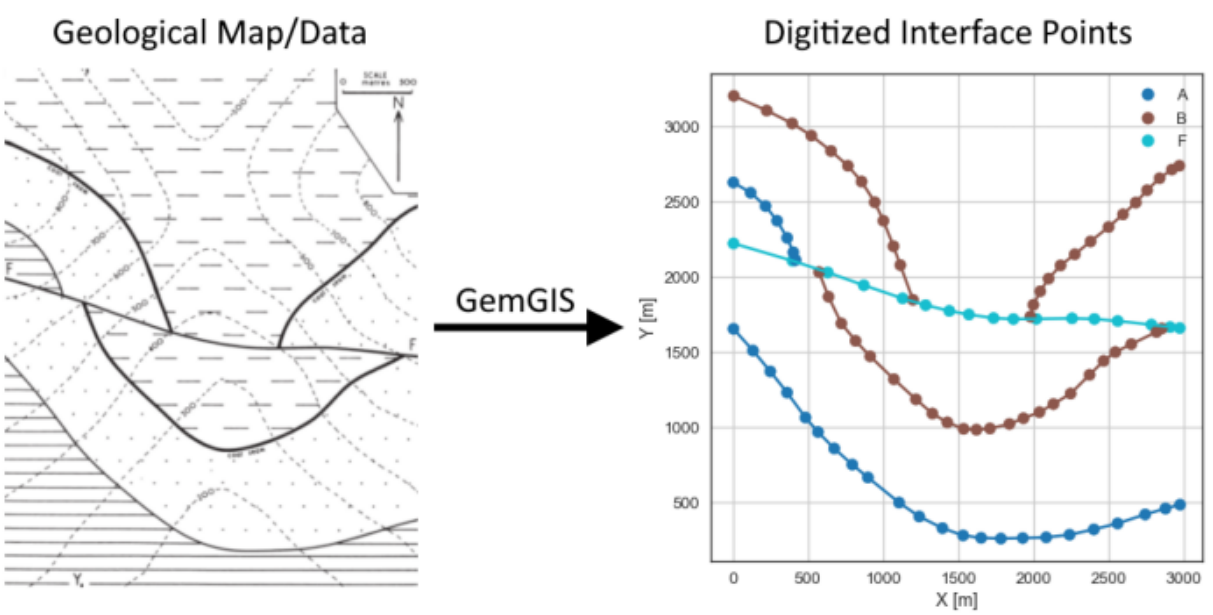
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster9.tif')
```

7.9.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example09.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces9.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	A	LINESTRING (4.825 1655.015, 134.029 1510.814, ...
1	None	B	LINESTRING (573.552 2032.244, 637.000 1868.432...
2	None	A	LINESTRING (3.671 2628.080, 119.032 2559.441, ...
3	None	B	LINESTRING (6.555 3203.152, 225.740 3105.672, ...
4	None	B	LINESTRING (1979.796 1736.344, 1999.407 1814.7...

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

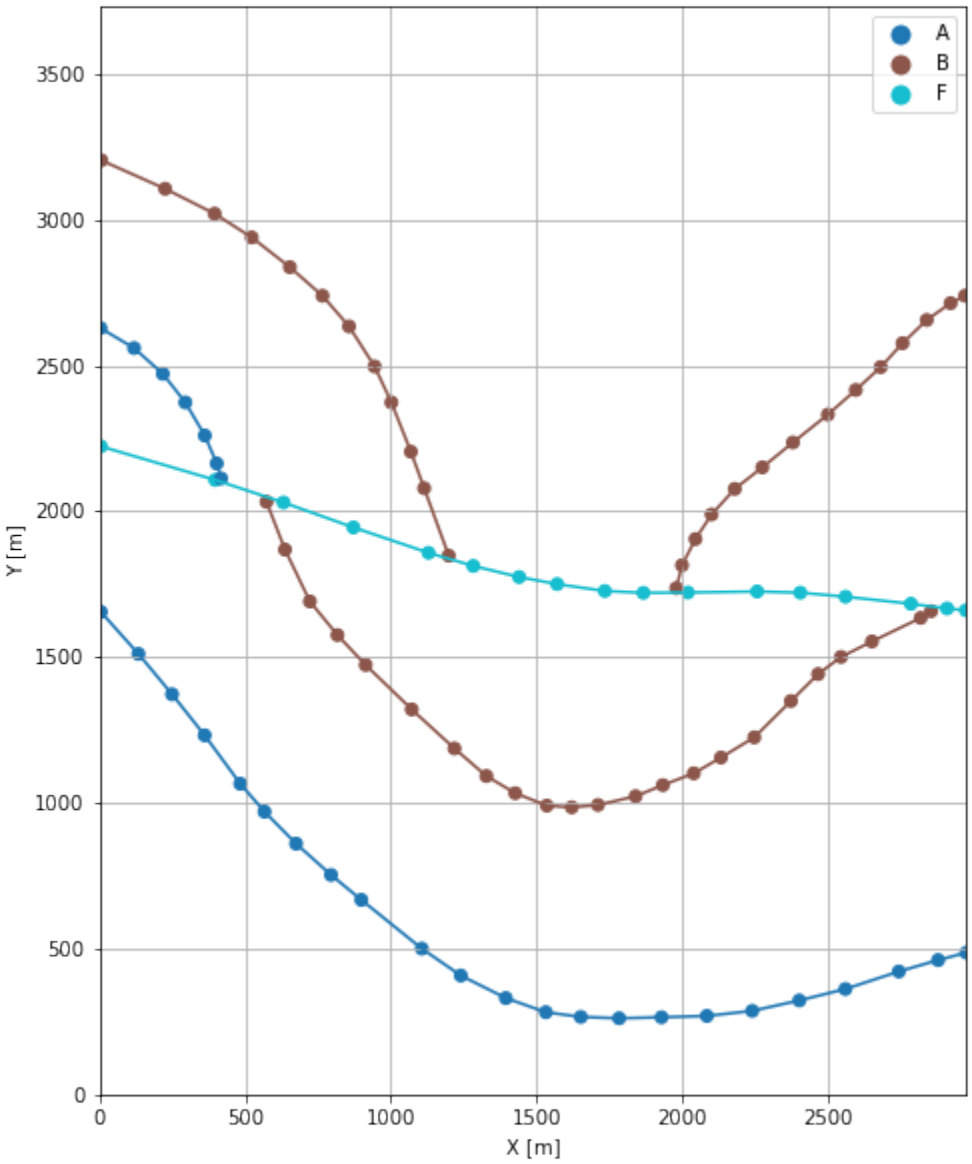
[12]:

	formation	geometry	X	Y	Z
95	F	POINT (2973.626 1660.206)	2973.63	1660.21	612.70
87	F	POINT (1734.655 1726.538)	1734.65	1726.54	351.47
79	F	POINT (4.825 2222.588)	4.82	2222.59	741.10
80	F	POINT (397.627 2107.228)	397.63	2107.23	655.76
81	F	POINT (631.809 2029.936)	631.81	2029.94	577.42

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2977)
ax.set_ylim(0, 3731)
```

[13]:
```
(0.0, 3731.0)
```



7.9.6 Orientations from Strike Lines

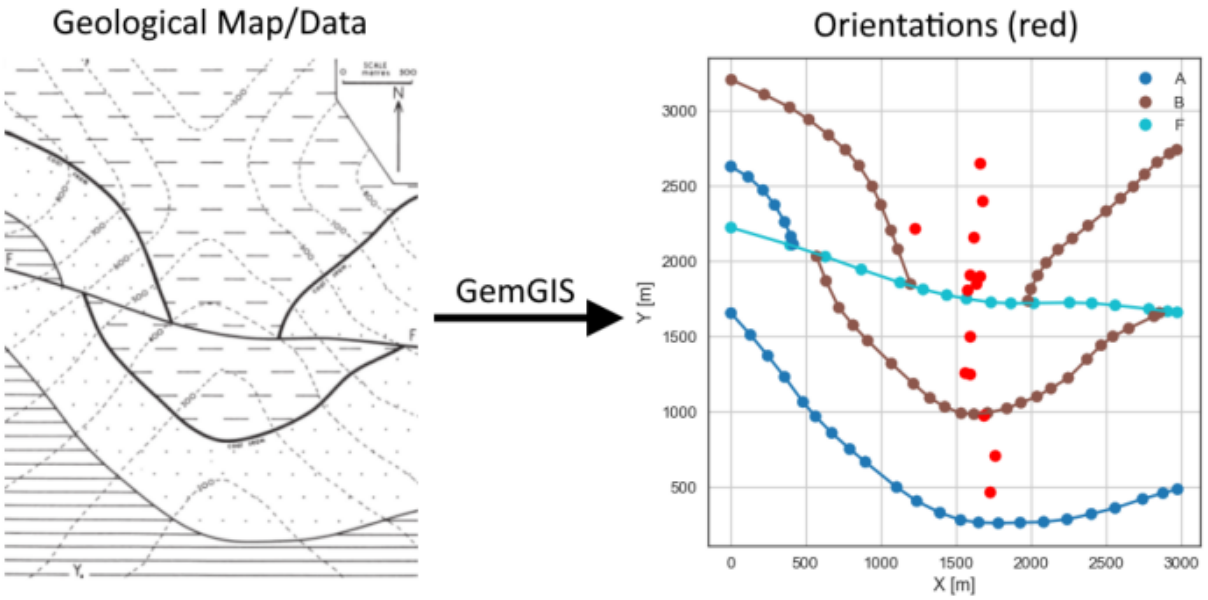
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/orientations_example09.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes9.shp')
strikes.head()
```

[15]:
```
   id formation    Z                                          geometry
0   1         F  400  LINESTRING (1196.498 1839.015, 1975.758 1721.924)
1   2         F  500   LINESTRING (863.107 1948.030, 2263.005 1724.231)
2   3         F  600   LINESTRING (570.091 2050.413, 2855.670 1673.473)
3   4         F  700   LINESTRING (235.257 2155.391, 2973.914 1720.194)
4   1         A  200    LINESTRING (1210.630 428.734, 2123.419 273.285)
```

Calculate Orientations for each formation

[16]:
```
orientations_f = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'F'].sort_values(by='Z', ascending=True).reset_index())
orientations_f
```

[16]:
```
    dip  azimuth      Z                  geometry  polarity formation  \
0 64.53   188.96 450.00  POINT (1574.592 1808.300)      1.00         F
1 65.13   189.29 550.00  POINT (1637.968 1849.037)      1.00         F
2 62.92   189.17 650.00  POINT (1658.733 1899.867)      1.00         F

        X       Y
0 1574.59 1808.30
1 1637.97 1849.04
2 1658.73 1899.87
```

[17]:
```
orientations_a = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'A'].sort_values(by='Z', ascending=True).reset_index())
orientations_a
```

[17]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.42   189.39 250.00   POINT (1722.398 466.081)      1.00         A
1 22.33   189.31 350.00   POINT (1758.376 707.978)      1.00         A
2 21.48   189.15 450.00   POINT (1681.589 975.542)      1.00         A
3 21.43   189.10 550.00  POINT (1558.442 1257.310)      1.00         A

        X       Y
0 1722.40  466.08
1 1758.38  707.98
2 1681.59  975.54
3 1558.44 1257.31
```

[18]:
```
orientations_a1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'A1'].sort_values(by='Z', ascending=True).reset_index())
orientations_a1
```

[18]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.20   189.33 750.00  POINT (1228.511 2222.011)      1.00        A1

        X       Y
0 1228.51 2222.01
```

[19]:
```
orientations_b = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B'].sort_values(by='Z', ascending=True).reset_index())
orientations_b
```

[19]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 21.25   189.23 350.00  POINT (1589.589 1252.119)      1.00         B
1 23.14   189.46 450.00  POINT (1591.463 1503.028)      1.00         B

        X       Y
0 1589.59 1252.12
1 1591.46 1503.03
```

[20]:
```
orientations_b1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B1'].sort_values(by='Z', ascending=True).reset_index())
orientations_b1
```

[20]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.28   189.22 450.00  POINT (1590.166 1911.692)      1.00        B1
1 22.28   189.36 550.00  POINT (1620.159 2159.140)      1.00        B1
2 22.54   189.09 650.00  POINT (1676.686 2398.513)      1.00        B1
3 22.20   189.22 750.00  POINT (1657.651 2651.729)      1.00        B1

        X       Y
0 1590.17 1911.69
1 1620.16 2159.14
2 1676.69 2398.51
3 1657.65 2651.73
```

Merging Orientations

[21]:
```
import pandas as pd
orientations = pd.concat([orientations_f, orientations_a, orientations_a1, orientations_
↪b, orientations_b1]).reset_index()
orientations['formation'] = ['F', 'F', 'F', 'A', 'A', 'A', 'A', 'A', 'B', 'B', 'B', 'B',
↪'B', 'B']
orientations = orientations[orientations['formation'].isin(['F', 'A', 'B'])]
orientations
```

[21]:
```
    index   dip  azimuth      Z                   geometry  polarity  \
0       0 64.53   188.96 450.00  POINT (1574.592 1808.300)      1.00
1       1 65.13   189.29 550.00  POINT (1637.968 1849.037)      1.00
2       2 62.92   189.17 650.00  POINT (1658.733 1899.867)      1.00
3       0 22.42   189.39 250.00   POINT (1722.398 466.081)      1.00
4       1 22.33   189.31 350.00   POINT (1758.376 707.978)      1.00
5       2 21.48   189.15 450.00   POINT (1681.589 975.542)      1.00
6       3 21.43   189.10 550.00  POINT (1558.442 1257.310)      1.00
7       0 22.20   189.33 750.00  POINT (1228.511 2222.011)      1.00
8       0 21.25   189.23 350.00  POINT (1589.589 1252.119)      1.00
9       1 23.14   189.46 450.00  POINT (1591.463 1503.028)      1.00
10      0 22.28   189.22 450.00  POINT (1590.166 1911.692)      1.00
11      1 22.28   189.36 550.00  POINT (1620.159 2159.140)      1.00
12      2 22.54   189.09 650.00  POINT (1676.686 2398.513)      1.00
13      3 22.20   189.22 750.00  POINT (1657.651 2651.729)      1.00

   formation       X       Y
0          F 1574.59 1808.30
1          F 1637.97 1849.04
2          F 1658.73 1899.87
3          A 1722.40  466.08
4          A 1758.38  707.98
5          A 1681.59  975.54
6          A 1558.44 1257.31
7          A 1228.51 2222.01
8          B 1589.59 1252.12
9          B 1591.46 1503.03
10         B 1590.17 1911.69
11         B 1620.16 2159.14
12         B 1676.69 2398.51
13         B 1657.65 2651.73
```

Plotting the Orientations

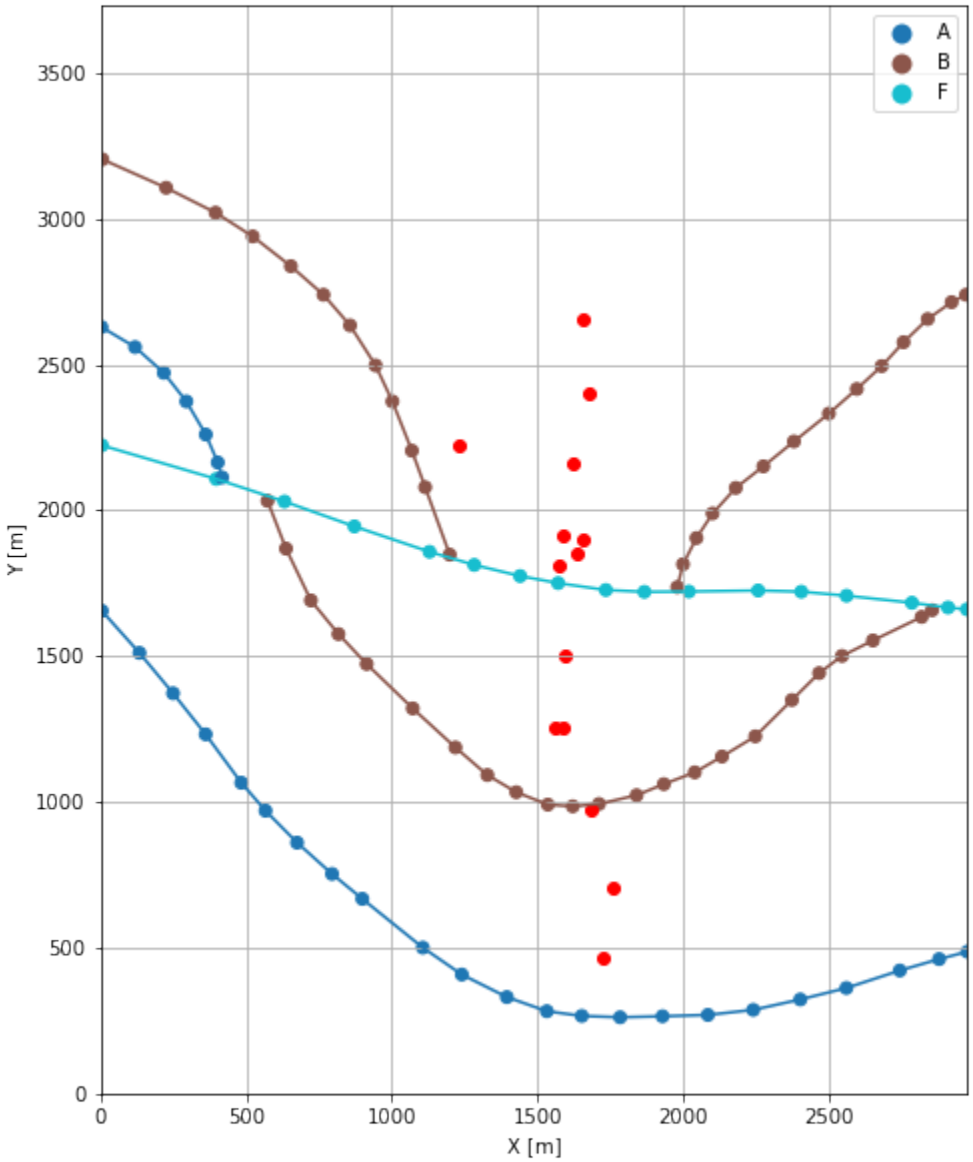
[22]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

(continues on next page)

(continued from previous page)

```
ax.set_xlim(0, 2977)
ax.set_ylim(0, 3731)
```

[22]: (0.0, 3731.0)



7.9.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[23]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[24]:
```
geo_model = gp.create_model('Model9')
geo_model
```

[24]:
```
Model9  2022-04-05 10:00
```

Initiate Data

[25]:
```
gp.init_data(geo_model, [0, 2977, 0, 3731, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[25]:
```
Model9  2022-04-05 10:00
```

Model Surfaces

[26]:
```
geo_model.surfaces
```

[26]:

	surface	series	order_surfaces	color	id
0	F	Default series	1	#015482	1
1	B	Default series	2	#9f0052	2
2	A	Default series	3	#ffbe00	3

Mapping the Stack to Surfaces

[27]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('F'),
                          'Strata1': ('B', 'A'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('C')
geo_model.set_is_fault(['Fault1'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[27]:
```
         order_series BottomRelation  isActive  isFault  isFinite
Fault1              1          Fault      True     True     False
Strata1             2        Erosion      True    False     False
```

Showing the Number of Data Points

[28]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[28]:
```
  surface   series  order_surfaces    color  id  No. of Interfaces  No. of Orientations
0       F   Fault1               1  #527682   1                 17                    3
1       B  Strata1               1  #9f0052   2                 49                    6
2       A  Strata1               2  #ffbe00   3                 30                    5
3       C  Strata1               3  #728f02   4                  0                    0
```

Loading Digital Elevation Model

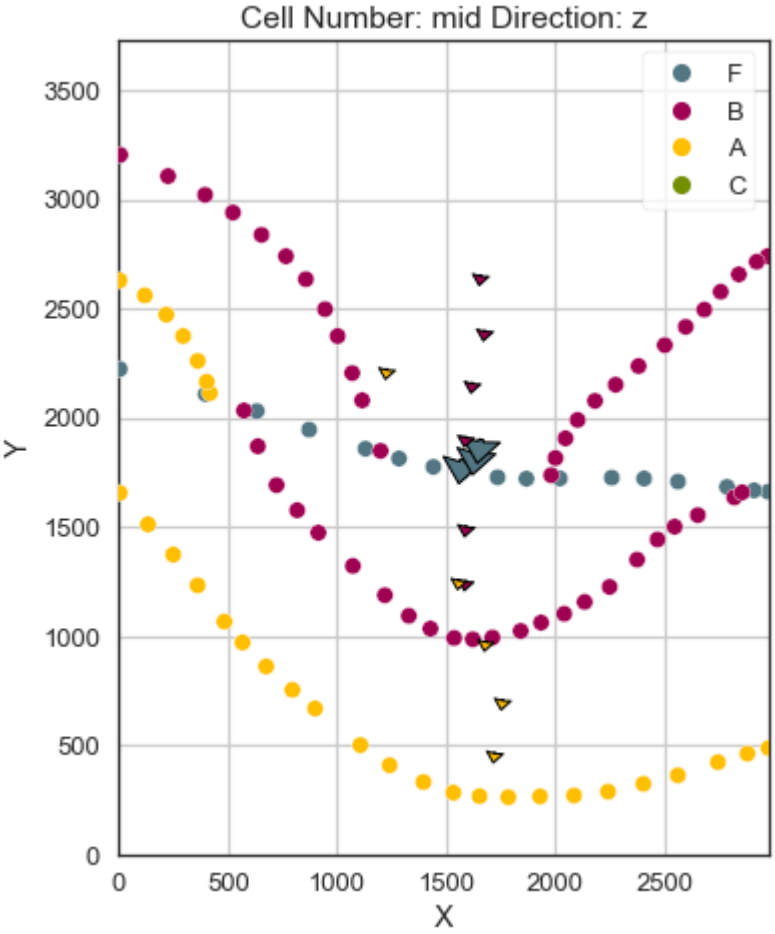
[29]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster9.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

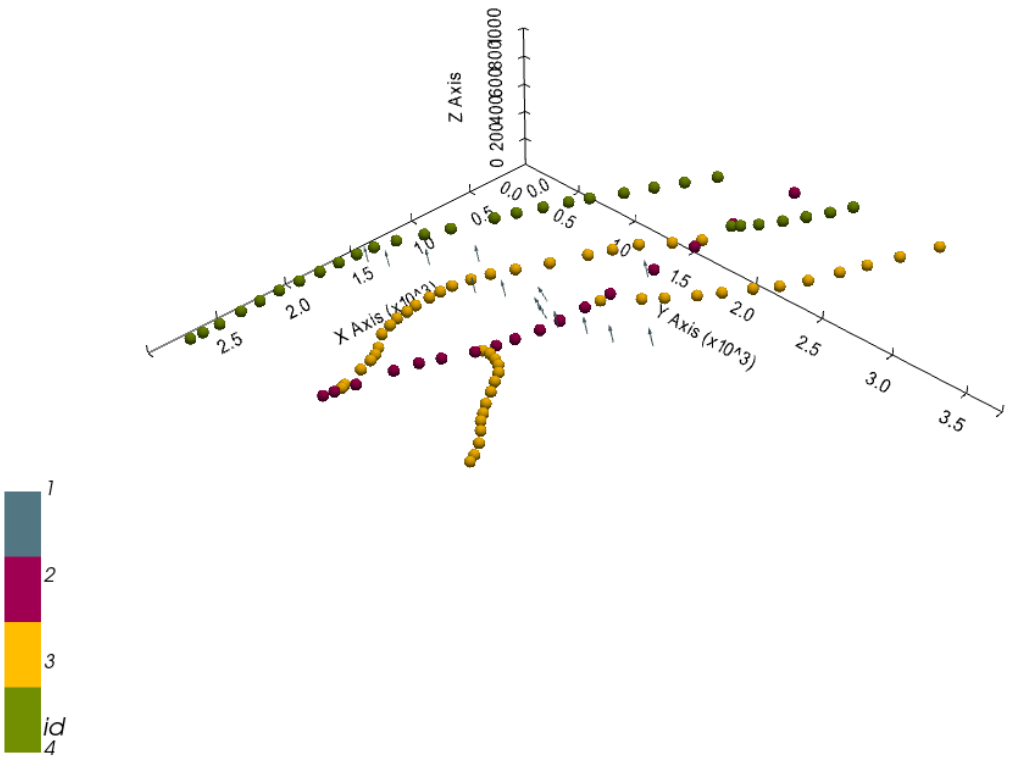
[29]:
```
Grid Object. Values:
array([[  14.885     ,   18.655     ,    5.        ],
       [  14.885     ,   18.655     ,   15.        ],
       [  14.885     ,   18.655     ,   25.        ],
       ...,
       [2972.00503356, 3705.99329759,  772.98675537],
       [2972.00503356, 3715.99597855,  772.22625732],
       [2972.00503356, 3725.99865952,  771.48016357]])
```

Plotting Input Data

[30]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[31]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[31]:
```
<gempy.plot.vista.GemPyToVista at 0x2a531e26640>
```

Setting the Interpolator

[32]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                    values
range              4876.77
$C_o$            566259.29
drift equations     [3, 3]
```

[32]:
```
<gempy.core.interpolator.InterpolatorModel at 0x2a52bddbbb0>
```

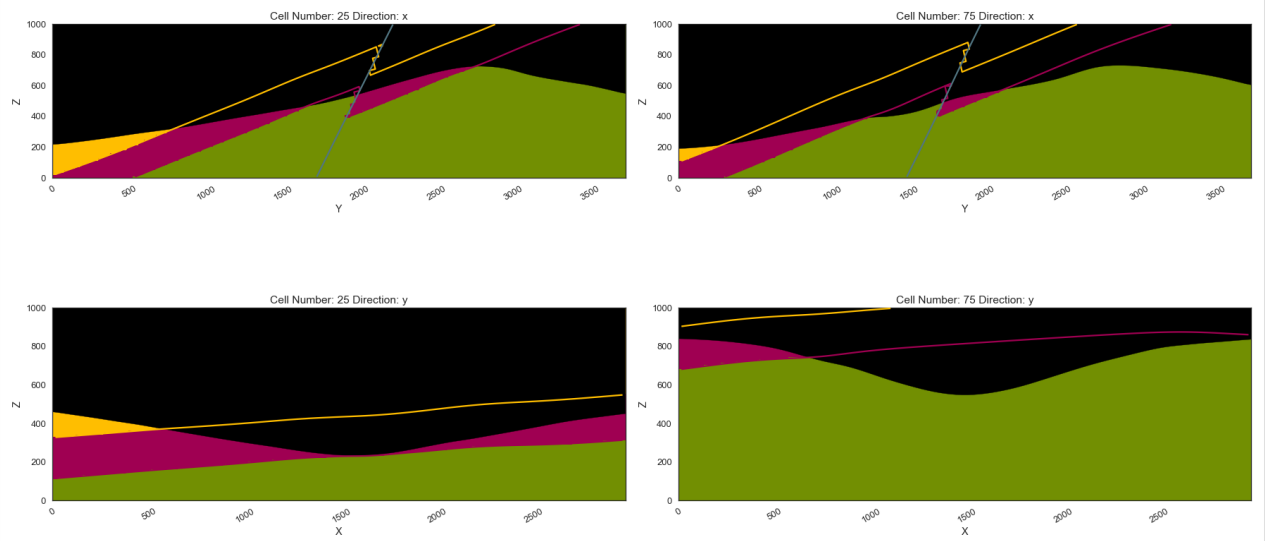
Computing Model

[33]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

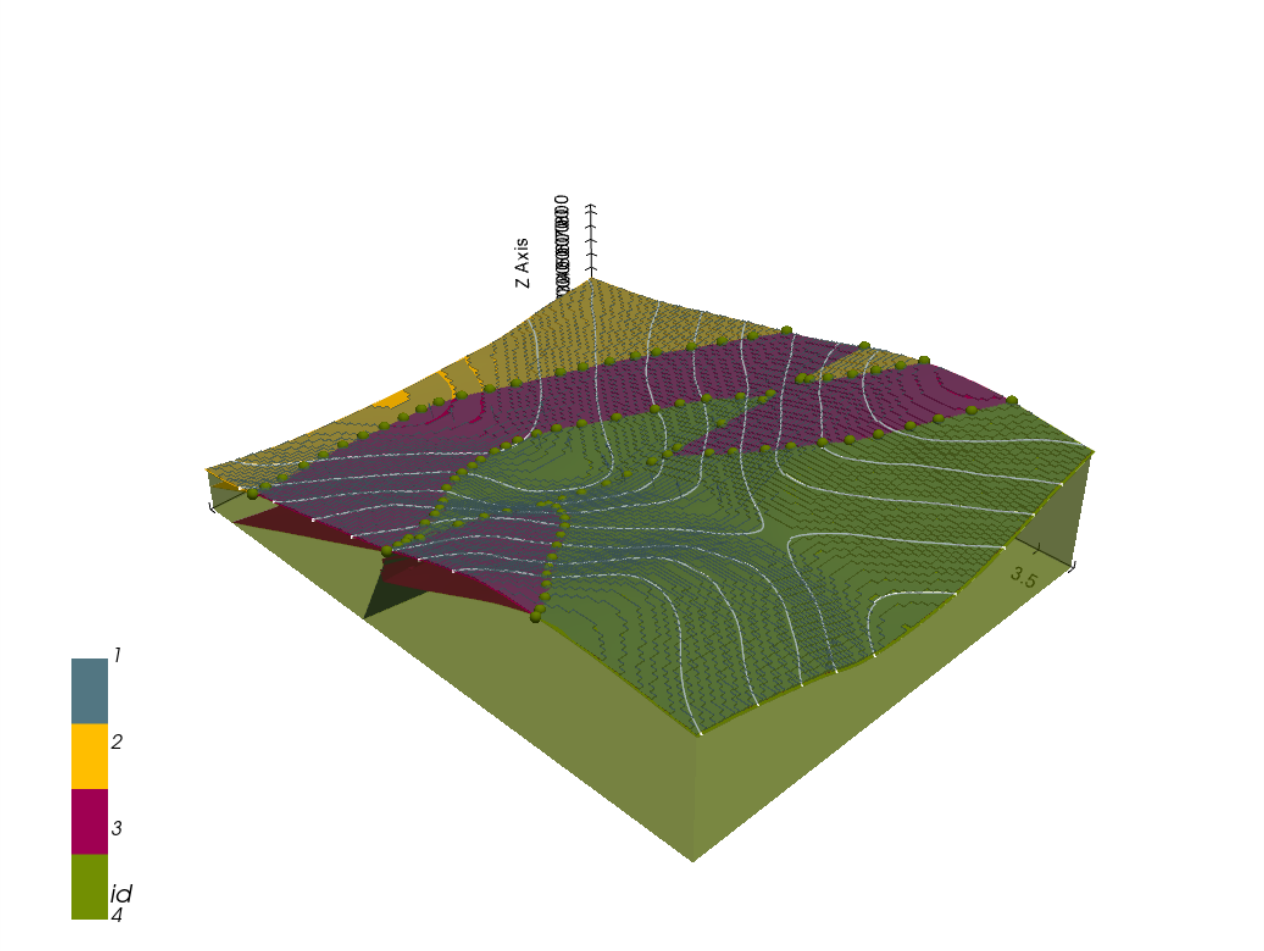
[34]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[34]:
```
<gempy.plot.visualization_2d.Plot2D at 0x2a52f05da90>
```



Plotting 3D Model

[35]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



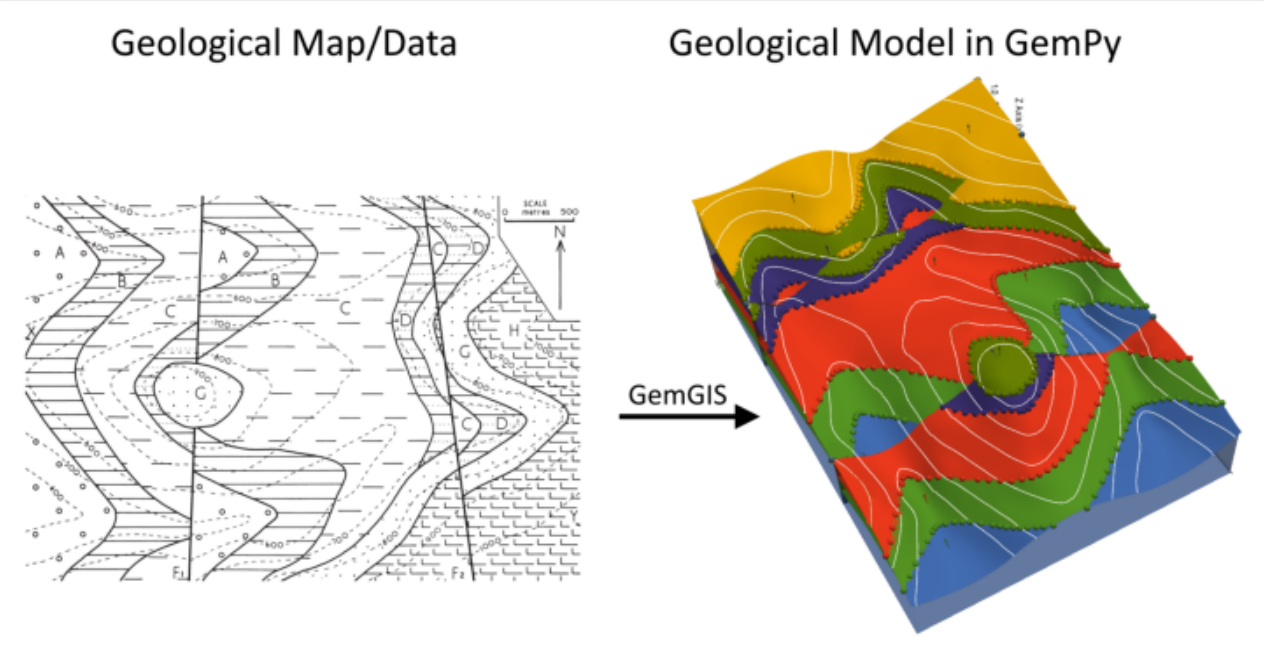
[ ]:

7.10 Example 10 - Faulted Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3954 m wide (W-E extent) and 2738 m high (N-S extent). The vertical model extent varies between 0 m and 1000 m. The model represents folded layers (yellow to light green) which are separated to a second set of layers (blue and purple) by an unconformity. A light blue layer represents the basement. The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example10.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.10.1 Licensing

7.10.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.10.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

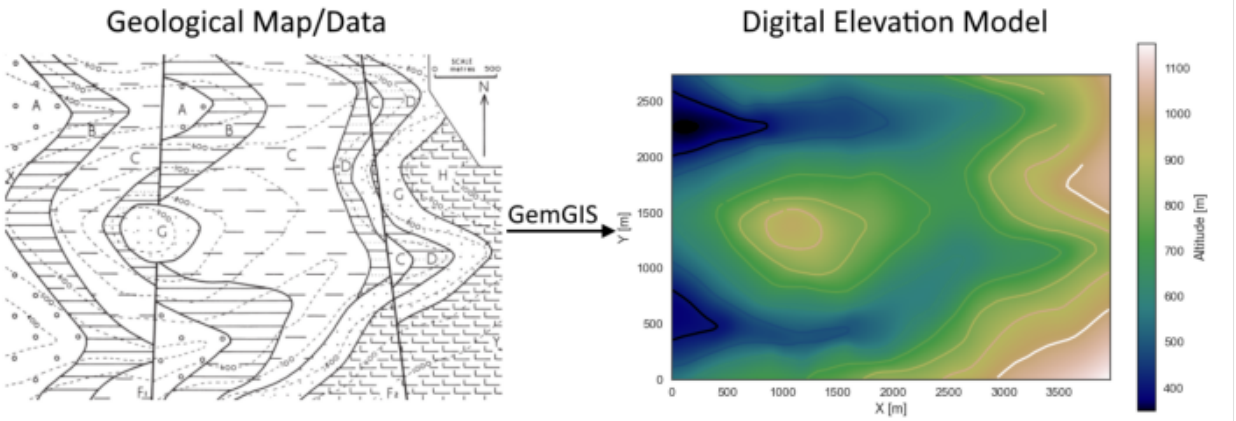
[4]:
```
file_path = 'data/example10/'
gg.download_gemgis_data.download_tutorial_data(filename="example10_faulted_folded_layers.
↪zip", dirpath=file_path)
```

```
Downloading file 'example10_faulted_folded_layers.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2Fexample10_faulted_folded_layers.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\example\data\example10'.
```

7.10.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example10.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo10.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	600	LINESTRING (500.103 2733.663, 594.070 2684.564...
1	None	500	LINESTRING (217.356 2726.044, 324.445 2643.506...
2	None	400	LINESTRING (11.222 2589.327, 69.634 2556.312, ...
3	None	900	LINESTRING (1037.237 1522.677, 1083.798 1526.9...
4	None	800	LINESTRING (912.795 1610.718, 990.677 1620.030...

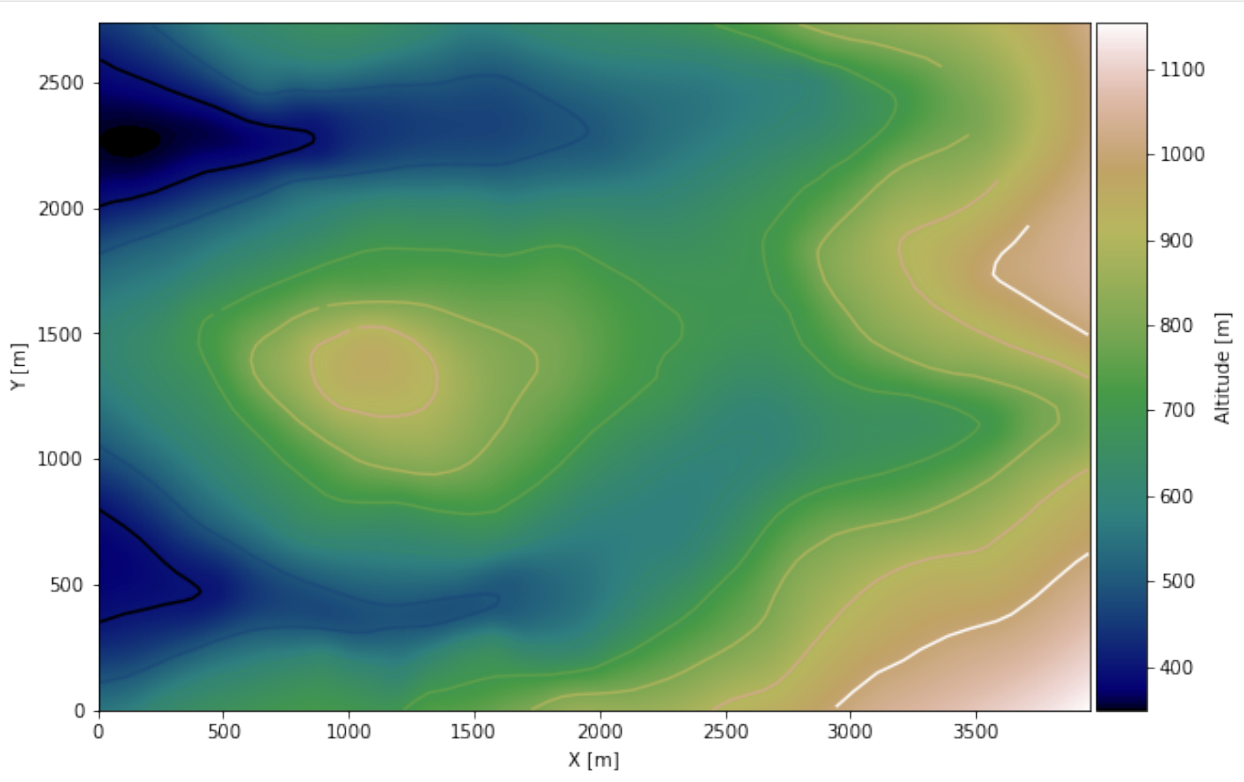
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 3954, 0, 2738], cmap='gist_earth
↪')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3954)
ax.set_ylim(0, 2738)
```

[8]:
```
(0.0, 2738.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster10.tif', extent = [0, 3954, 0, 2738], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

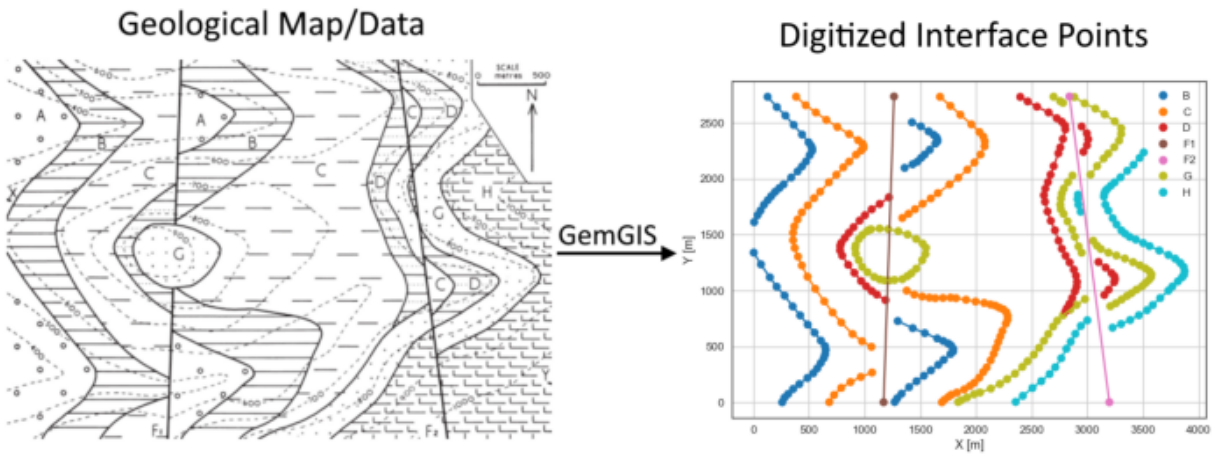
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster10.tif')
```

7.10.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example10.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces10.shp')
interfaces.head()
```

[11]:
```
     id formation                                           geometry
0  None        F1     LINESTRING (1263.266 2736.203, 1170.145 1.855)
1  None        F2     LINESTRING (2839.537 2736.203, 3198.049 1.855)
2  None         B  LINESTRING (128.046 2733.240, 211.007 2641.813...
```

(continues on next page)

(continued from previous page)

```
3  None         B  LINESTRING (3.603 1339.823, 67.941 1234.851, 1...
4  None         C  LINESTRING (681.688 2.279, 724.862 66.616, 789...
```

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

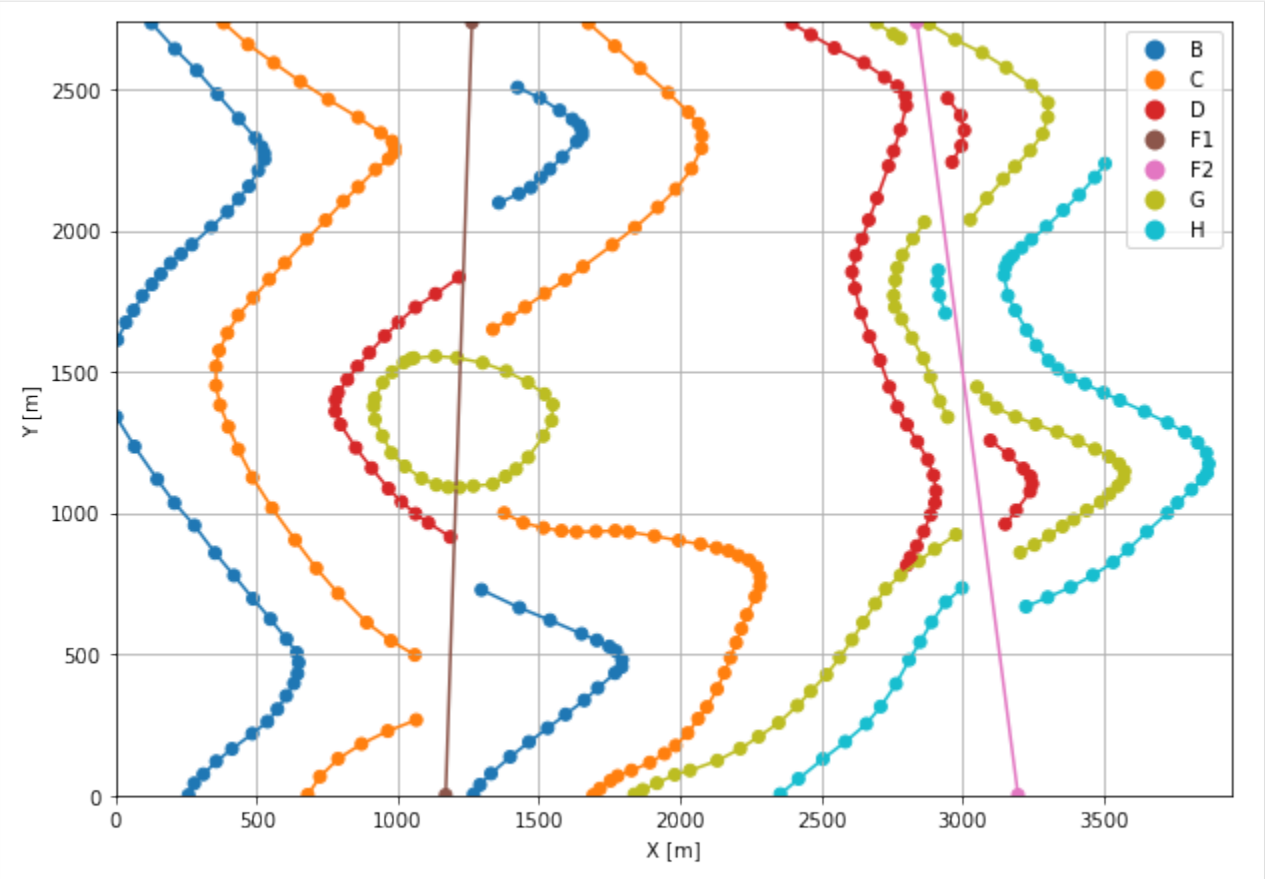
[12]:

	formation	geometry	X	Y	Z
322	H	POINT (2940.276 1706.378)	2940.28	1706.38	814.55
371	H	POINT (3262.810 1592.094)	3262.81	1592.09	879.04
369	H	POINT (3188.314 1716.537)	3188.31	1716.54	888.36
368	H	POINT (3162.071 1769.023)	3162.07	1769.02	887.87
367	H	POINT (3148.526 1840.979)	3148.53	1840.98	889.10

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3954)
ax.set_ylim(0, 2738)
```

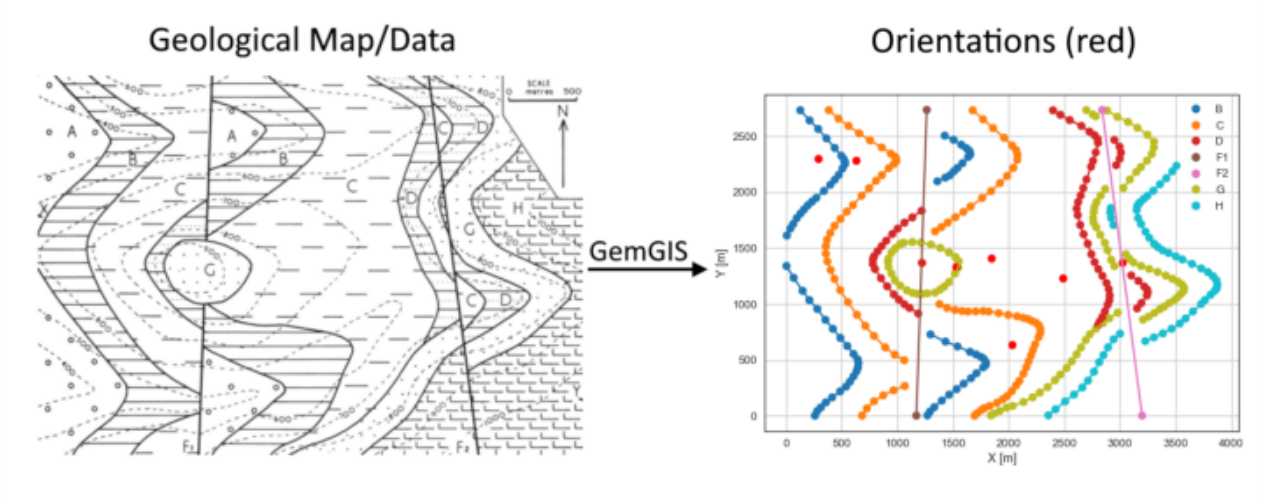
[13]:
```
(0.0, 2738.0)
```



7.10.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example10.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes10.shp')
strikes.head()
```

[15]:
```
    id formation    Z                                          geometry
0 1.00         B  400  LINESTRING (416.718 2417.478, 467.511 2152.509)
1 2.00         B  500   LINESTRING (216.087 1913.782, 39.158 2721.388)
2  NaN        B1  500    LINESTRING (505.606 676.130, 576.716 313.808)
3 1.00         C  500  LINESTRING (751.105 2464.038, 832.373 2115.261)
4 2.00         C  600  LINESTRING (549.626 1825.741, 375.238 2734.933)
```

Calculate Orientations for each formation

[16]:
```
orientations_f1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'F1'].sort_values(by='Z', ascending=True).reset_index())
orientations_f1
```

[16]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 89.75    91.97 1000.00  POINT (1216.176 1369.029)      1.00        F1

        X       Y
0 1216.18 1369.03
```

[17]:
```
orientations_f2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'F2'].sort_values(by='Z', ascending=True).reset_index())
orientations_f2
```

[17]:
```
    dip  azimuth       Z                   geometry  polarity formation  \
0 89.75    82.52 1000.00  POINT (3018.793 1368.818)      1.00        F2

        X       Y
0 3018.79 1368.82
```

[18]:
```
orientations_c2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C2'].sort_values(by='Z', ascending=True).reset_index())
orientations_c2
```

[18]:
```
    dip  azimuth      Z                    geometry  polarity formation  \
0 16.56    76.45 650.00  POINT (1845.267 1413.896)      1.00        C2
1 17.33    76.32 750.00  POINT (1524.849 1332.628)      1.00        C2

        X       Y
0 1845.27 1413.90
1 1524.85 1332.63
```

[19]:
```
orientations_c = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C'].sort_values(by='Z', ascending=True).reset_index())
orientations_c
```

[19]:
```
    dip  azimuth      Z                    geometry  polarity formation      X  \
0 17.45    78.85 550.00  POINT (627.085 2284.993)      1.00         C 627.09

        Y
0 2284.99
```

[20]:
```
orientations_c1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C1'].sort_values(by='Z', ascending=True).reset_index())
orientations_c1
```

[20]:
```
    dip  azimuth      Z                    geometry  polarity formation       X  \
0 16.85    76.63 650.00  POINT (2029.391 637.612)      1.00        C1 2029.39

       Y
0 637.61
```

[21]:
```
orientations_b = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B'].sort_values(by='Z', ascending=True).reset_index())
orientations_b
```

[21]:
```
    dip  azimuth      Z                    geometry  polarity formation      X  \
0 18.63    77.79 450.00  POINT (284.869 2301.289)      1.00         B 284.87

        Y
0 2301.29
```

[22]:
```
orientations_g = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'G'].sort_values(by='Z', ascending=True).reset_index())
orientations_g
```

[22]:
```
   dip  azimuth      Z                    geometry  polarity formation       X  \
0 6.00    90.92 750.00  POINT (2481.976 1233.264)      1.00         G 2481.98

        Y
0 1233.26
```

Merging Orientations

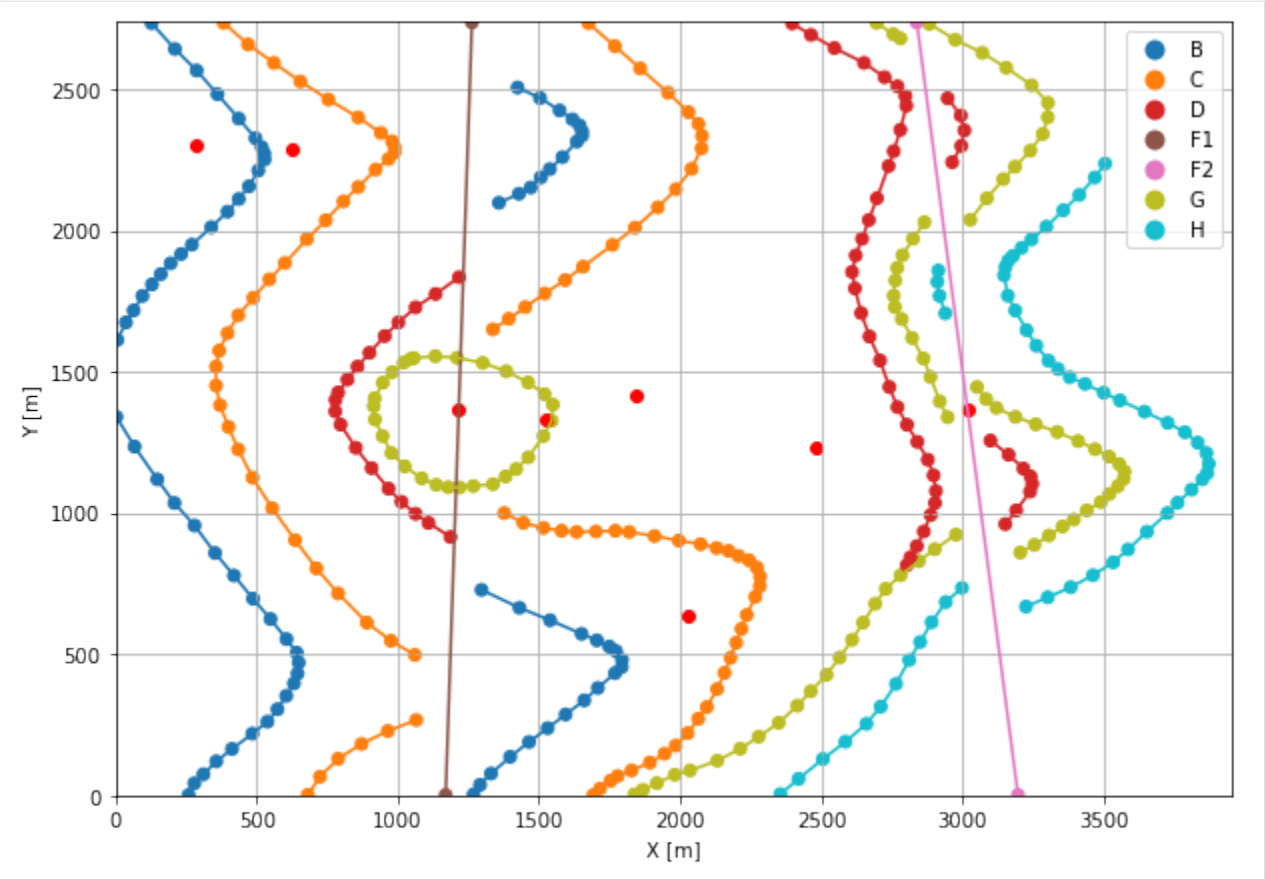
[23]:
```
import pandas as pd
orientations = pd.concat([orientations_f1, orientations_f2, orientations_c2,␣
↪orientations_c, orientations_c1, orientations_b, orientations_g]).reset_index()
orientations['formation'] = ['F1', 'F2', 'C', 'C', 'C', 'C', 'B', 'G']
orientations = orientations[orientations['formation'].isin(['F1', 'F2', 'C', 'B', 'G'])]
orientations
```

[23]:
```
   index   dip  azimuth       Z                   geometry  polarity  \
0      0 89.75    91.97 1000.00  POINT (1216.176 1369.029)      1.00
1      0 89.75    82.52 1000.00  POINT (3018.793 1368.818)      1.00
2      0 16.56    76.45  650.00  POINT (1845.267 1413.896)      1.00
3      1 17.33    76.32  750.00  POINT (1524.849 1332.628)      1.00
4      0 17.45    78.85  550.00   POINT (627.085 2284.993)      1.00
5      0 16.85    76.63  650.00   POINT (2029.391 637.612)      1.00
6      0 18.63    77.79  450.00   POINT (284.869 2301.289)      1.00
7      0  6.00    90.92  750.00  POINT (2481.976 1233.264)      1.00

  formation       X       Y
0        F1 1216.18 1369.03
1        F2 3018.79 1368.82
2         C 1845.27 1413.90
3         C 1524.85 1332.63
4         C  627.09 2284.99
5         C 2029.39  637.61
6         B  284.87 2301.29
7         G 2481.98 1233.26
```

Plotting the Orientations

[24]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3954)
ax.set_ylim(0, 2738)
```

[24]:
```
(0.0, 2738.0)
```



7.10.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[25]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[26]:
```
geo_model = gp.create_model('Model10')
geo_model
```

[26]:
```
Model10  2022-04-05 10:47
```

Initiate Data

[27]:
```
gp.init_data(geo_model, [0, 3954, 0, 2738, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[27]:
```
Model10  2022-04-05 10:47
```

Model Surfaces

[28]:
```
geo_model.surfaces
```

[28]:

	surface	series	order_surfaces	color	id
0	H	Default series	1	#015482	1
1	G	Default series	2	#9f0052	2
2	F2	Default series	3	#ffbe00	3
3	F1	Default series	4	#728f02	4
4	D	Default series	5	#443988	5
5	C	Default series	6	#ff3f20	6
6	B	Default series	7	#5DA629	7

Mapping the Stack to Surfaces

[29]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('F1'),
                          'Fault2': ('F2'),
                          'Strata1': ('H', 'G'),
                          'Strata2': ('D', 'C', 'B'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_is_fault(['Fault1', 'Fault2'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[29]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault1	1	Fault	True	True	False
Fault2	2	Fault	True	True	False
Strata1	3	Erosion	True	False	False
Strata2	4	Erosion	True	False	False

Adding additional Orientations

[30]:
```
geo_model.add_orientations(X=1200, Y=1350, Z=1025, surface='G', orientation=[90,5,1])
geo_model.add_orientations(X=3500, Y=350, Z=1000, surface='G', orientation=[90,6,1])
geo_model.add_orientations(X=3500, Y=350, Z=1000, surface='H', orientation=[90,6,1])
geo_model.add_orientations(X=3500, Y=2000, Z=1000, surface='G', orientation=[90,6,1])
geo_model.add_orientations(X=3500, Y=2000, Z=1000, surface='H', orientation=[90,6,1])
geo_model.add_orientations(X=2800, Y=2000, Z=1000, surface='D', orientation=[90,18,1])
geo_model.add_orientations(X=2800, Y=1000, Z=1000, surface='D', orientation=[90,18,1])
```

[30]:

	X	Y	Z	G_x	G_y	G_z	smooth	surface
0	1216.18	1369.03	1000.00	1.00	-0.03	0.00	0.01	F1
1	3018.79	1368.82	1000.00	0.99	0.13	0.00	0.01	F2
10	3500.00	350.00	1000.00	0.10	0.00	0.99	0.01	H
12	3500.00	2000.00	1000.00	0.10	0.00	0.99	0.01	H
7	2481.98	1233.26	750.00	0.10	-0.00	0.99	0.01	G
8	1200.00	1350.00	1025.00	0.09	0.00	1.00	0.01	G
9	3500.00	350.00	1000.00	0.10	0.00	0.99	0.01	G
11	3500.00	2000.00	1000.00	0.10	0.00	0.99	0.01	G
13	2800.00	2000.00	1000.00	0.31	0.00	0.95	0.01	D
14	2800.00	1000.00	1000.00	0.31	0.00	0.95	0.01	D
2	1845.27	1413.90	650.00	0.28	0.07	0.96	0.01	C
3	1524.85	1332.63	750.00	0.29	0.07	0.95	0.01	C
4	627.09	2284.99	550.00	0.29	0.06	0.95	0.01	C
5	2029.39	637.61	650.00	0.28	0.07	0.96	0.01	C
6	284.87	2301.29	450.00	0.31	0.07	0.95	0.01	B

Showing the Number of Data Points

[31]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[31]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
3	F1	Fault1	1	#527682	1	2	1
2	F2	Fault2	1	#527682	2	2	1
0	H	Strata1	1	#ffbe00	3	55	2
1	G	Strata1	2	#728f02	4	100	4
4	D	Strata2	1	#443988	5	65	2
5	C	Strata2	2	#ff3f20	6	96	4
6	B	Strata2	3	#5DA629	7	79	1
7	Basement	Strata2	4	#4878d0	8	0	0

Loading Digital Elevation Model

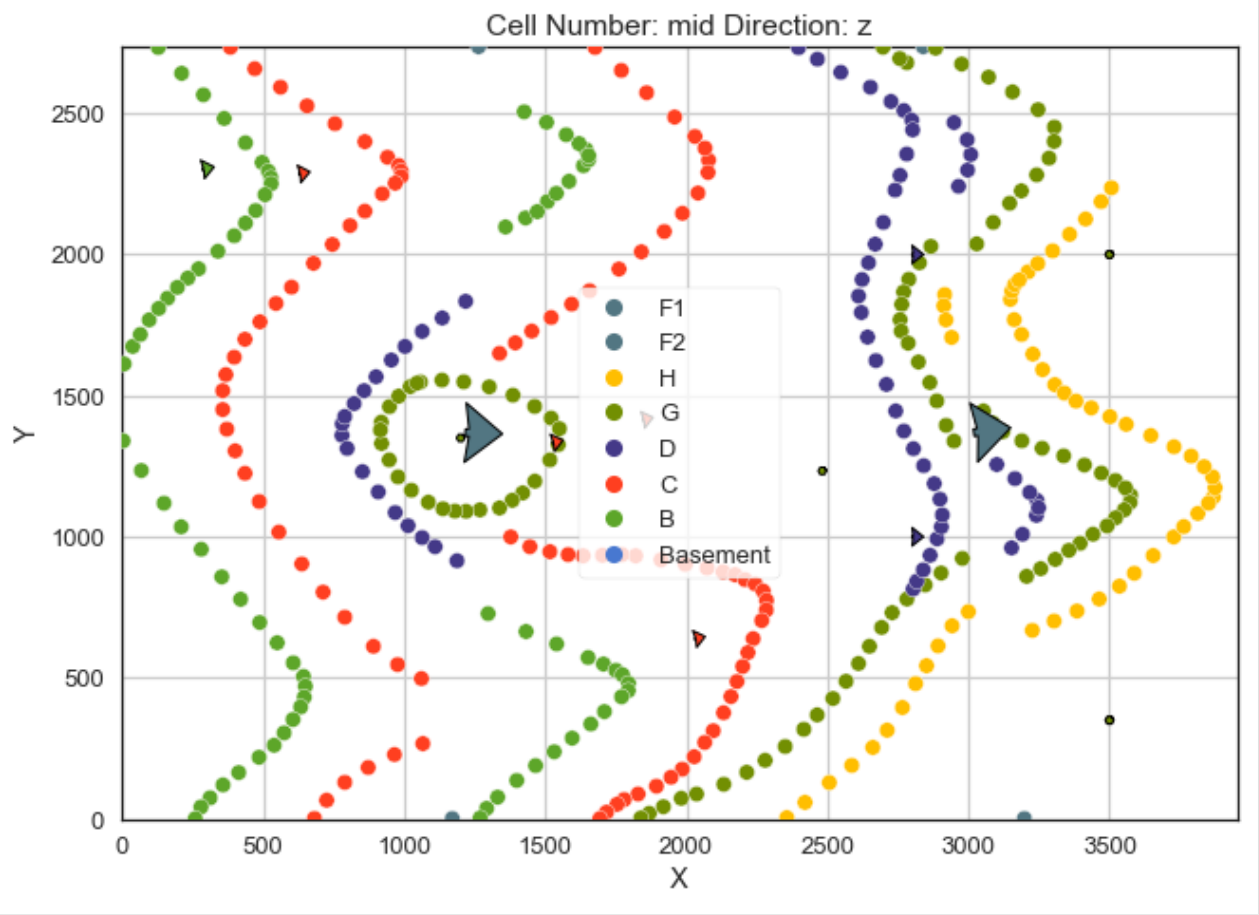
[32]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster10.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

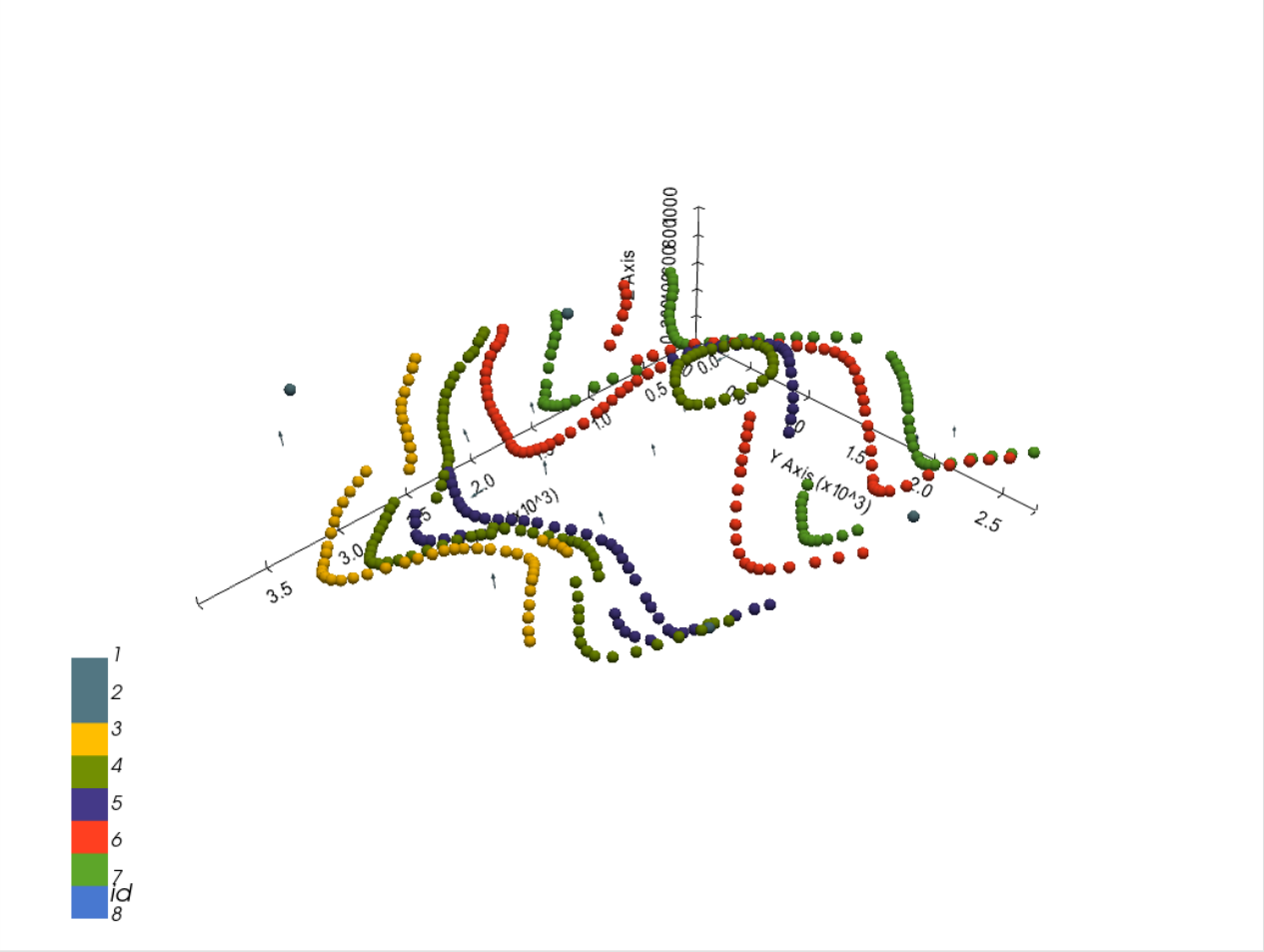
[32]:
```
Grid Object. Values:
array([[  19.77      ,   13.69      ,    5.        ],
       [  19.77      ,   13.69      ,   15.        ],
       [  19.77      ,   13.69      ,   25.        ],
       ...,
       [3948.99493671, 2713.01824818, 1023.34295654],
       [3948.99493671, 2723.01094891, 1024.94226074],
       [3948.99493671, 2733.00364964, 1026.57104492]])
```

Plotting Input Data

[33]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[34]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[34]:
```
<gempy.plot.vista.GemPyToVista at 0x1da0740ba90>
```

Setting the Interpolator

[35]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  2
Compilation Done!
Kriging values:
                        values
range                 4912.31
$C_o$                574541.9
drift equations  [3, 3, 3, 3]
```

[35]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1da06704df0>
```

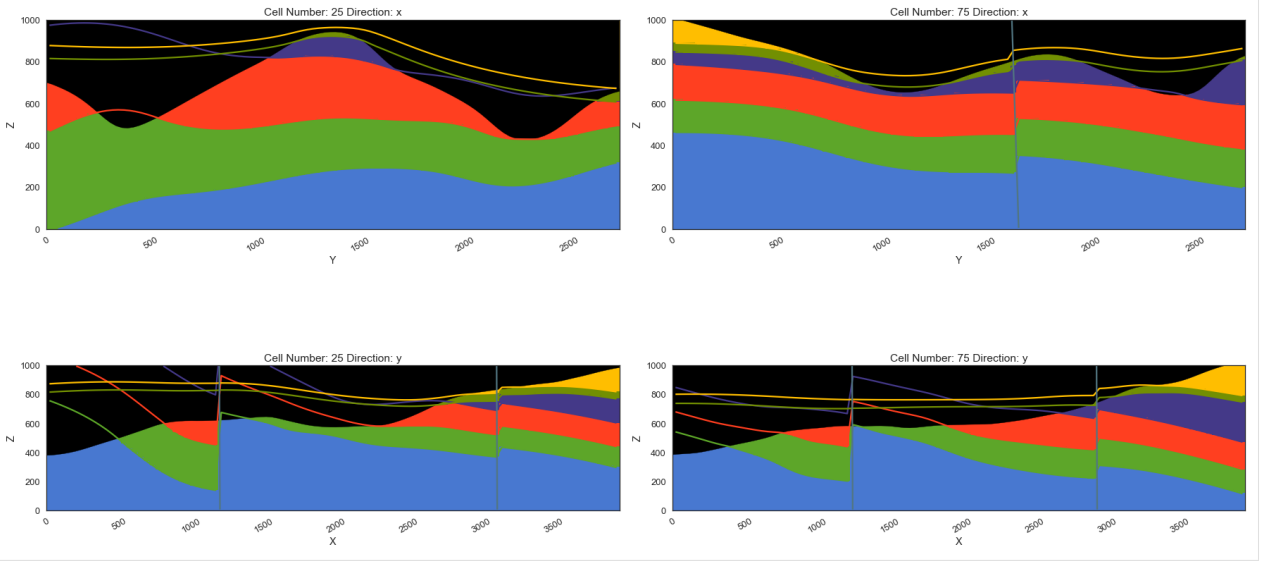
Computing Model

[36]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

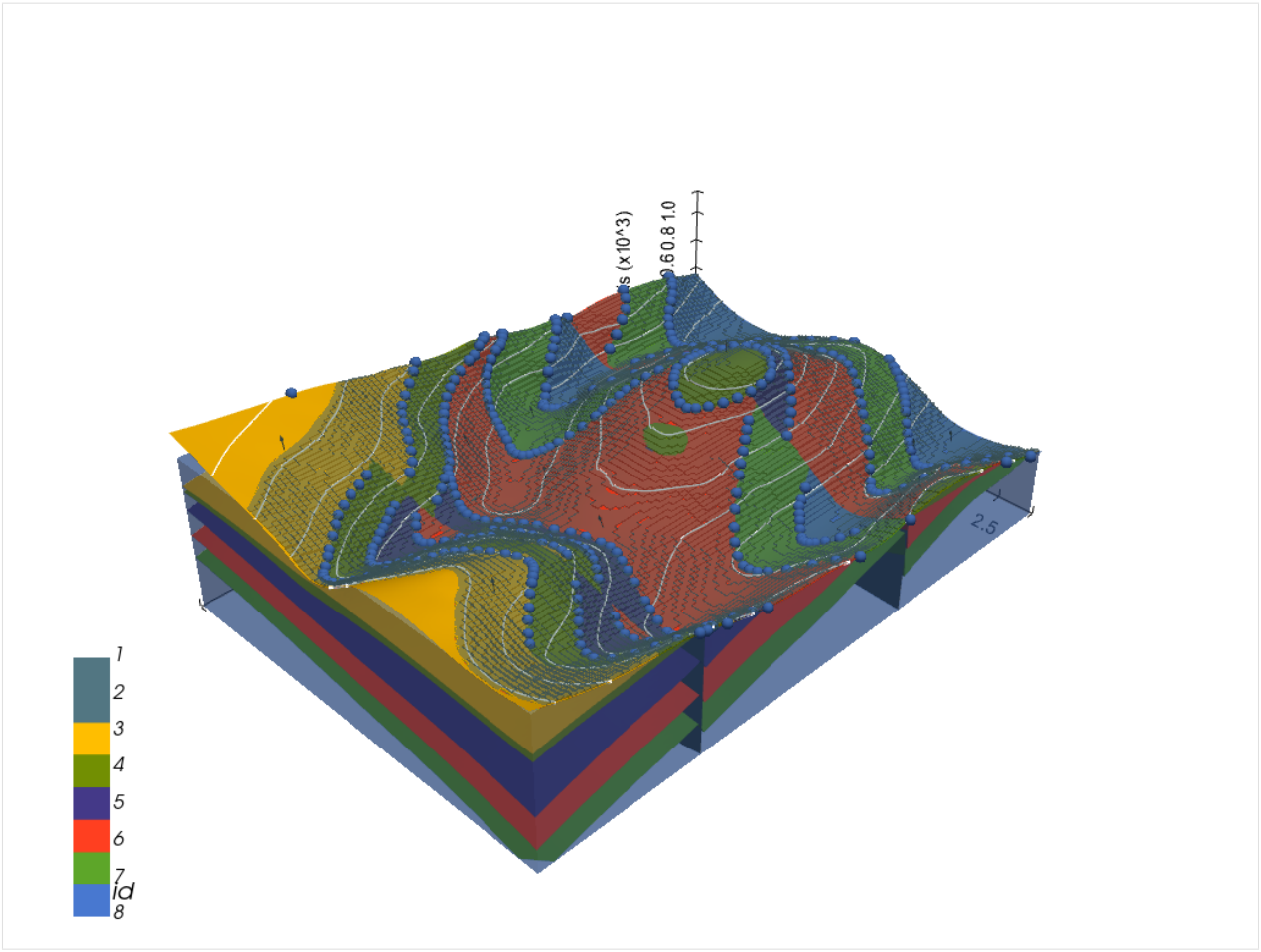
[37]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[37]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1da09edd640>
```



Plotting 3D Model

[38]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



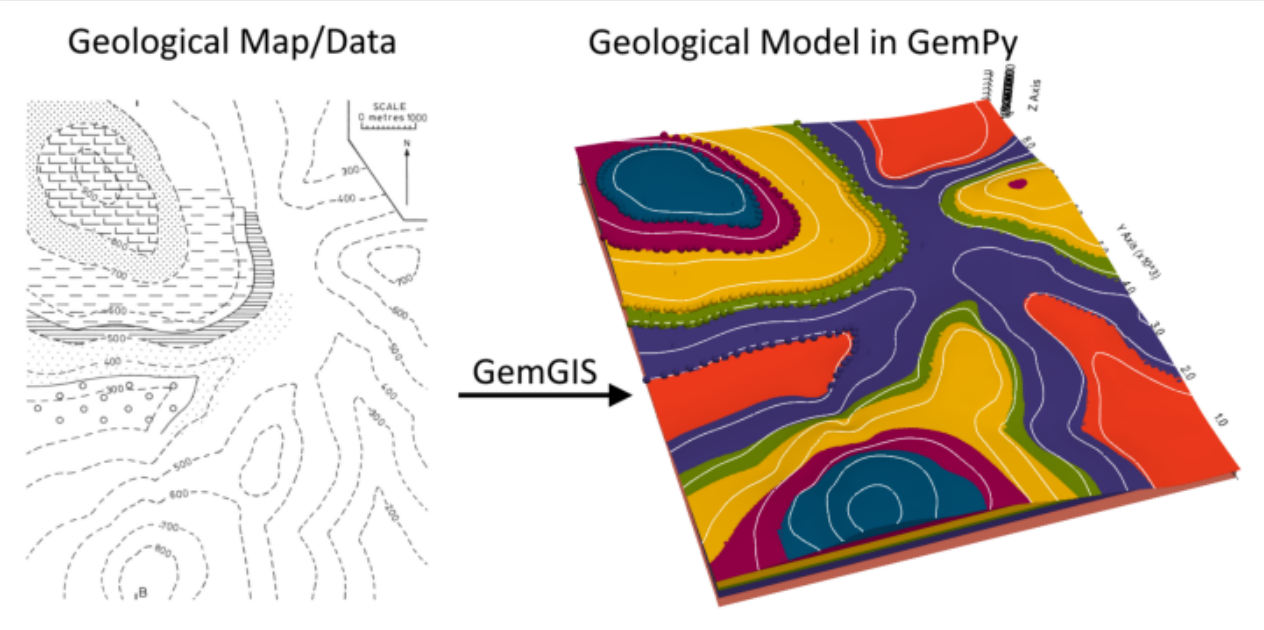
[ ]:

7.11 Example 11 - Horizontal Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 7364 m wide (W-E extent) and 9176 m high (N-S extent). The vertical model extent varies between 0 m and 1000 m. The model represents horizontally deposited layers (blue to purple) above a basement (red). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example11.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.11.1 Licensing

7.11.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.11.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

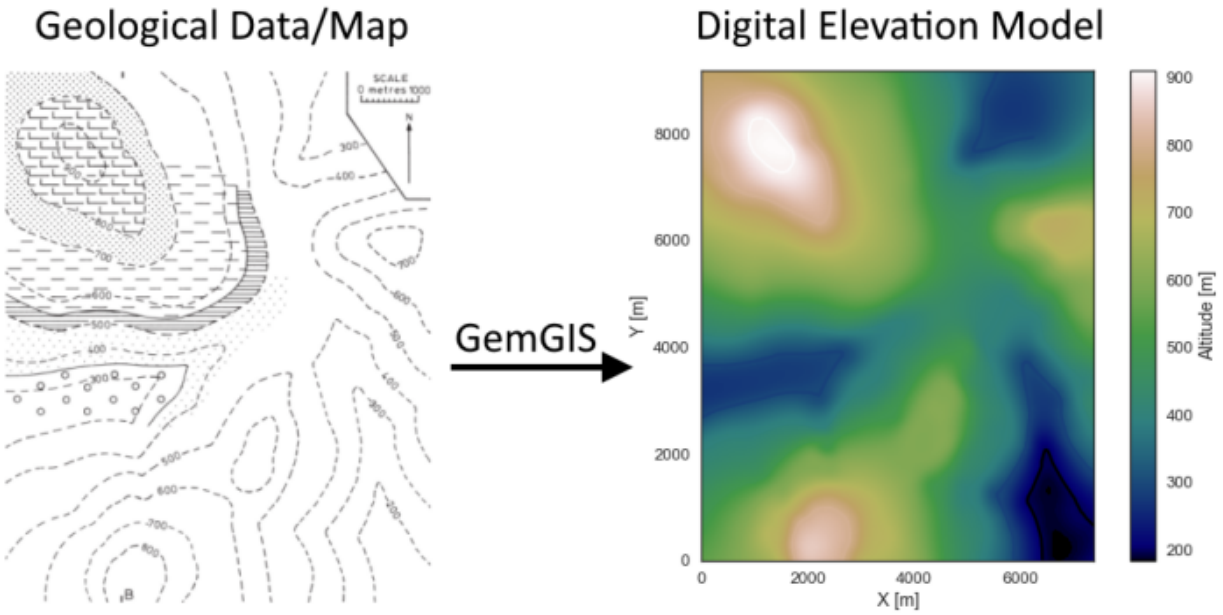
[4]:
```
file_path = 'data/example11/'
gg.download_gemgis_data.download_tutorial_data(filename="example11_horizontal_layers.zip
↪", dirpath=file_path)
```

```
Downloading file 'example11_horizontal_layers.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample11_horizontal_layers.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\example\data\example11'.
```

7.11.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example11.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo11.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	300	LINESTRING (5554.550 9169.979, 5469.438 8954.3...
1	None	400	LINESTRING (4595.623 9164.305, 4696.339 8940.1...
2	None	700	LINESTRING (6411.342 6429.379, 6524.824 6460.5...
3	None	600	LINESTRING (7360.338 6813.800, 7170.255 6803.8...
4	None	500	LINESTRING (6784.415 7146.446, 6510.639 7045.7...

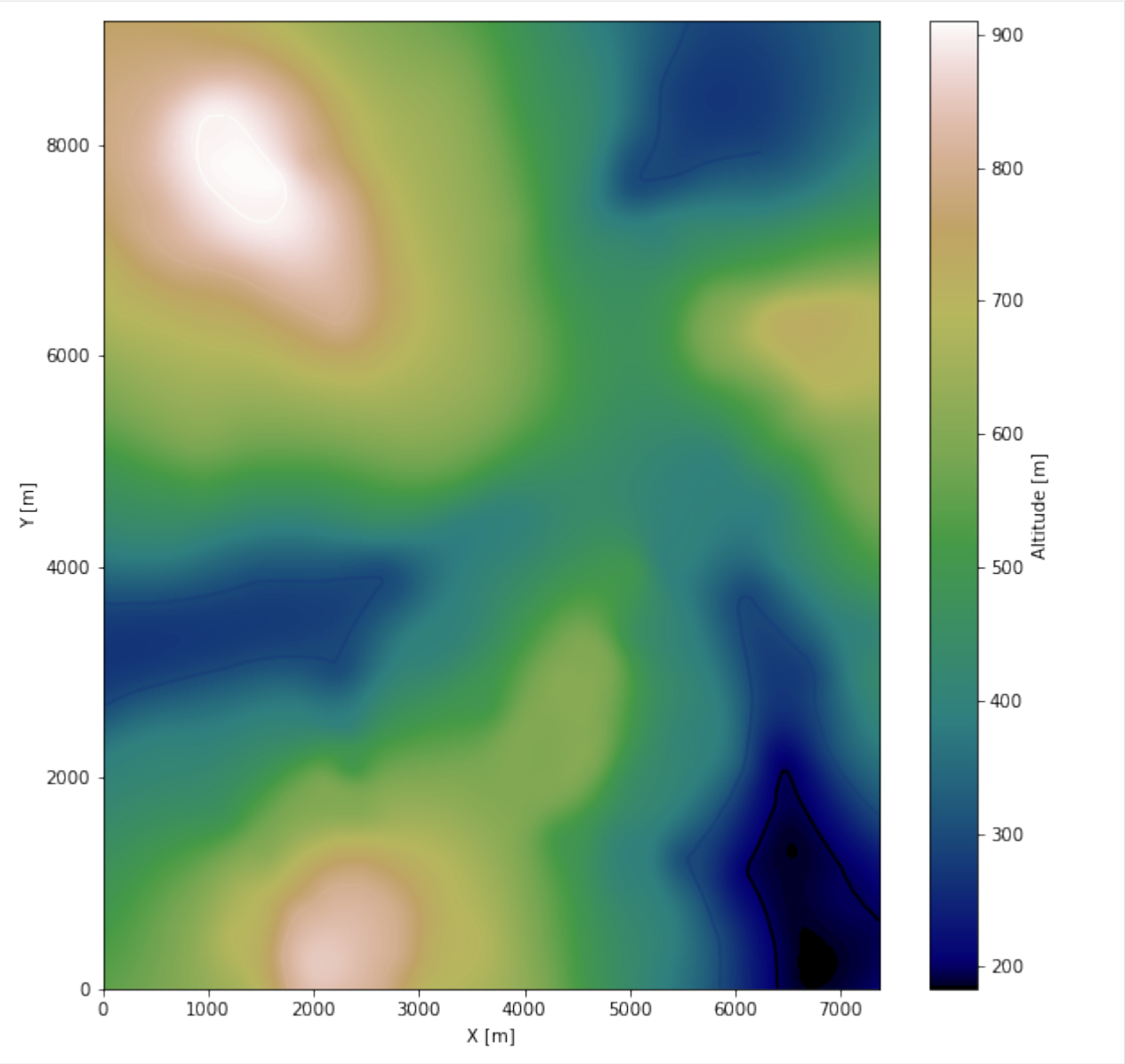
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=15)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 7364, 0, 9176], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 7364)
ax.set_ylim(0, 9176)
```

[8]:
```
(0.0, 9176.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster11.tif', extent = [0, 7364, 0, 9176], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

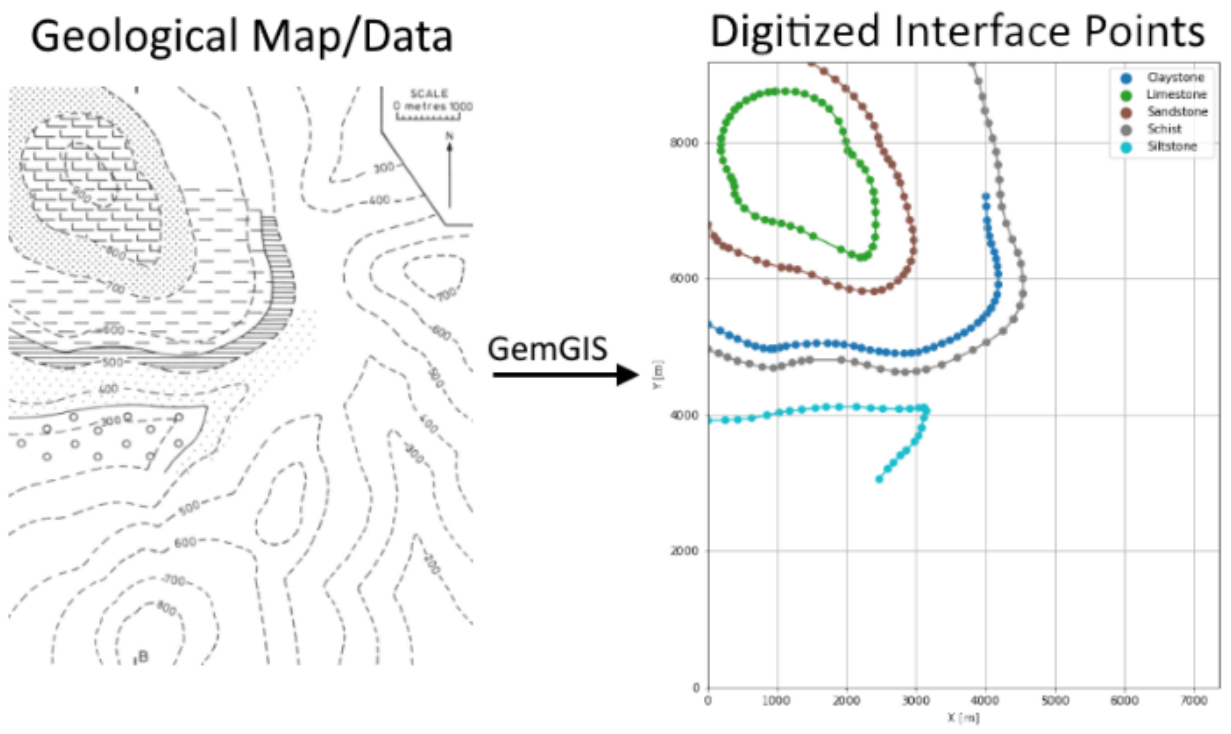
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster11.tif')
```

7.11.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example11.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces11.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Limestone	LINESTRING (993.266 8745.838, 1122.353 8748.67...
1	None	Sandstone	LINESTRING (1496.845 9169.979, 1676.998 9043.7...
2	None	Claystone	LINESTRING (4006.933 7206.733, 4012.608 7058.4...

(continues on next page)

(continued from previous page)

```
3  None     Schist  LINESTRING (3809.049 9166.432, 3899.834 8898.3...
4  None  Siltstone  LINESTRING (8.807 3915.744, 258.468 3921.418, ...
```

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Limestone',
↪'Sandstone', 'Claystone', 'Schist', 'Siltstone'])]
interfaces_coords
```

[12]:

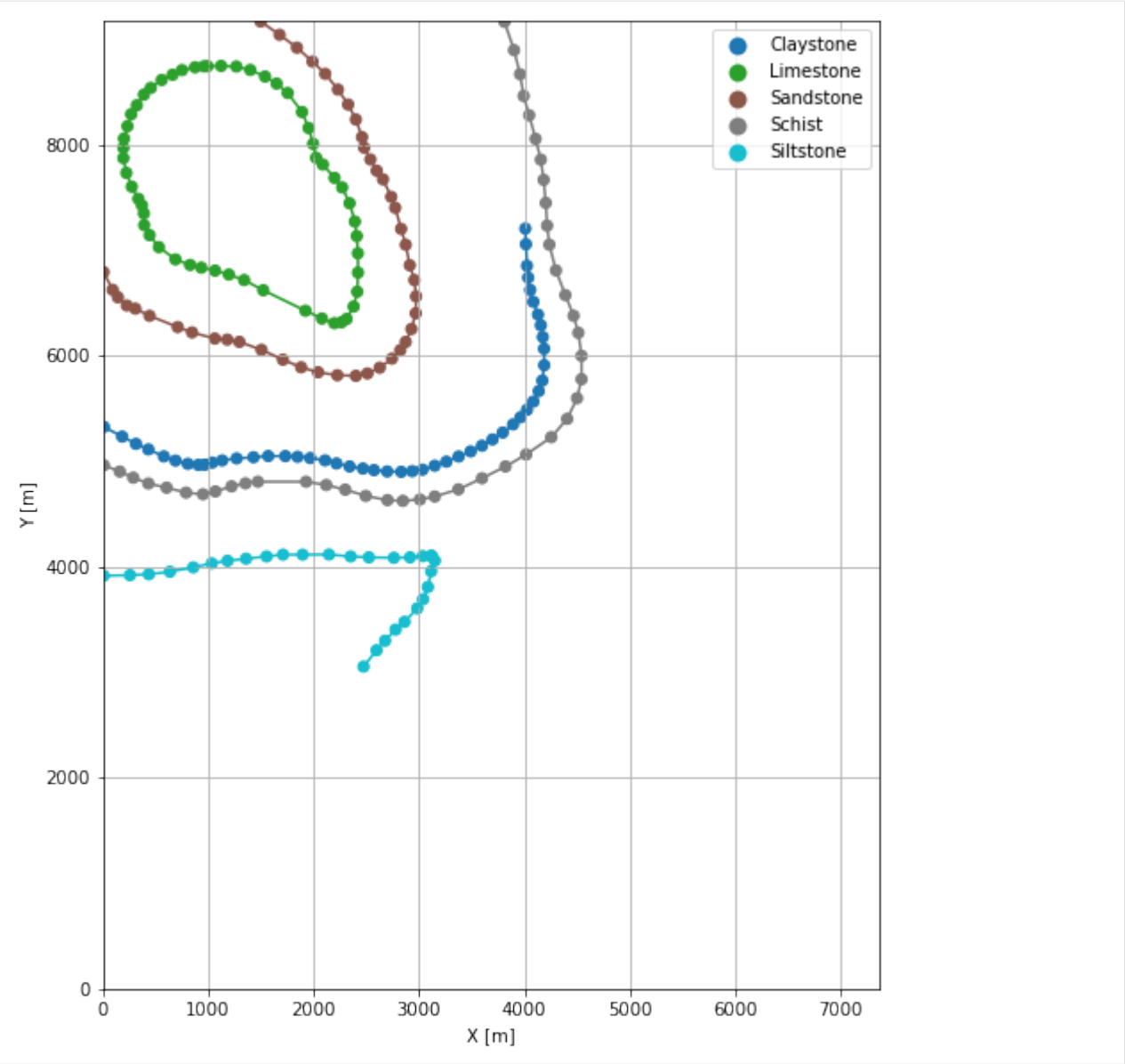
	formation	geometry	X	Y	Z
0	Limestone	POINT (993.266 8745.838)	993.27	8745.84	800.72
1	Limestone	POINT (1122.353 8748.675)	1122.35	8748.68	801.43
2	Limestone	POINT (1267.043 8743.001)	1267.04	8743.00	800.10
3	Limestone	POINT (1398.966 8710.375)	1398.97	8710.37	802.97
4	Limestone	POINT (1539.401 8647.960)	1539.40	8647.96	802.86
..	...	...	...	...	...
215	Siltstone	POINT (2861.471 3478.836)	2861.47	3478.84	364.48
216	Siltstone	POINT (2776.359 3405.073)	2776.36	3405.07	363.79
217	Siltstone	POINT (2677.062 3300.101)	2677.06	3300.10	357.72
218	Siltstone	POINT (2596.205 3209.316)	2596.21	3209.32	351.91
219	Siltstone	POINT (2472.793 3054.696)	2472.79	3054.70	338.67

[220 rows x 5 columns]

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 7364)
ax.set_ylim(0, 9176)
```

[13]:
```
(0.0, 9176.0)
```

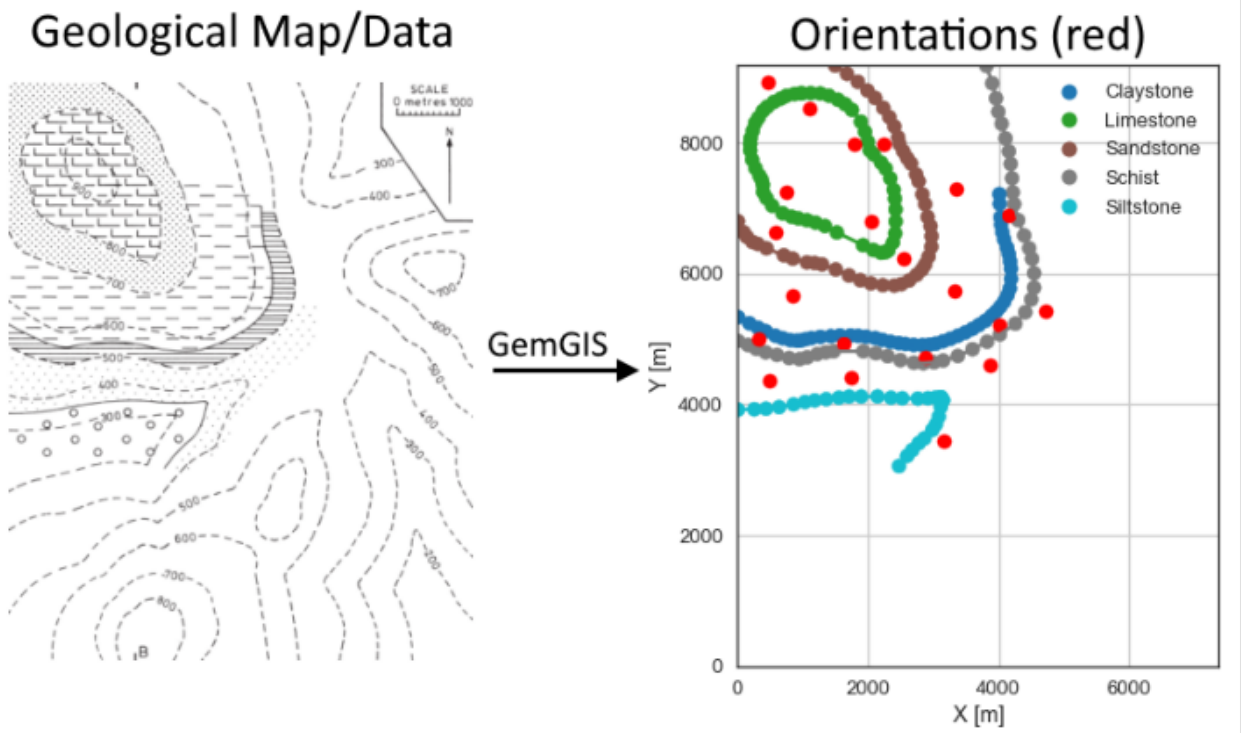


7.11.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

For this example, the orientations were defined by points in QGIS as the layers were deposited horizontally and have no dip.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example11.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



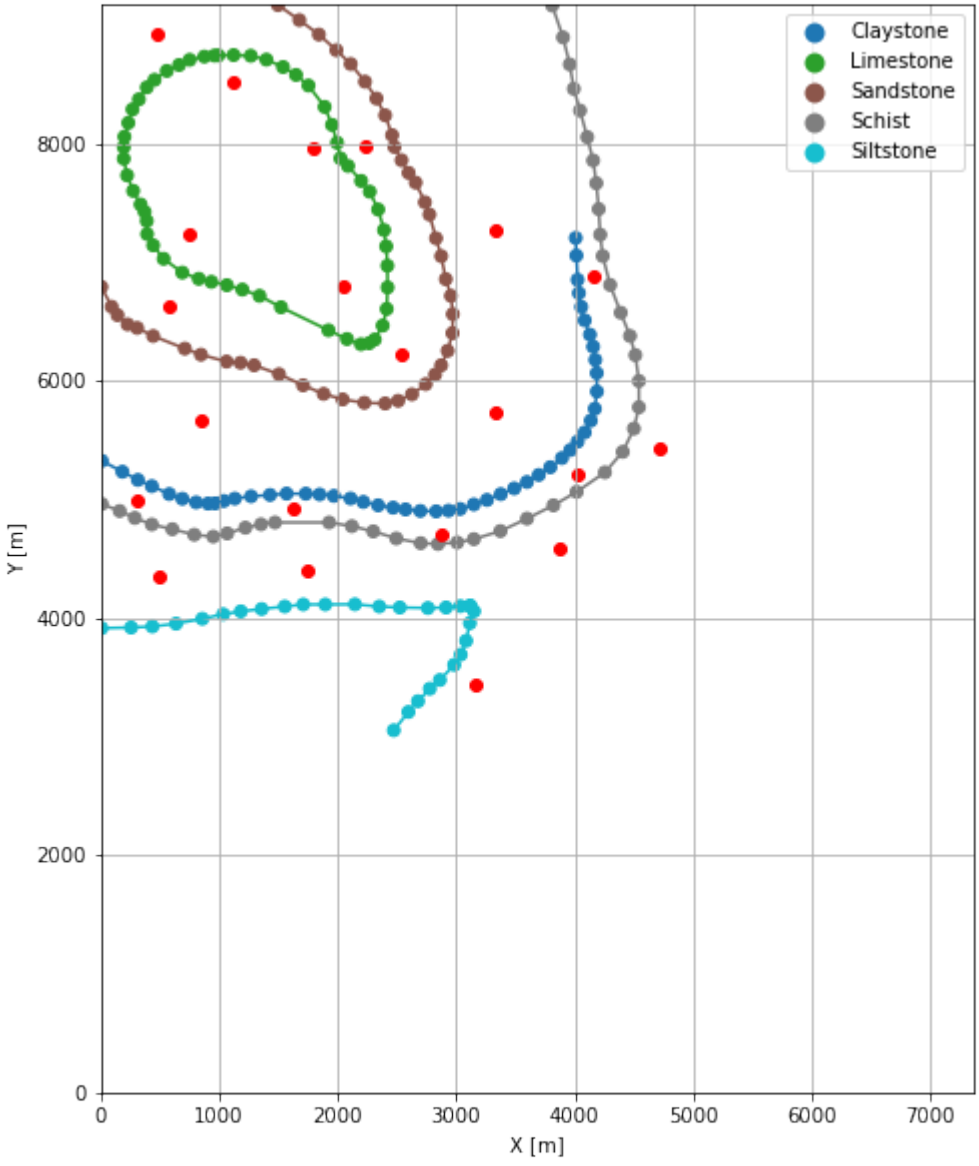
[15]:
```
orientations = gpd.read_file(file_path + 'orientations11.shp')
orientations = gg.vector.extract_xyz(gdf=orientations, dem=topo_raster)
orientations.head()
```

[15]:
```
   dip  azimuth  polarity  formation                  geometry       X  \
0 0.00     0.00      1.00  Limestone  POINT (1111.004 8516.036) 1111.00
1 0.00     0.00      1.00  Limestone  POINT (2041.560 6791.104) 2041.56
2 0.00     0.00      1.00  Limestone   POINT (747.861 7245.033)  747.86
3 0.00     0.00      1.00  Limestone  POINT (1791.899 7971.321) 1791.90
4 0.00     0.00      1.00  Sandstone  POINT (2540.883 6223.692) 2540.88

        Y      Z
0 8516.04 867.24
1 6791.10 842.33
2 7245.03 837.07
3 7971.32 848.08
4 6223.69 755.14
```

Plotting the Orientations

[16]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlim(0,7364)
plt.ylim(0,9176)
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
```

[16]:
```
Text(123.55887265135695, 0.5, 'Y [m]')
```



7.11.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[17]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[18]:
```
geo_model = gp.create_model('Model11')
geo_model
```

[18]:
```
Model11  2022-04-05 11:02
```

Initiate Data

[19]:
```
gp.init_data(geo_model, [0, 7364, 0, 9176, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[19]:
```
Model11  2022-04-05 11:02
```

Model Surfaces

[20]:
```
geo_model.surfaces
```

[20]:

	surface	series	order_surfaces	color	id
0	Limestone	Default series	1	#015482	1
1	Sandstone	Default series	2	#9f0052	2
2	Claystone	Default series	3	#ffbe00	3
3	Schist	Default series	4	#728f02	4
4	Siltstone	Default series	5	#443988	5

Mapping the Stack to Surfaces

[21]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Limestone', 'Sandstone', 'Claystone', 'Schist',
↪'Siltstone'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[21]:

	surface	series	order_surfaces	color	id
0	Limestone	Strata1	1	#015482	1
1	Sandstone	Strata1	2	#9f0052	2
2	Claystone	Strata1	3	#ffbe00	3
3	Schist	Strata1	4	#728f02	4
4	Siltstone	Strata1	5	#443988	5
5	Basement	Strata1	6	#ff3f20	6

Showing the Number of Data Points

[22]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[22]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Limestone	Strata1	1	#015482	1	54	4
1	Sandstone	Strata1	2	#9f0052	2	44	4
2	Claystone	Strata1	3	#ffbe00	3	50	3
3	Schist	Strata1	4	#728f02	4	43	5
4	Siltstone	Strata1	5	#443988	5	29	5
5	Basement	Strata1	6	#ff3f20	6	0	0

Loading Digital Elevation Model

[23]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster11.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[23]:
```
Grid Object. Values:
array([[3.68200000e+01, 4.58800000e+01, 5.00000000e+00],
       [3.68200000e+01, 4.58800000e+01, 1.50000000e+01],
       [3.68200000e+01, 4.58800000e+01, 2.50000000e+01],
       ...,
```

(continues on next page)

(continued from previous page)

```
       [7.35650102e+03, 9.13851634e+03, 3.66282013e+02],
       [7.35650102e+03, 9.15350980e+03, 3.66405060e+02],
       [7.35650102e+03, 9.16850327e+03, 3.66539032e+02]])
```

Defining Custom Section

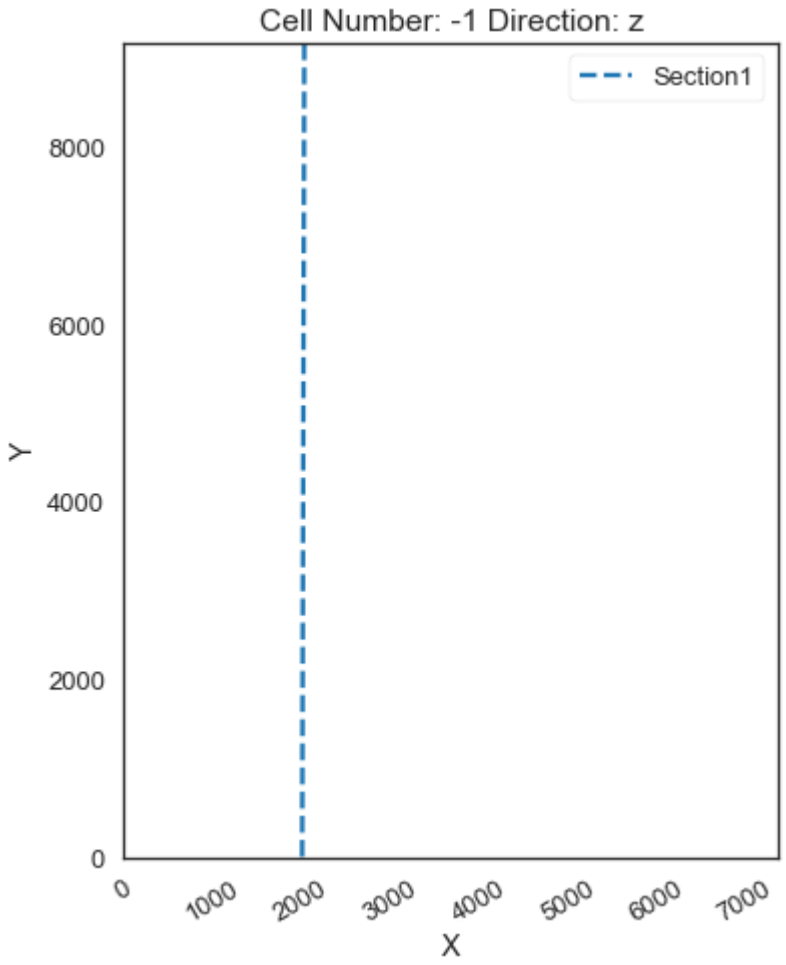
[24]:
```
custom_section = gpd.read_file(file_path + 'customsection11.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='name')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[24]:
```
                                          start                                 ␣
↪stop resolution    dist
Section1  [2033.048983708095, 9169.978630835749]  [2008.9339736694492, 4.
↪856286148714844]  [100, 80] 9165.15
```

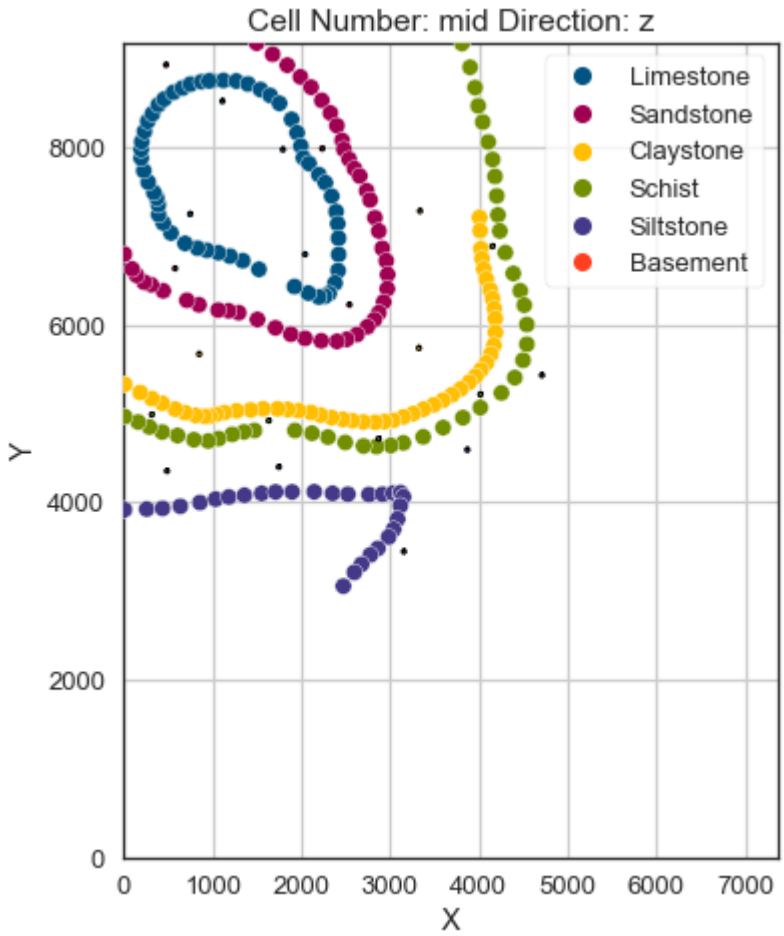
[25]:
```
gp.plot.plot_section_traces(geo_model)
```

[25]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1b5ae7f52e0>
```

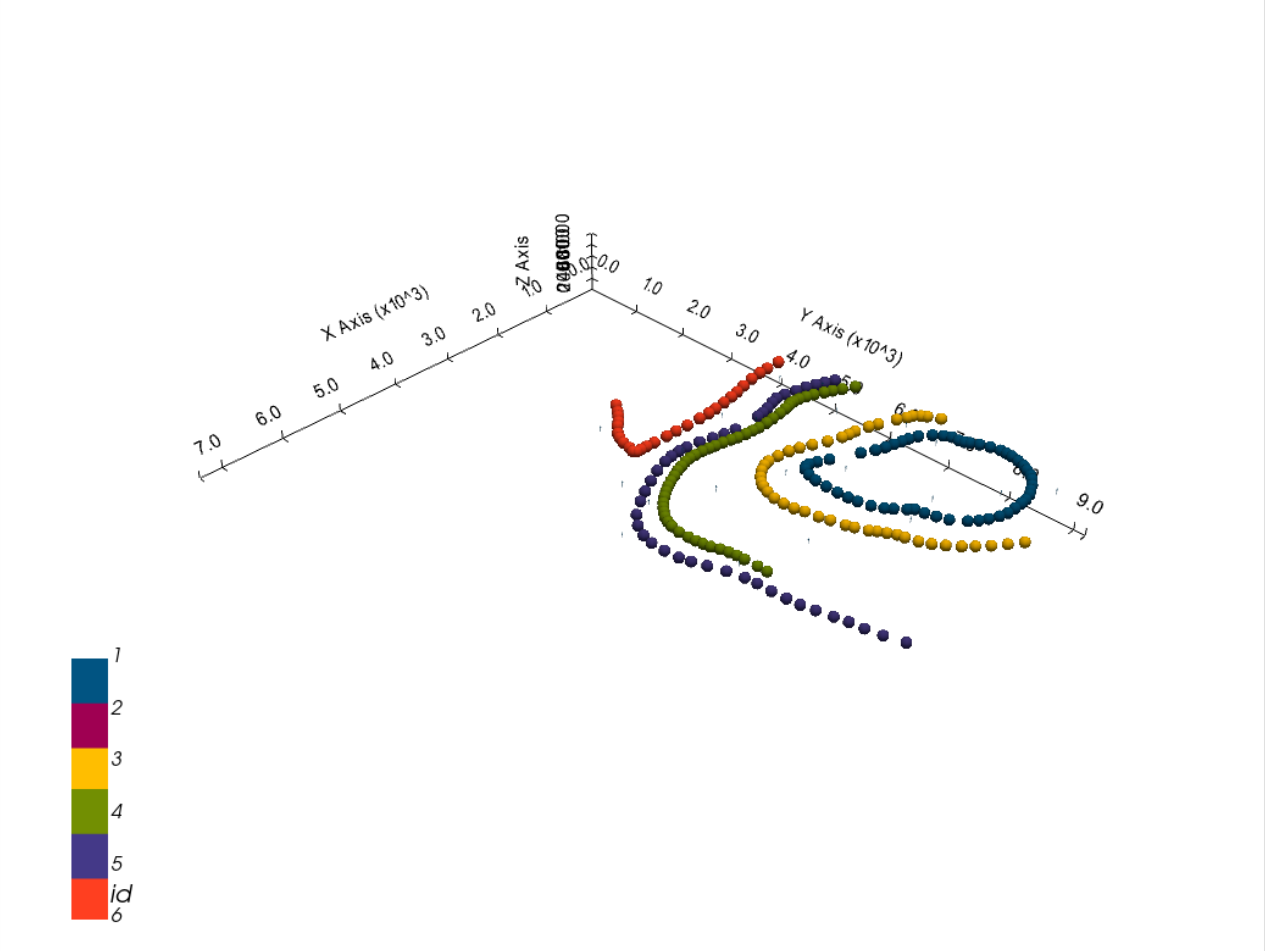


Plotting Input Data

[26]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[27]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[27]:
```
<gempy.plot.vista.GemPyToVista at 0x1b5b036d3d0>
```

Setting the Interpolator

[28]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                     values
range             11807.94
$C_o$           3319701.71
drift equations        [3]
```

[28]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1b5ae7f5d90>
```

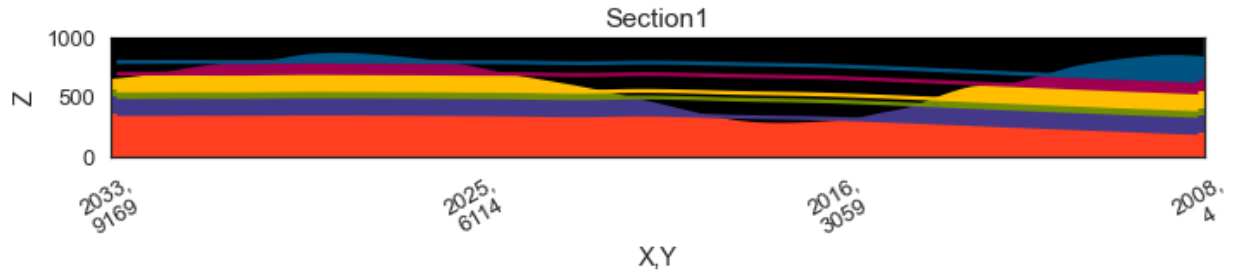
Computing Model

[29]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

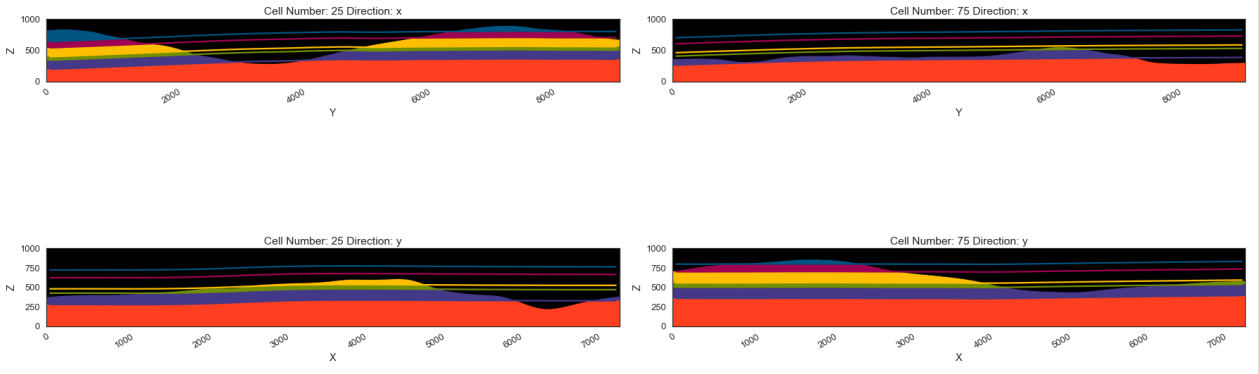
[30]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[30]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1b5b16055b0>
```



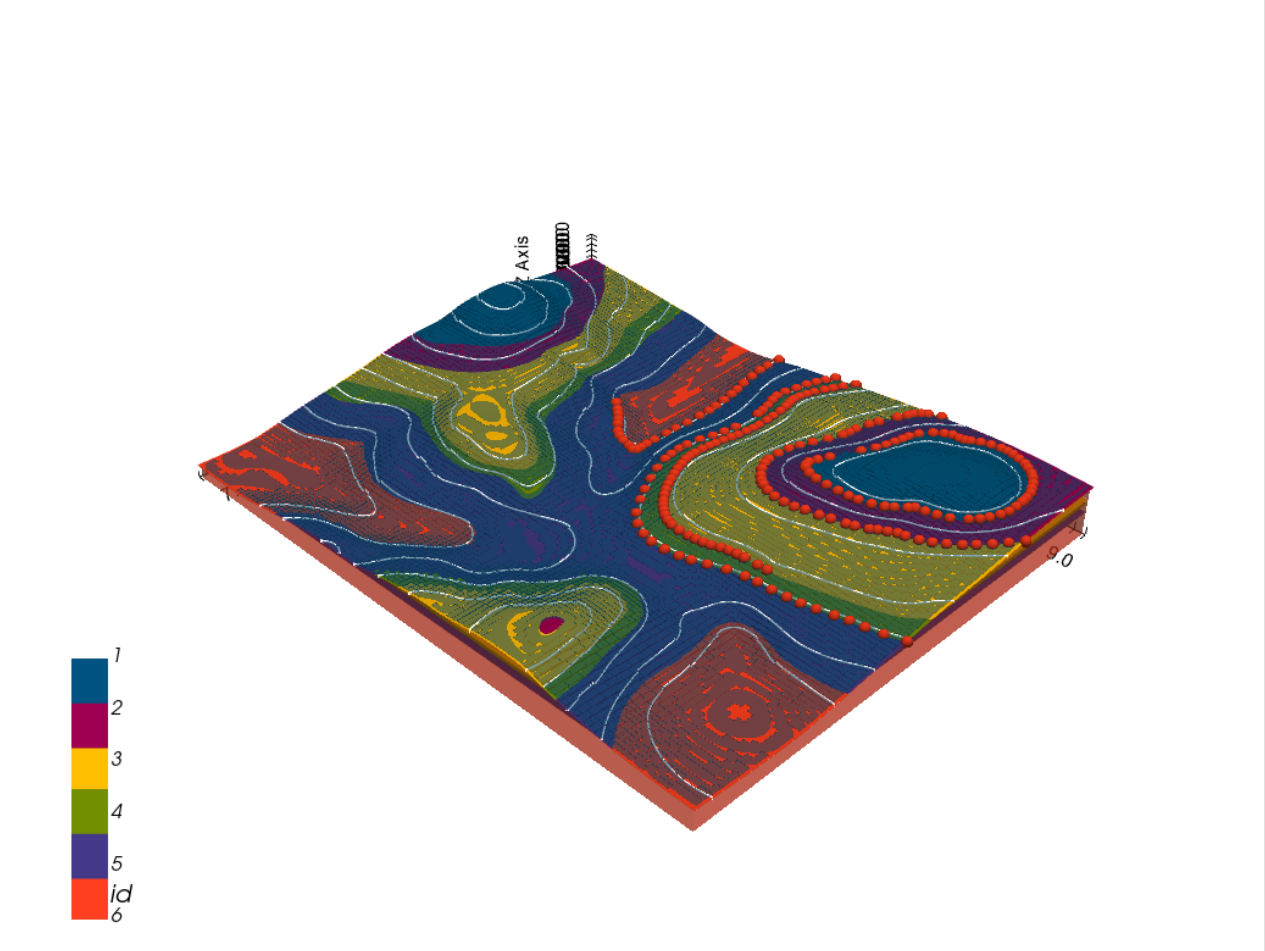
[31]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[31]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1b5b1a325b0>
```



Plotting 3D Model

[32]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

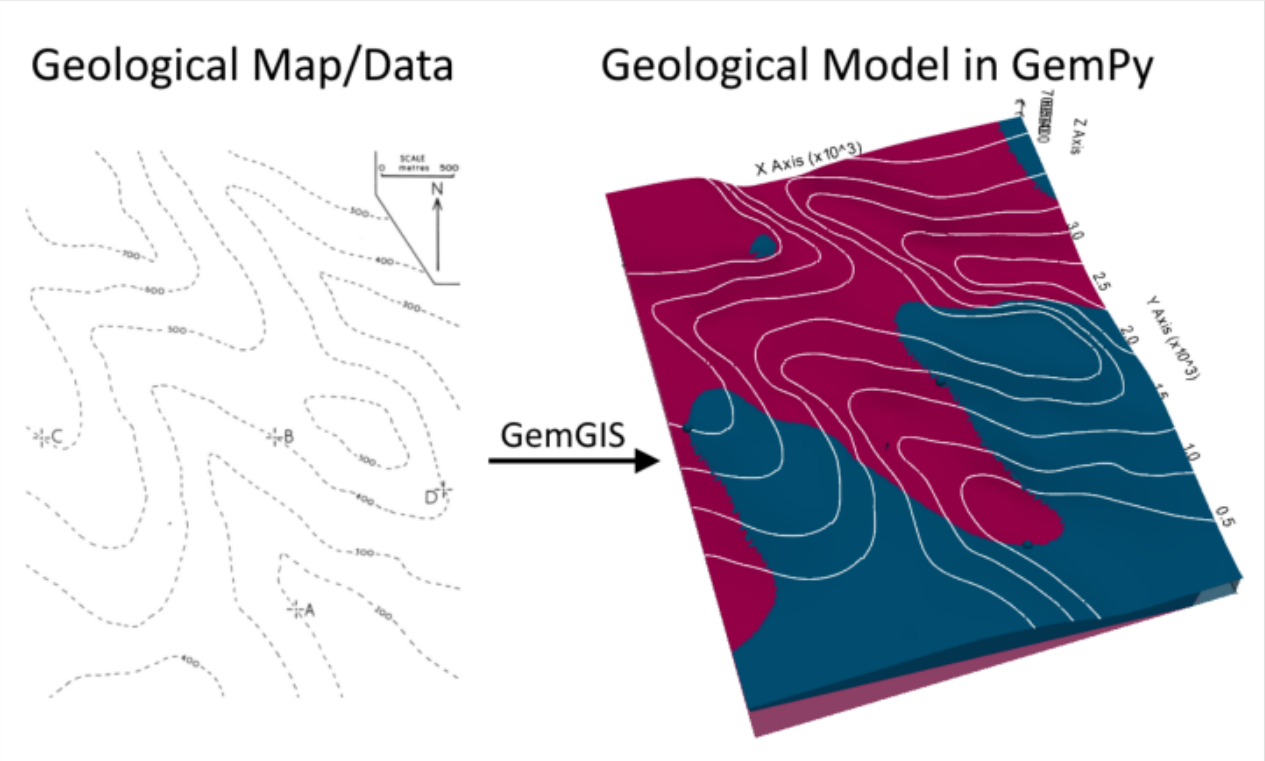
7.12 Example 12 - Three Point Problem

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 2964 m wide (W-E extent) and 3725 m high (N-S extent). the vertical model extent varies between 0 m and 1000 m. This example represents a classic "three-point-problem" of a planar dipping layer (blue) above a basement (purple).

The map has been georeferenced with QGIS. The outcrops of the layers were digitized in QGIS. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example12.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.12.1 Licensing

7.12.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.12.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using pooch. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

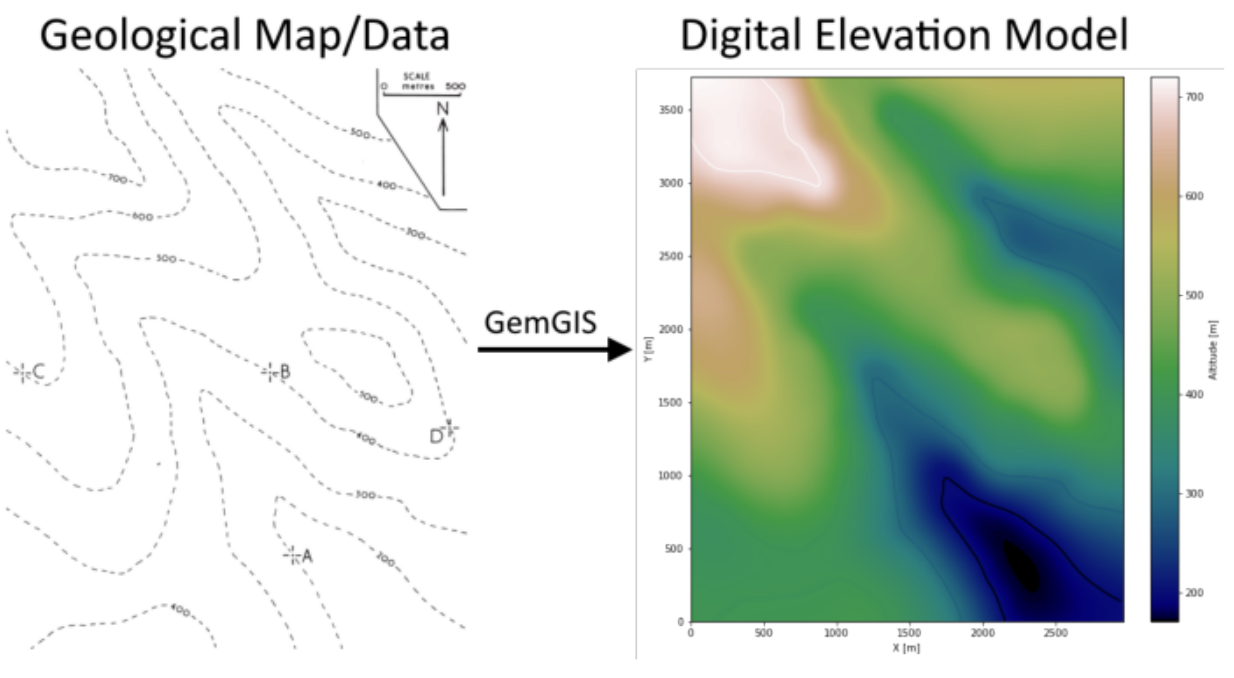
[4]:
```
file_path = 'data/example12/'
gg.download_gemgis_data.download_tutorial_data(filename="example12_three_point_problem.
↪zip", dirpath=file_path)
```

```
Downloading file 'example12_three_point_problem.zip' from 'https://rwth-aachen.sciebo.de/
↪s/AfXRsZywYDbUF34/download?path=%2Fexample12_three_point_problem.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\example\data\example12'.
```

7.12.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the SciPy Radial Basis Function interpolation wrapped in GemGIS. The respective function used for that is gg.vector.interpolate_raster().

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example12.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo12.shp')
topo.head()
```

[6]:
```
     id    Z                                                geometry
0  None  400  LINESTRING (166.181 5.047, 217.108 44.754, 269...
1  None  400  LINESTRING (3.471 901.028, 85.474 844.490, 153...
2  None  300  LINESTRING (1859.740 4.184, 1840.318 64.175, 1...
3  None  200  LINESTRING (2158.400 4.831, 2123.010 99.134, 2...
4  None  500  LINESTRING (1942.173 2049.057, 2007.344 2055.5...
```

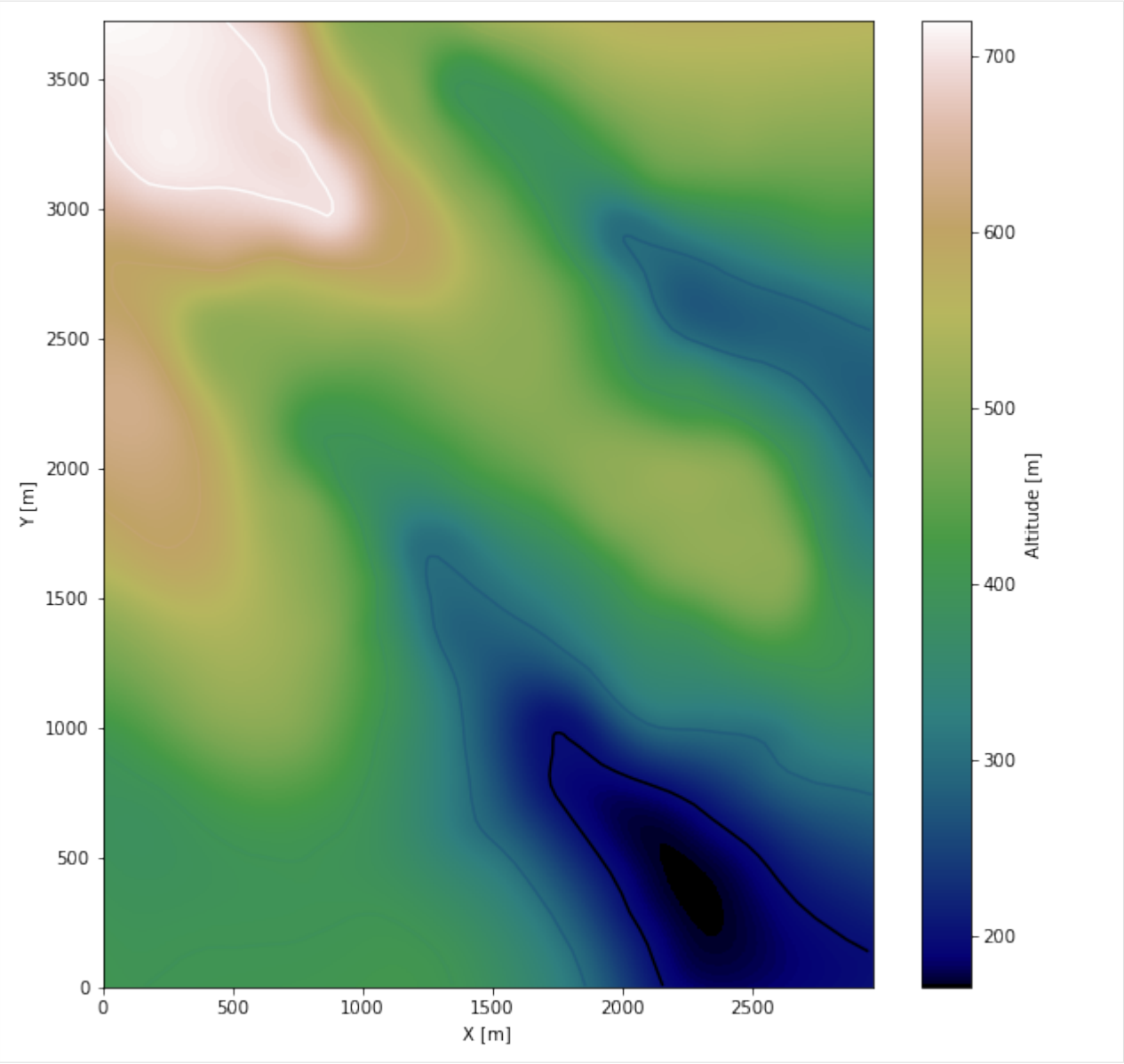
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=15)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 2966, 0, 3725], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2966)
ax.set_ylim(0, 3725)
```

[8]:
```
(0.0, 3725.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster12.tif', extent = [0, 2966, 0, 3725], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

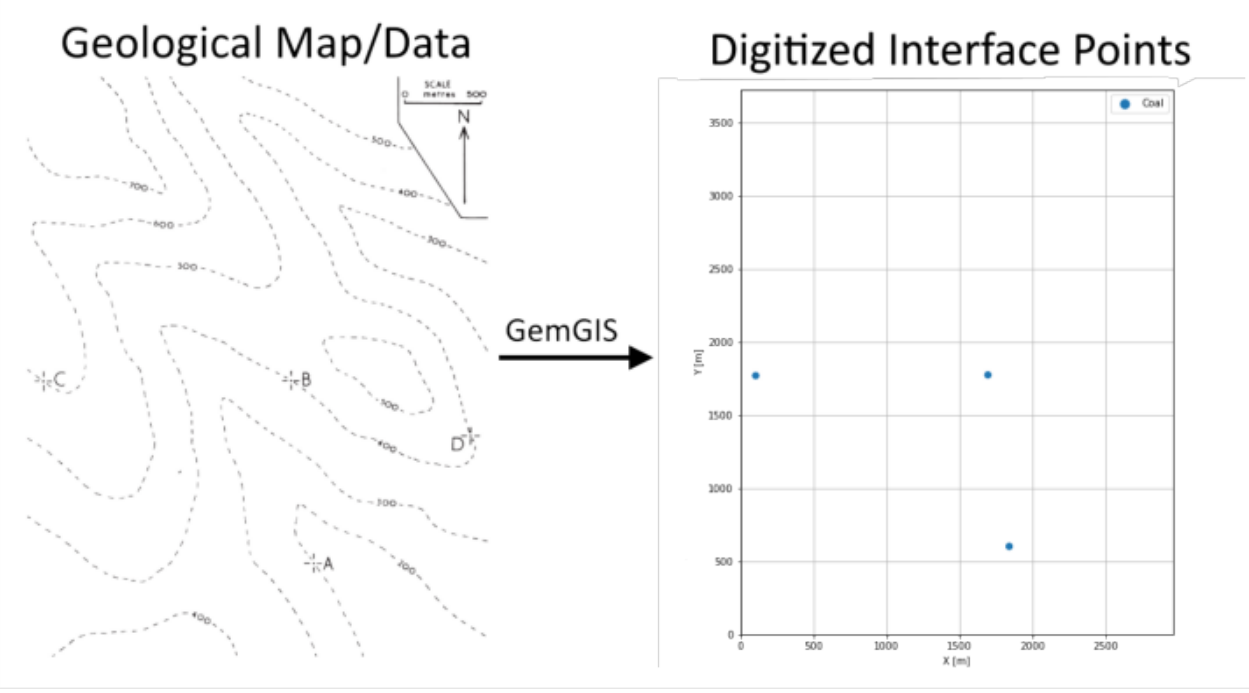
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster12.tif')
```

7.12.5 Interface Points of stratigraphic boundaries

The interface points for this three point example will be digitized as points with the respective height value as given by the contour lines and the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example12.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces12.shp')
interfaces.head()
```

[11]:

	id	formation	Z	geometry
0	None	Coal	600	POINT (104.302 1770.385)
1	None	Coal	400	POINT (1696.262 1775.038)
2	None	Coal	200	POINT (1842.732 602.462)

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=None)
interfaces_coords
```

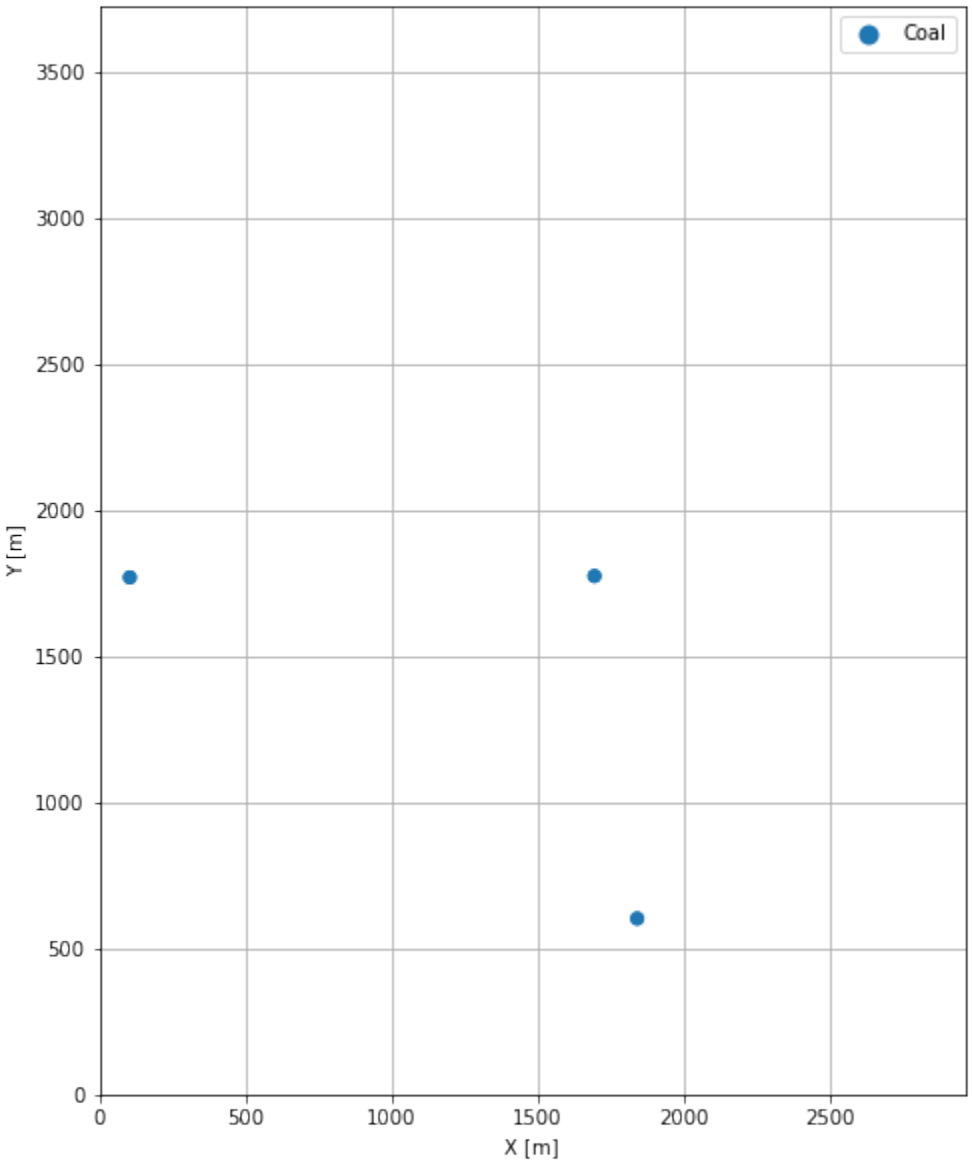
[12]:

	formation	Z	geometry	X	Y
0	Coal	600.00	POINT (104.302 1770.385)	104.30	1770.39
1	Coal	400.00	POINT (1696.262 1775.038)	1696.26	1775.04
2	Coal	200.00	POINT (1842.732 602.462)	1842.73	602.46

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2966)
ax.set_ylim(0, 3725)
```

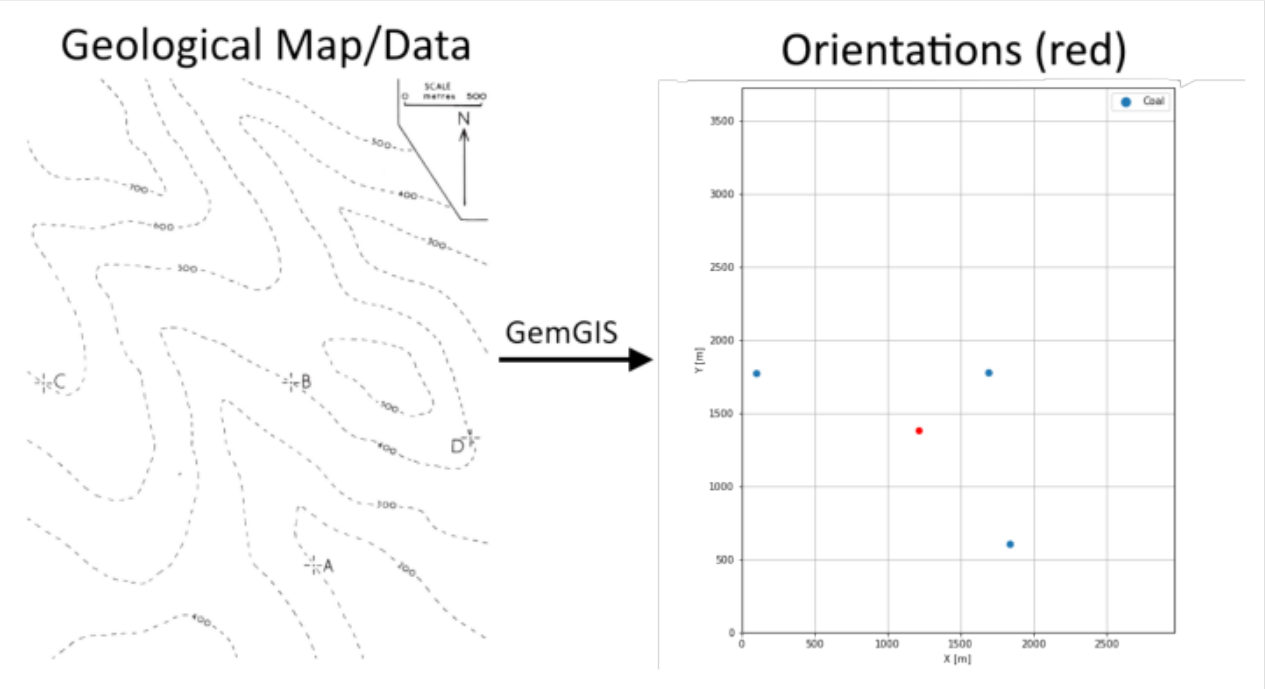
[13]:
```
(0.0, 3725.0)
```



7.12.6 Orientations from Strike Lines

For this three point example, an orientation is calculated using `gg.vector.calculate_orientation_for_three_point_problem()`.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example12.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
orientations = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces)
orientations
```

[15]:
```
      Z formation azimuth   dip polarity       X       Y  \
0 400.0      Coal  140.84 11.29        1 1214.43 1382.63

                     geometry
0  POINT (1214.432 1382.628)
```

Setting the data type of the fields

[16]:
```
orientations['Z'] = orientations['Z'].astype(float)
orientations['azimuth'] = orientations['azimuth'].astype(float)
orientations['dip'] = orientations['dip'].astype(float)
orientations['polarity'] = orientations['polarity'].astype(float)
orientations['X'] = orientations['X'].astype(float)
orientations['Y'] = orientations['Y'].astype(float)
orientations.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Z          1 non-null      float64
 1   formation  1 non-null      object
 2   azimuth    1 non-null      float64
 3   dip        1 non-null      float64
 4   polarity   1 non-null      float64
```

(continues on next page)

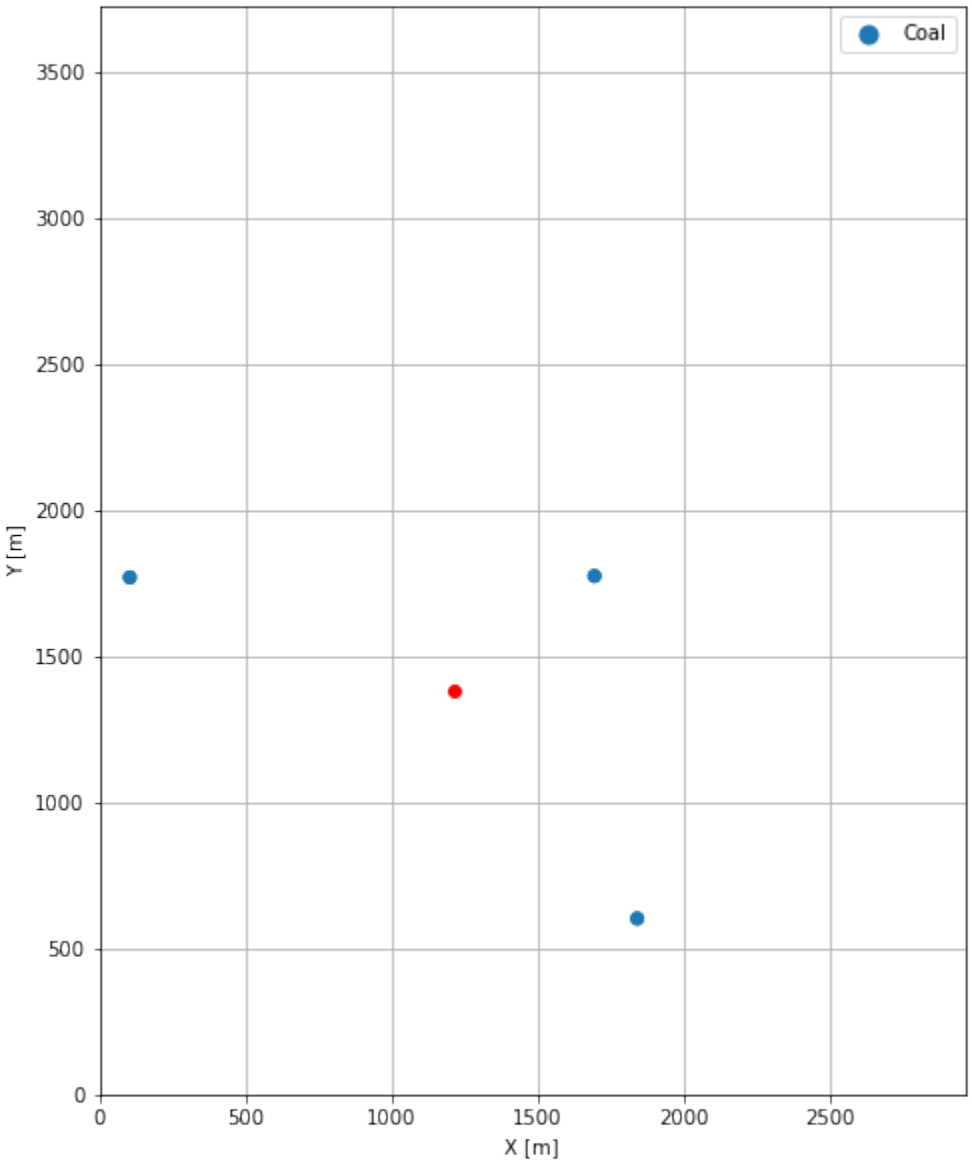
(continued from previous page)

```
 5   X           1 non-null      float64
 6   Y           1 non-null      float64
 7   geometry    1 non-null      geometry
dtypes: float64(6), geometry(1), object(1)
memory usage: 192.0+ bytes
```

Plotting the Orientations

[17]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2966)
ax.set_ylim(0, 3725)
```

[17]:
```
(0.0, 3725.0)
```



7.12.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[18]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[19]:
```
geo_model = gp.create_model('Model12')
geo_model
```

[19]:
```
Model12  2022-04-05 11:03
```

Initiate Data

[20]:
```
gp.init_data(geo_model, [0, 2966, 0, 3725, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[20]:
```
Model12  2022-04-05 11:03
```

Model Surfaces

[21]:
```
geo_model.surfaces
```

[21]:

	surface	series	order_surfaces	color	id
0	Coal	Default series	1	#015482	1

Mapping the Stack to Surfaces

[22]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Coal'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[22]:

	surface	series	order_surfaces	color	id
0	Coal	Strata1	1	#015482	1
1	Basement	Strata1	2	#9f0052	2

Showing the Number of Data Points

[23]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[23]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Coal	Strata1	1	#015482	1	3	1
1	Basement	Strata1	2	#9f0052	2	0	0

Loading Digital Elevation Model

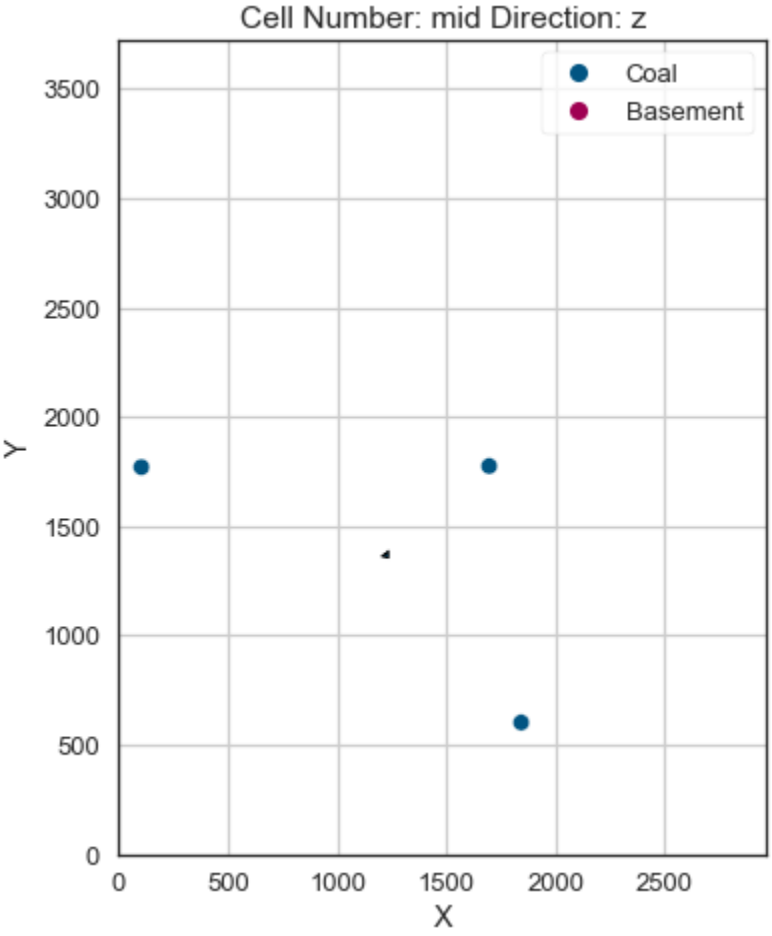
[24]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster12.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

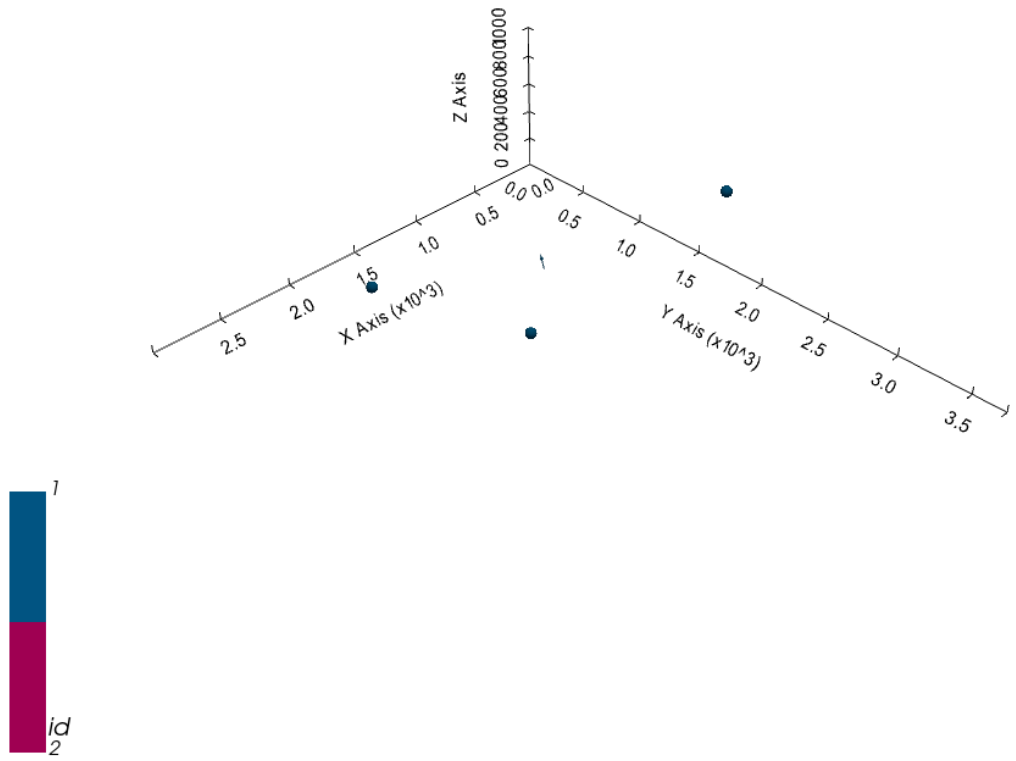
[24]:
```
Grid Object. Values:
array([[  14.83      ,   18.625     ,    5.        ],
       [  14.83      ,   18.625     ,   15.        ],
       [  14.83      ,   18.625     ,   25.        ],
       ...,
       [2958.51010101, 3687.44959677,  556.33007812],
       [2958.51010101, 3702.46975806,  558.58599854],
       [2958.51010101, 3717.48991935,  560.82910156]])
```

Plotting Input Data

[25]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[26]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[26]:
```
<gempy.plot.vista.GemPyToVista at 0x1e22fef8130>
```

Setting the Interpolator

[27]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              4865.47
$C_o$            563637.64
drift equations        [3]
```

[27]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1e228fd5e20>
```

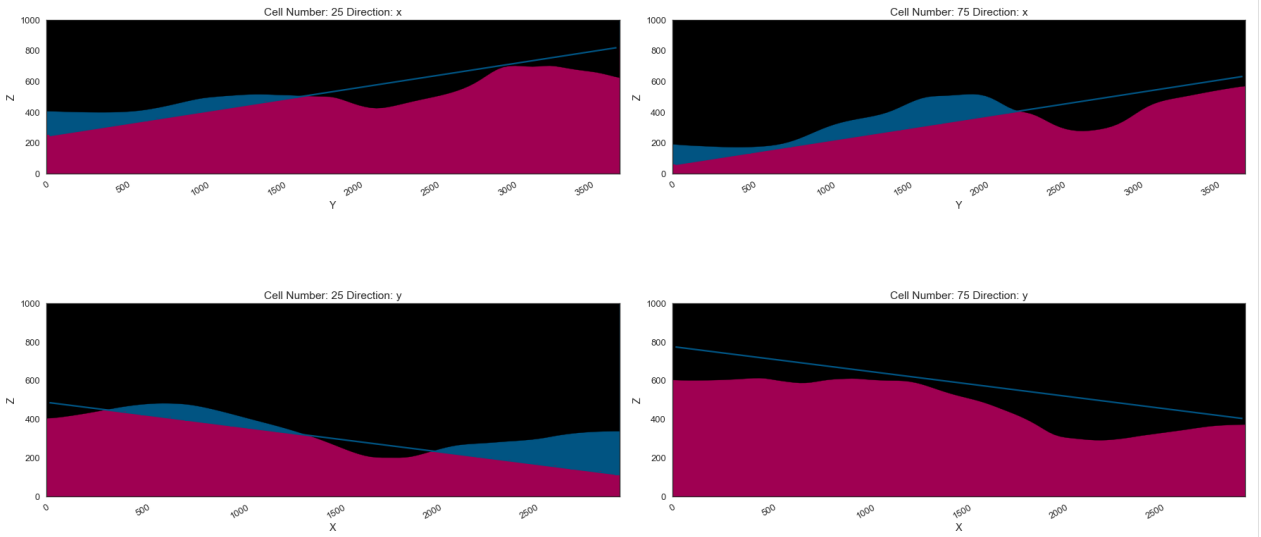
Computing Model

[28]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

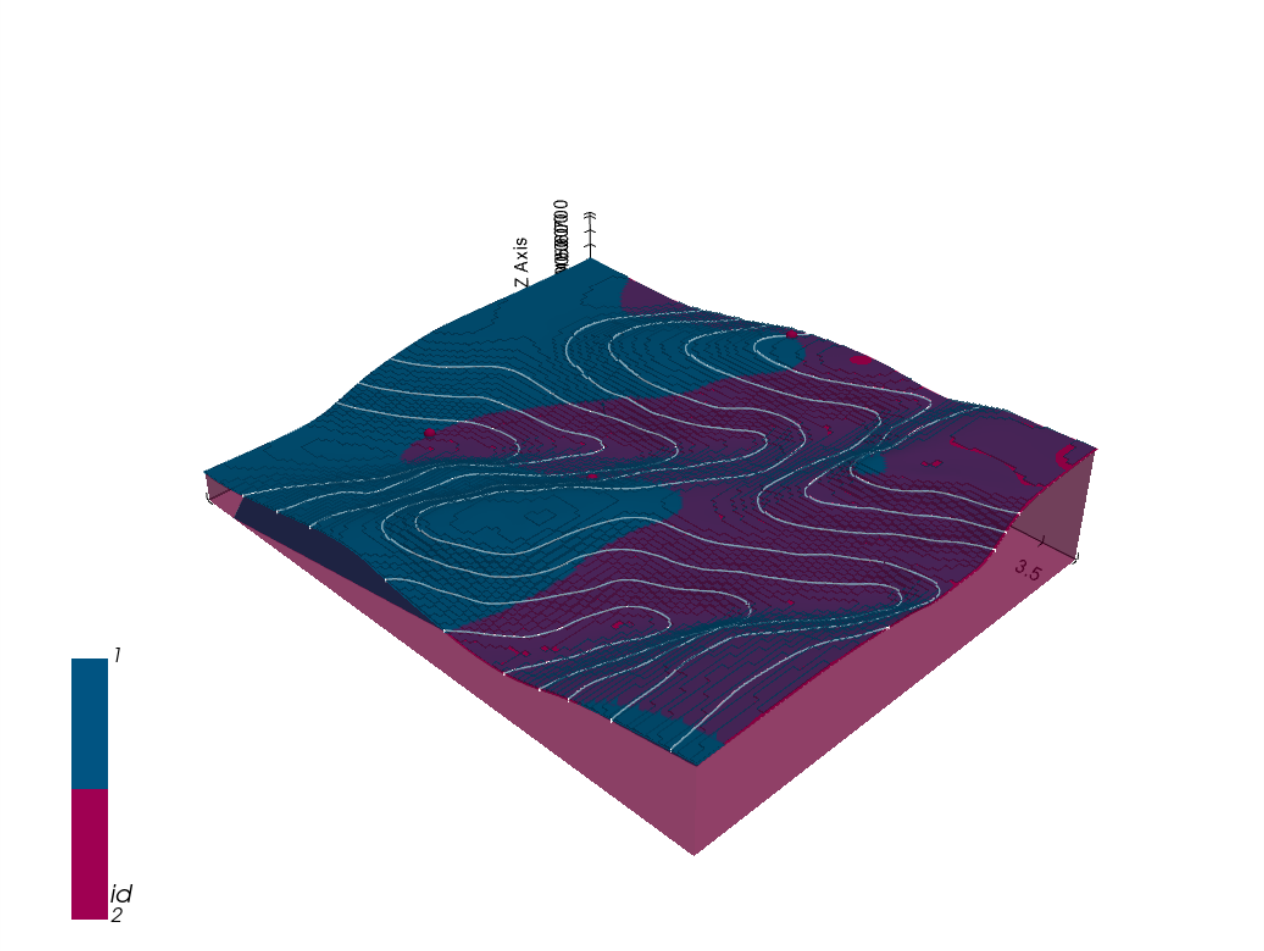
[29]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[29]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1e235edfac0>
```



Plotting 3D Model

[30]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.13 Example 13 - Three Point Problem

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 2991 m wide (W-E extent) and 3736 m high (N-S extent). The vertical model extent varies between 250 m and 1200 m. This example represents a classic "three-point-problem" of planar dippings layer (blue and purple) above an unspecified basement (yellow). The interface points were not taken at the surface but rather in boreholes

The map has been georeferenced with QGIS. The outcrops of the layers were digitized in QGIS. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

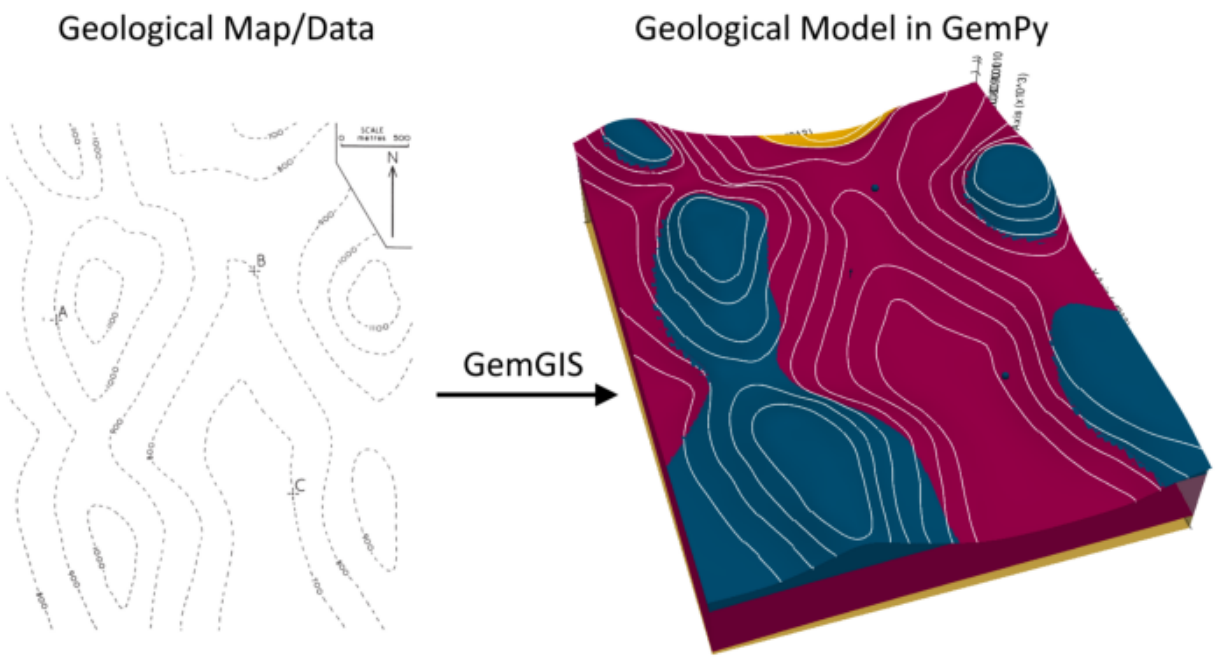
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example13.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.13.1 Licensing

7.13.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.13.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using pooch. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

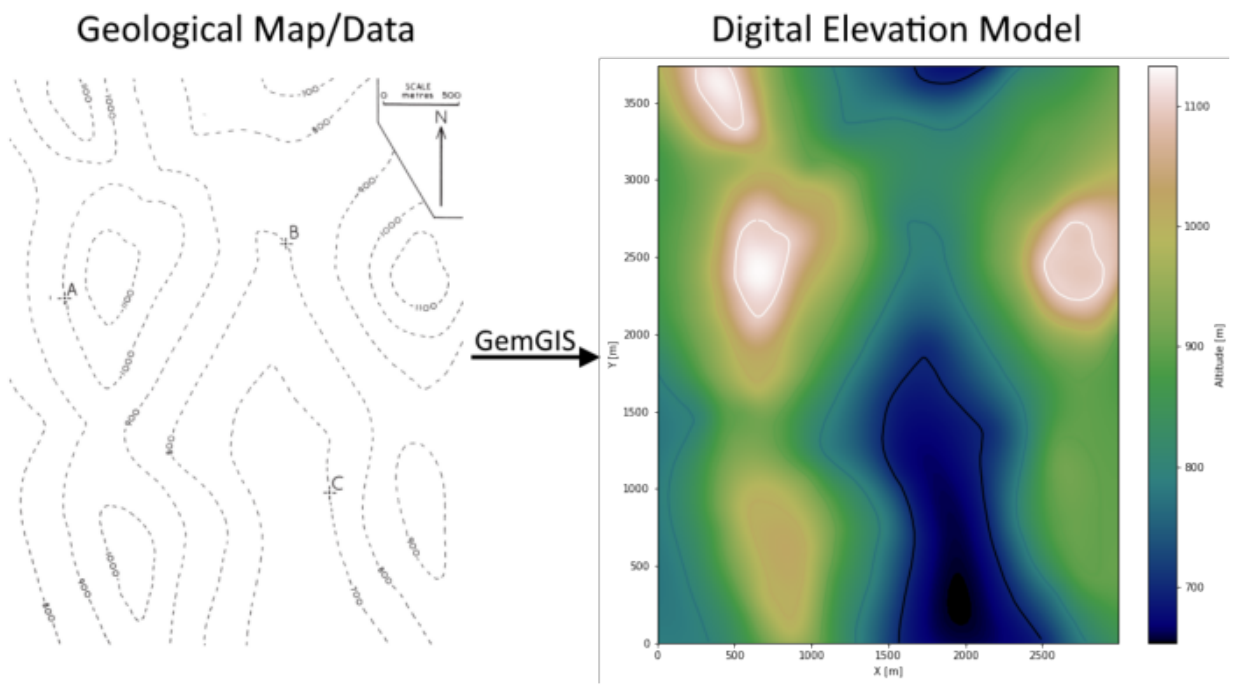
[4]:
```
file_path = 'data/example13/'
gg.download_gemgis_data.download_tutorial_data(filename="example13_three_point_problem.
↪zip", dirpath=file_path)
```

```
Downloading file 'example13_three_point_problem.zip' from 'https://rwth-aachen.sciebo.de/
↪s/AfXRsZywYDbUF34/download?path=%2Fexample13_three_point_problem.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\example\data\example13'.
```

7.13.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the SciPy Radial Basis Function interpolation wrapped in GemGIS. The respective function used for that is gg.vector.interpolate_raster().

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example13.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo13.shp')
topo.head()
```

[6]:
```
     id     Z                                                geometry
0  None   800  LINESTRING (1.482 1748.098, 50.293 1669.250, 9...
1  None   900  LINESTRING (2.060 3333.723, 69.355 3237.834, 1...
2  None  1000  LINESTRING (681.366 917.450, 738.552 907.053, ...
3  None  1000  LINESTRING (36.141 3724.208, 57.225 3659.223, ...
4  None  1100  LINESTRING (249.868 3718.720, 252.467 3636.407...
```

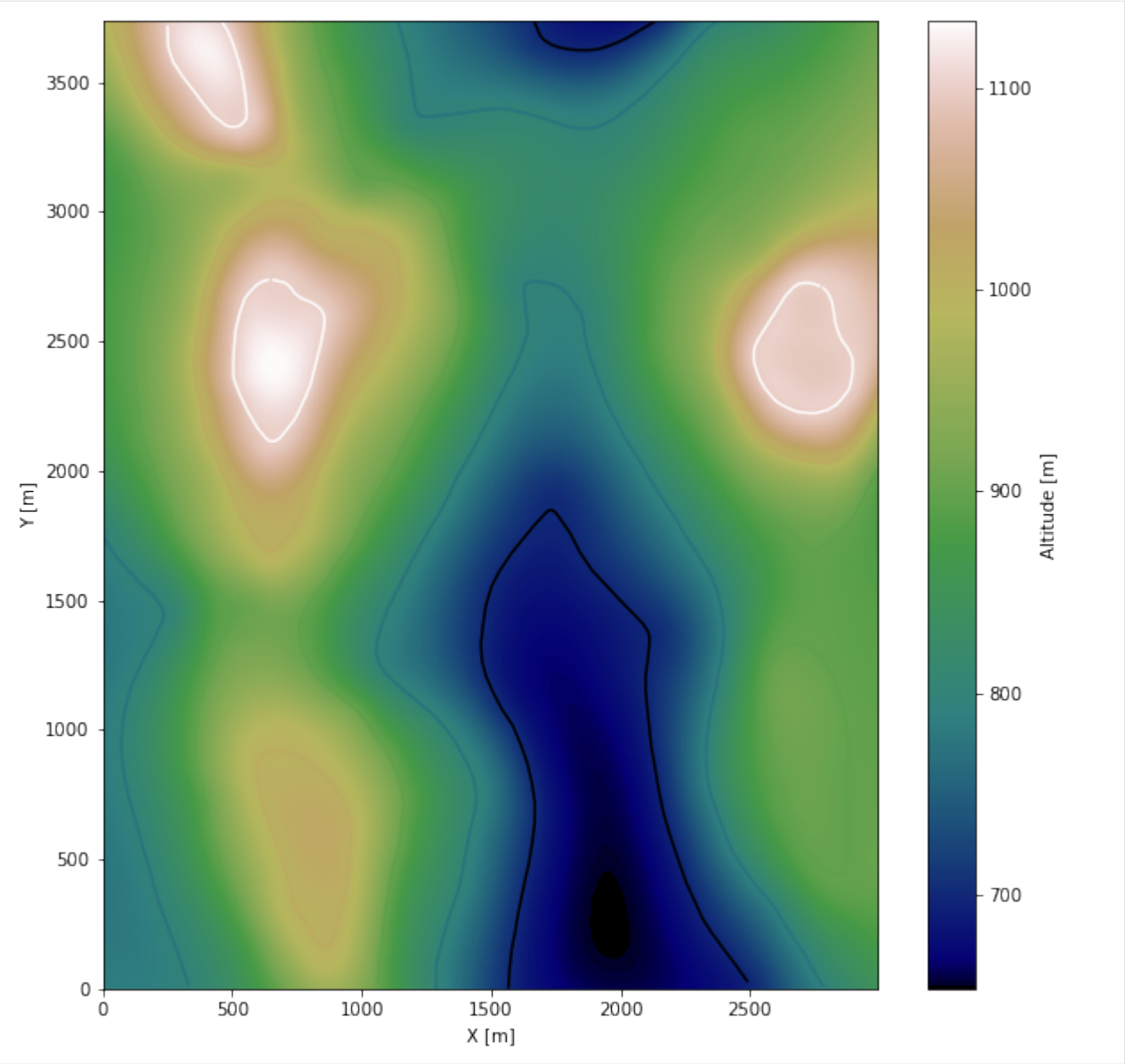
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 2991, 0, 3736], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2991)
ax.set_ylim(0, 3736)
```

[8]:
```
(0.0, 3736.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster13.tif', extent = [0, 2991, 0, 3736], crs ='EPSG : 4326', overwrite_f ile = True)$

Opening Raster

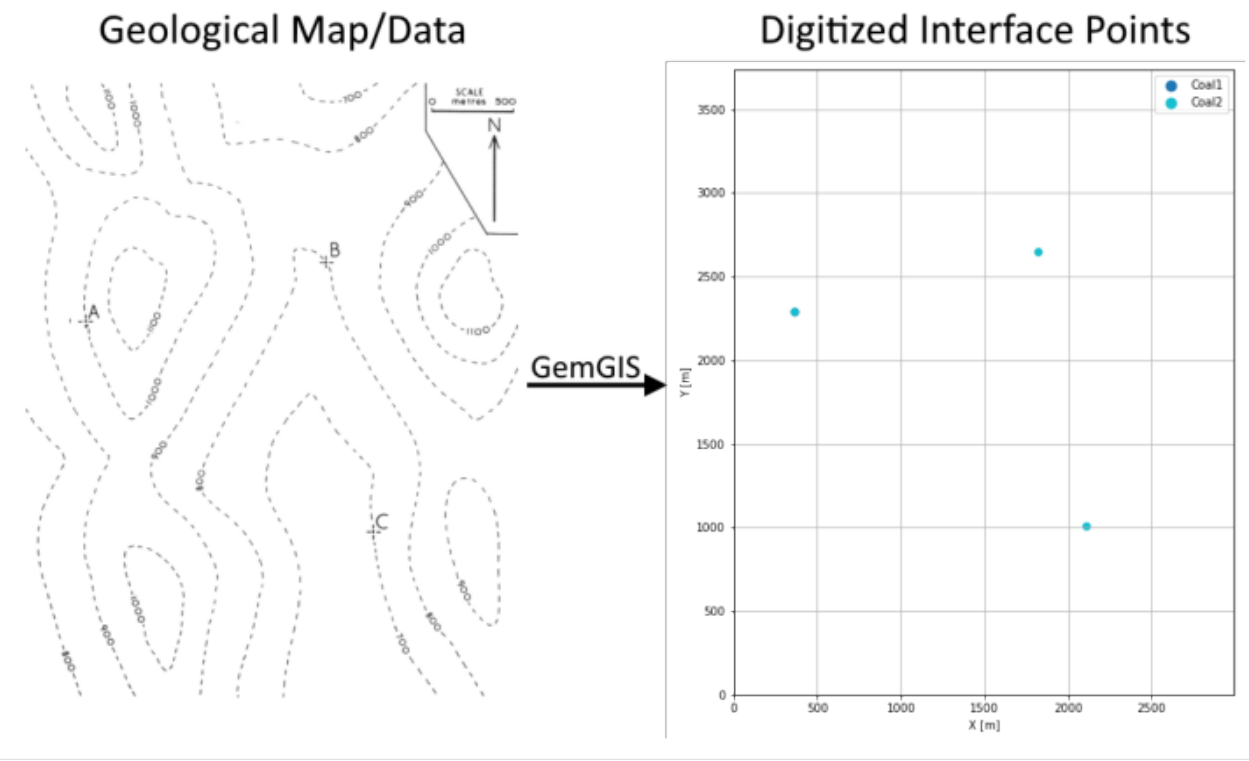
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster13.tif')
```

7.13.5 Interface Points of stratigraphic boundaries

The interface points for this three point example will be digitized as points with the respective height value as given by the borehole information and the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example13.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces13.shp')
interfaces.head()
```

[11]:

	id	formation	Z	geometry
0	None	Coal1	950	POINT (366.363 2287.350)
1	None	Coal2	550	POINT (366.363 2287.350)
2	None	Coal2	650	POINT (1822.758 2645.532)
3	None	Coal2	450	POINT (2110.811 1006.118)
4	None	Coal1	1050	POINT (1822.758 2645.532)

Extracting Z coordinate from Digital Elevation Model

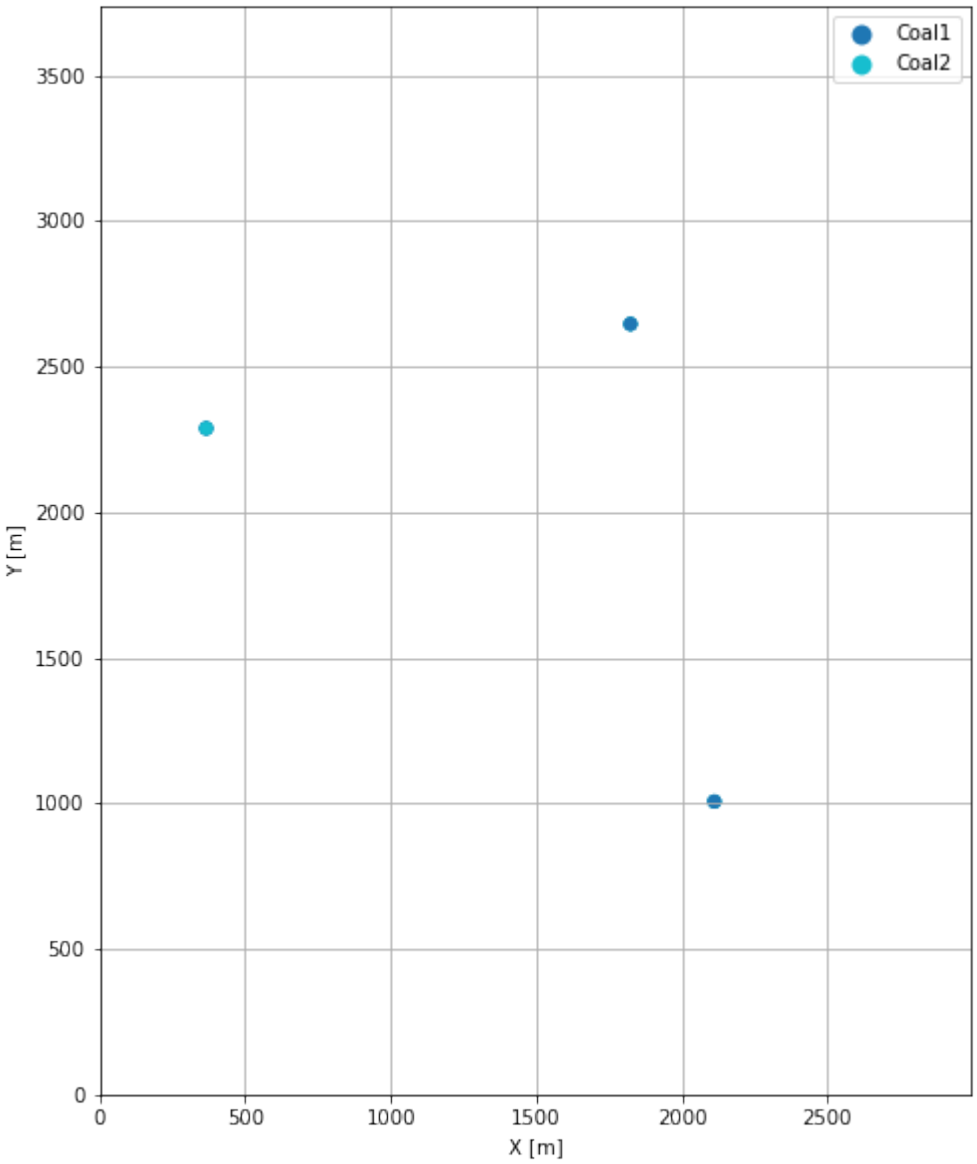
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=None)
interfaces_coords
```

[12]:
```
  formation       Z                    geometry       X       Y
0     Coal1  950.00   POINT (366.363 2287.350)  366.36 2287.35
1     Coal2  550.00   POINT (366.363 2287.350)  366.36 2287.35
2     Coal2  650.00  POINT (1822.758 2645.532) 1822.76 2645.53
3     Coal2  450.00  POINT (2110.811 1006.118) 2110.81 1006.12
4     Coal1 1050.00  POINT (1822.758 2645.532) 1822.76 2645.53
5     Coal1  850.00  POINT (2110.811 1006.118) 2110.81 1006.12
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2991)
ax.set_ylim(0, 3736)
```

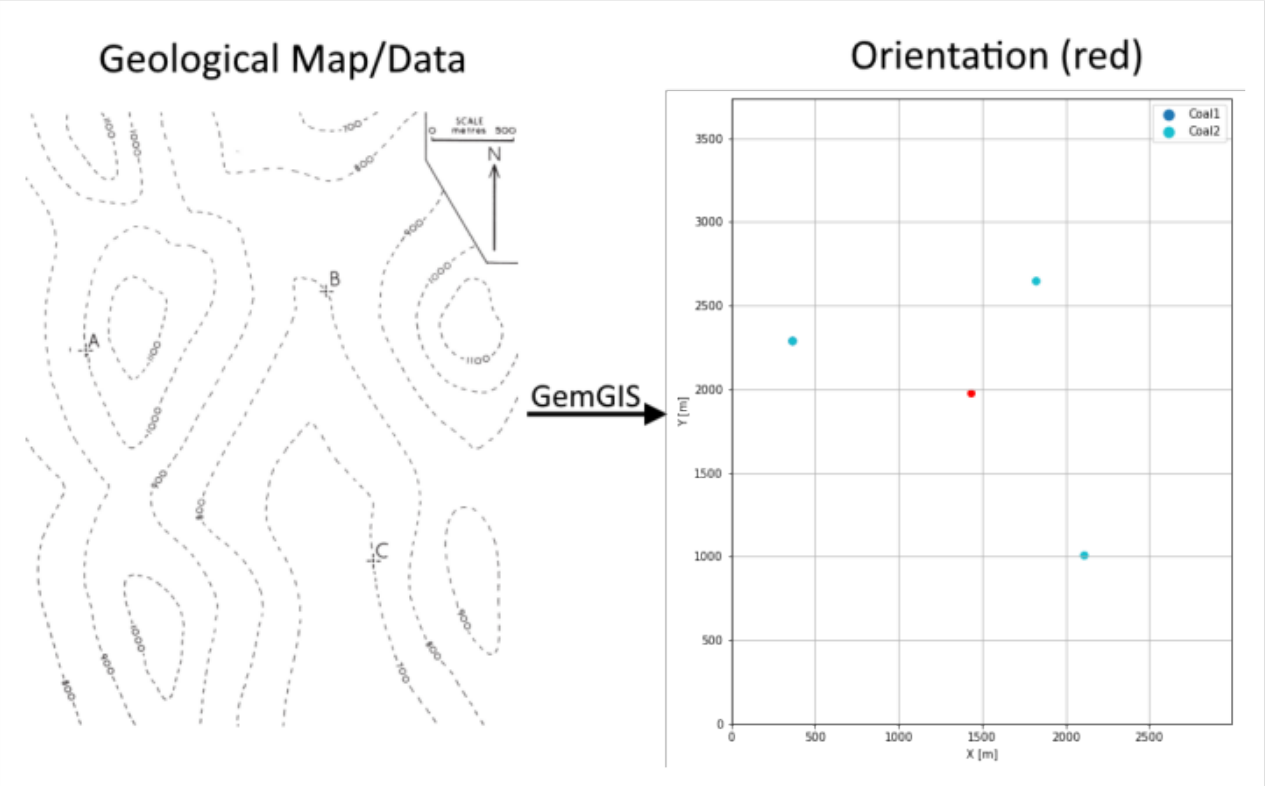
[13]:
```
(0.0, 3736.0)
```



7.13.6 Orientations from Strike Lines

For this three point example, an orientation is calculated using `gg.vector.calculate_orientation_for_three_point_problem()`.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example13.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
import pandas as pd
orientations1 = gg.vector.calculate_orientation_for_three_point_
↪problem(gdf=interfaces[interfaces['formation'] == 'Coal1'])
orientations2 = gg.vector.calculate_orientation_for_three_point_
↪problem(gdf=interfaces[interfaces['formation'] == 'Coal2'])
orientations = pd.concat([orientations1, orientations2]).reset_index()
orientations
```

[15]:
```
   index     Z formation azimuth    dip polarity       X       Y  \
0      0 950.0     Coal1   16.09 172.38        1 1433.31 1979.67
1      0 550.0     Coal2   16.09 172.38        1 1433.31 1979.67

                    geometry
0  POINT (1433.311 1979.667)
1  POINT (1433.311 1979.667)
```

Changing the Data Type of fields

[16]:
```
orientations['Z'] = orientations['Z'].astype(float)
orientations['azimuth'] = orientations['azimuth'].astype(float)
orientations['dip'] = orientations['dip'].astype(float)
orientations['dip'] = 180 - orientations['dip']
orientations['azimuth'] = 180 - orientations['azimuth']
orientations['polarity'] = orientations['polarity'].astype(float)
orientations['X'] = orientations['X'].astype(float)
```

(continues on next page)

(continued from previous page)

```
orientations['Y'] = orientations['Y'].astype(float)
orientations.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 2 entries, 0 to 1
Data columns (total 9 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   index      2 non-null      int64
 1   Z          2 non-null      float64
 2   formation  2 non-null      object
 3   azimuth    2 non-null      float64
 4   dip        2 non-null      float64
 5   polarity   2 non-null      float64
 6   X          2 non-null      float64
 7   Y          2 non-null      float64
 8   geometry   2 non-null      geometry
dtypes: float64(6), geometry(1), int64(1), object(1)
memory usage: 272.0+ bytes
```

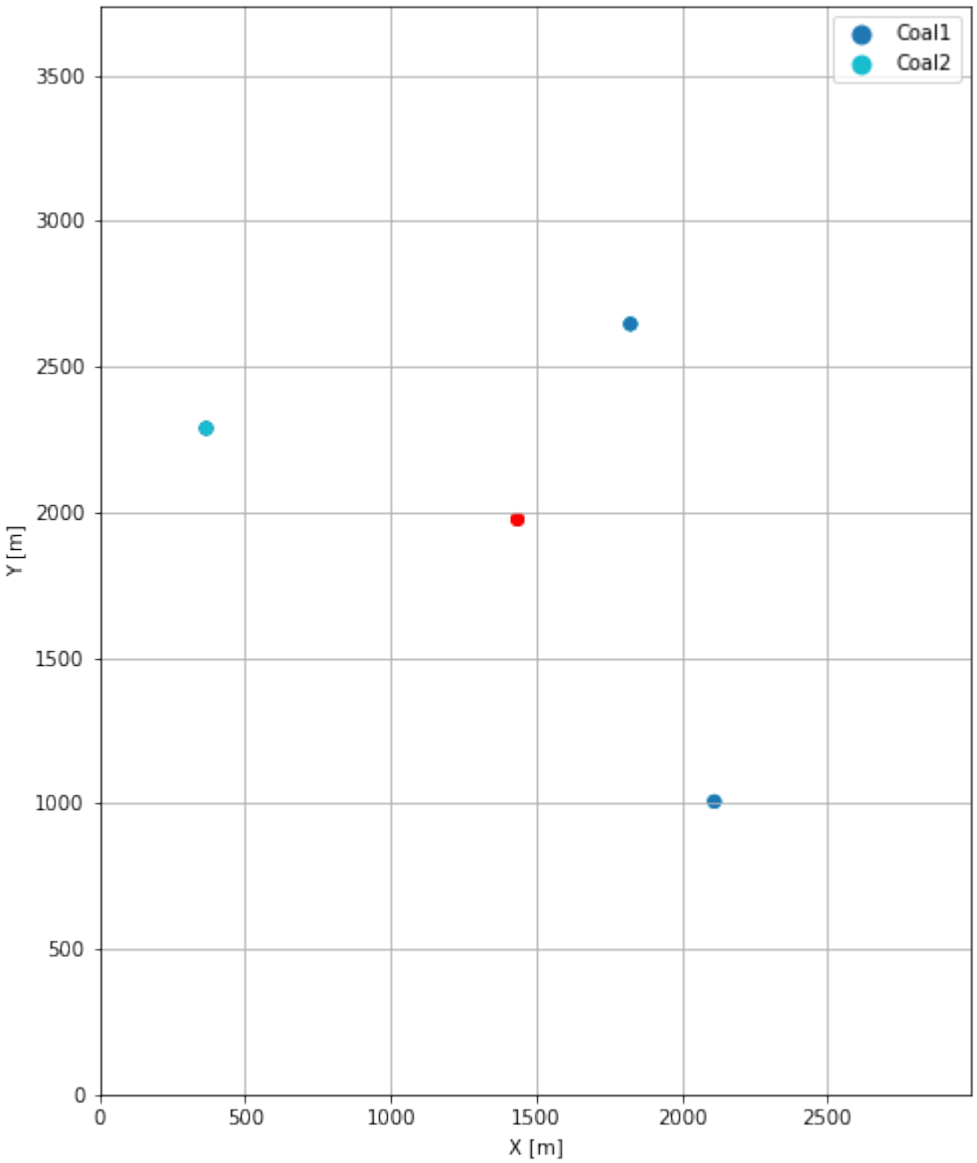
[17]:
```
orientations
```

[17]:
```
   index      Z formation  azimuth  dip  polarity       X       Y  \
0      0 950.00     Coal1   163.91 7.62      1.00 1433.31 1979.67
1      0 550.00     Coal2   163.91 7.62      1.00 1433.31 1979.67

                    geometry
0  POINT (1433.311 1979.667)
1  POINT (1433.311 1979.667)
```

Plotting the Orientations

[18]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2991)
ax.set_ylim(0, 3736)
```

[18]:
```
(0.0, 3736.0)
```



7.13.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[19]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[20]:
```
geo_model = gp.create_model('Model13')
geo_model
```

[20]:
```
Model13  2022-04-05 11:05
```

Initiate Data

[21]:
```
gp.init_data(geo_model, [0, 2991, 0, 3736, 250, 1200], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[21]:
```
Model13  2022-04-05 11:05
```

Model Surfaces

[22]:
```
geo_model.surfaces
```

[22]:

	surface	series	order_surfaces	color	id
0	Coal1	Default series	1	#015482	1
1	Coal2	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[23]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Coal1', 'Coal2'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[23]:

	surface	series	order_surfaces	color	id
0	Coal1	Strata1	1	#015482	1
1	Coal2	Strata1	2	#9f0052	2
2	Basement	Strata1	3	#ffbe00	3

Showing the Number of Data Points

[24]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[24]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Coal1	Strata1	1	#015482	1	3	1
1	Coal2	Strata1	2	#9f0052	2	3	1
2	Basement	Strata1	3	#ffbe00	3	0	0

Loading Digital Elevation Model

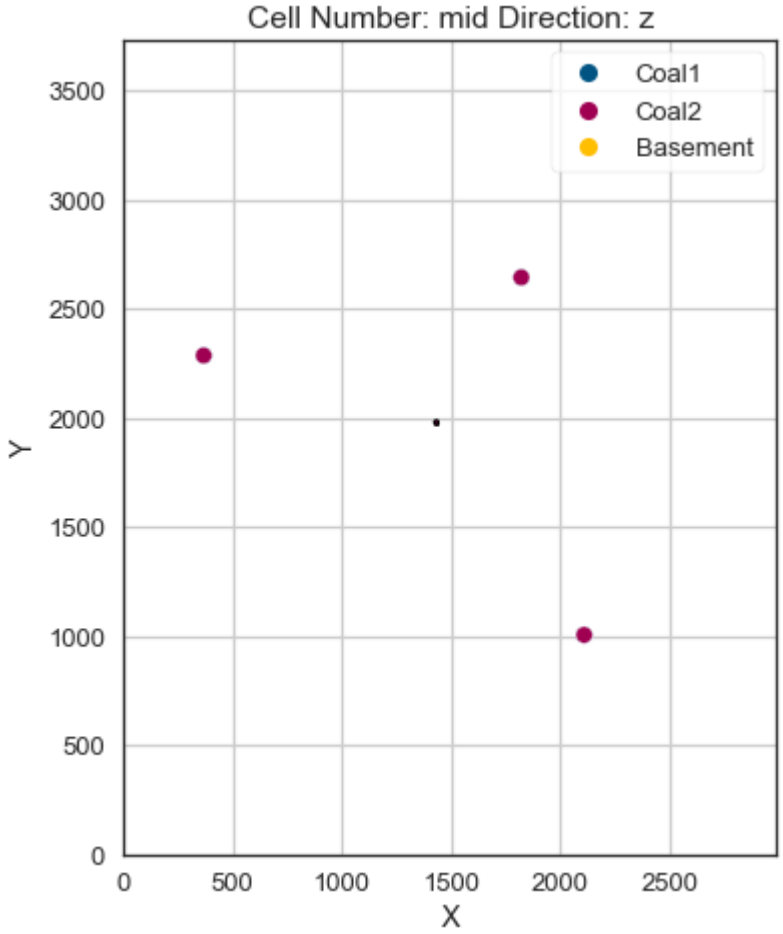
[25]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster13.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

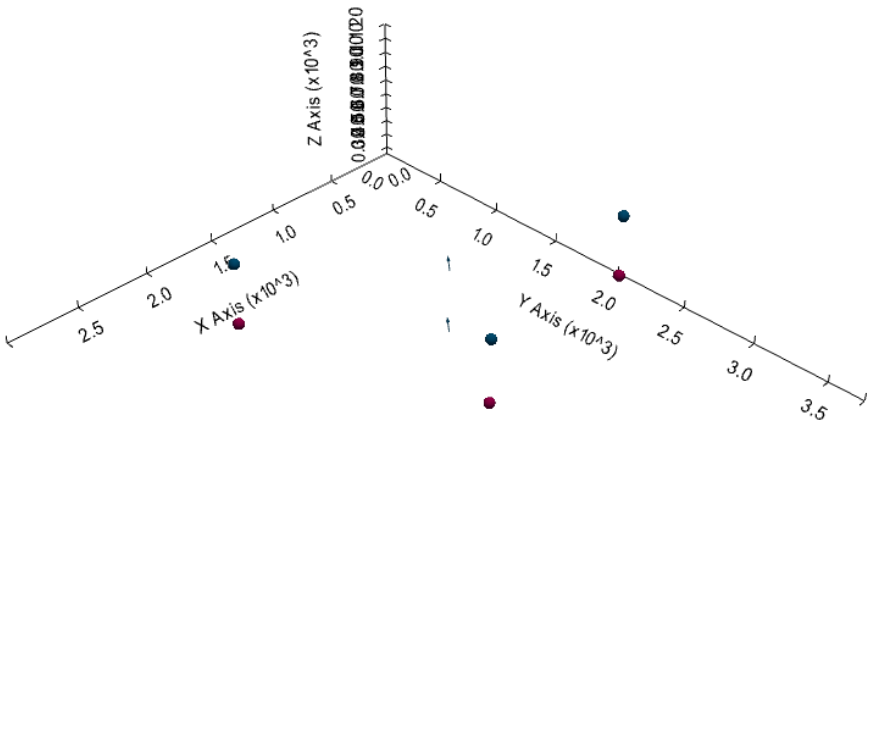
[25]:
```
Grid Object. Values:
array([[  14.955     ,   18.68      ,  254.75      ],
       [  14.955     ,   18.68      ,  264.25      ],
       [  14.955     ,   18.68      ,  273.75      ],
       ...,
       [2985.99832776, 3711.02673797,  907.33477783],
       [2985.99832776, 3721.01604278,  906.7131958 ],
       [2985.99832776, 3731.00534759,  906.11096191]])
```

Plotting Input Data

[26]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[27]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[27]:
```
<gempy.plot.vista.GemPyToVista at 0x1f20d692d30>
```

Setting the Interpolator

[28]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              4879.17
$C_o$            566816.12
drift equations       [3]
```

[28]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1f205cb0970>
```

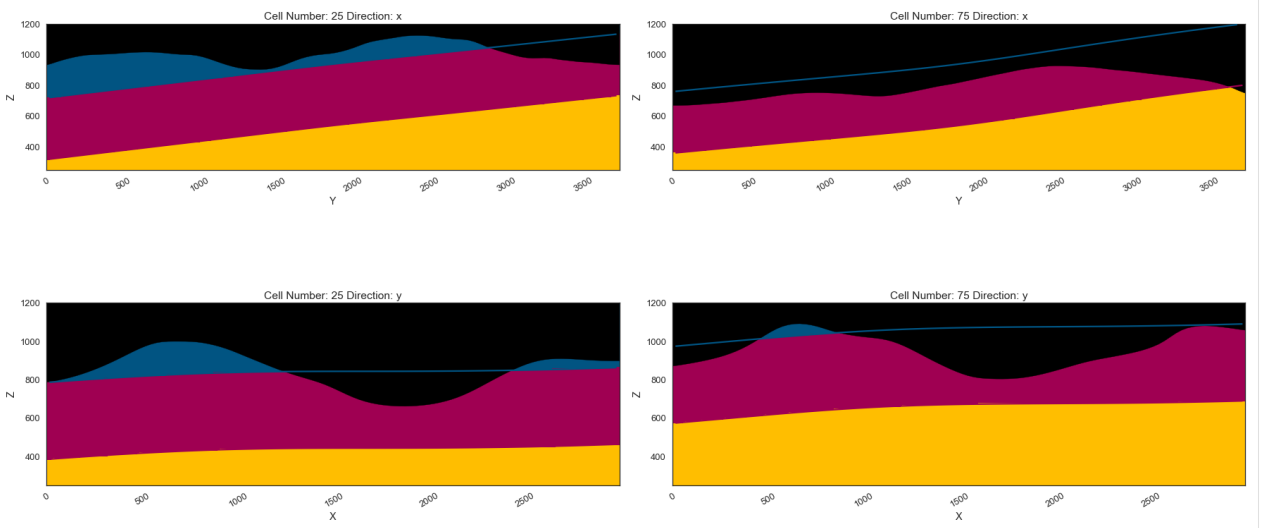
Computing Model

[29]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

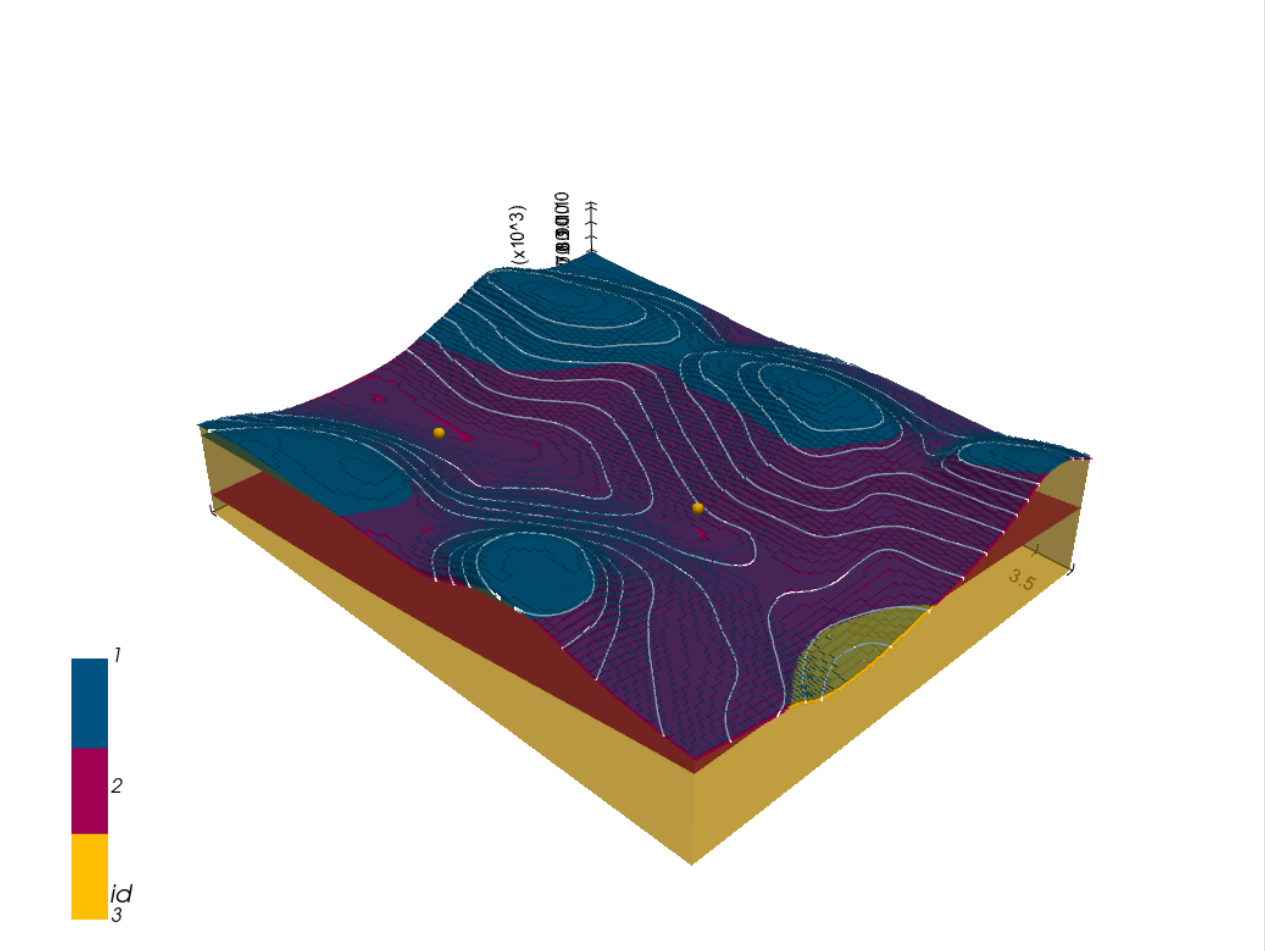
[30]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[30]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1f212ac02e0>
```



Plotting 3D Model

[31]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.14 Example 14 - Three Point Problem

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 2973 m wide (W-E extent) and 3698 m high (N-S extent). The vertical model extent varies between 0 m and 1000 m. This example represents a classic "three-point-problem" of planar dipping layers (blue and purple) above an unspecified basement. The interface points were not recorded at the surface but rather in boreholes at depth.

The map has been georeferenced with QGIS. The outcrops of the layers were digitized in QGIS. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

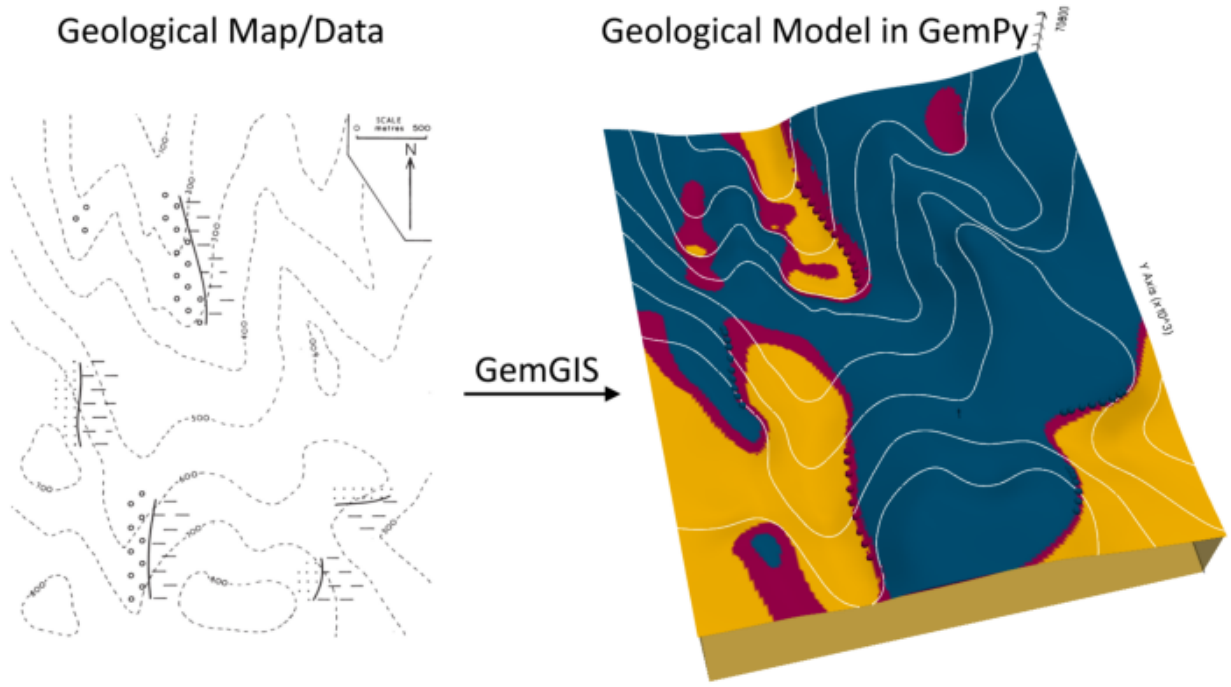
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example14.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.14.1 Licensing

7.14.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.14.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

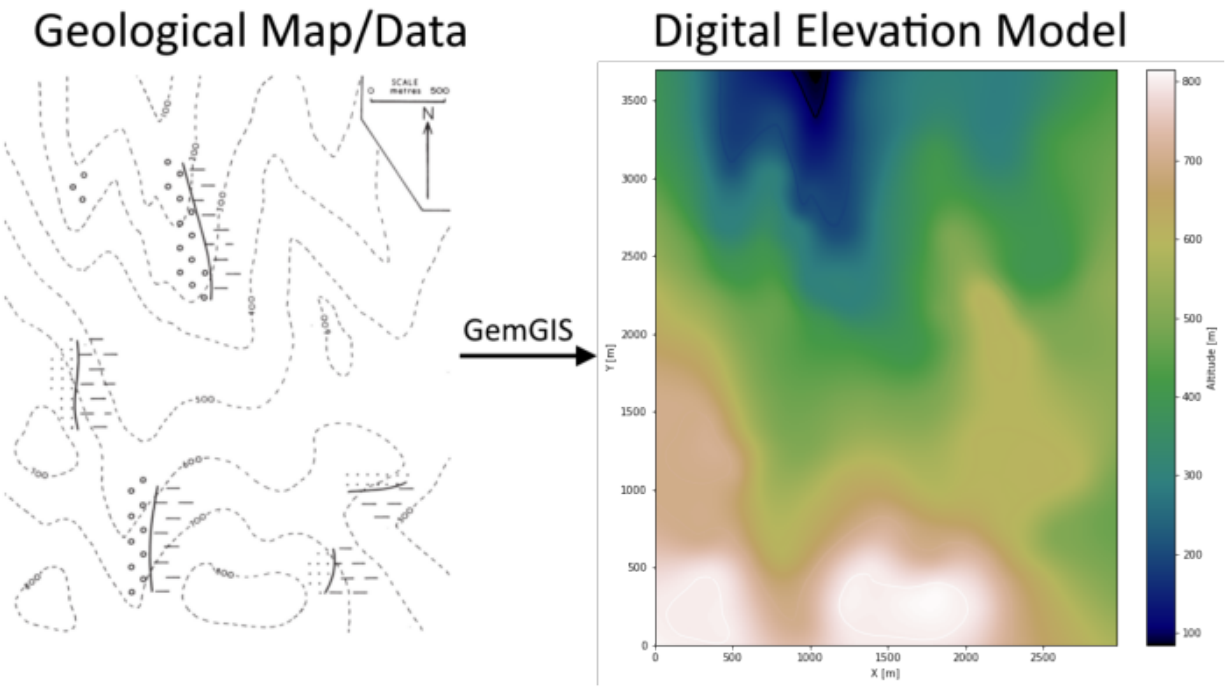
[4]:
```
file_path = 'data/example14/'
gg.download_gemgis_data.download_tutorial_data(filename="example14_three_point_problem.
↪zip", dirpath=file_path)
```

```
Downloading file 'example14_three_point_problem.zip' from 'https://rwth-aachen.sciebo.de/
↪s/AfXRsZywYDbUF34/download?path=%2Fexample14_three_point_problem.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\example\data\example14'.
```

7.14.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example14.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo14.shp')
topo.head()
```

[6]:
```
     id    Z                                                geometry
0  None  800  LINESTRING (373.021 427.168, 400.177 396.582, ...
1  None  800  LINESTRING (1394.650 431.742, 1406.656 411.732...
2  None  700  LINESTRING (3.131 921.975, 58.014 877.954, 104...
3  None  700  LINESTRING (263.826 1502.823, 293.269 1491.960...
4  None  600  LINESTRING (2110.419 2231.741, 2145.293 2212.3...
```

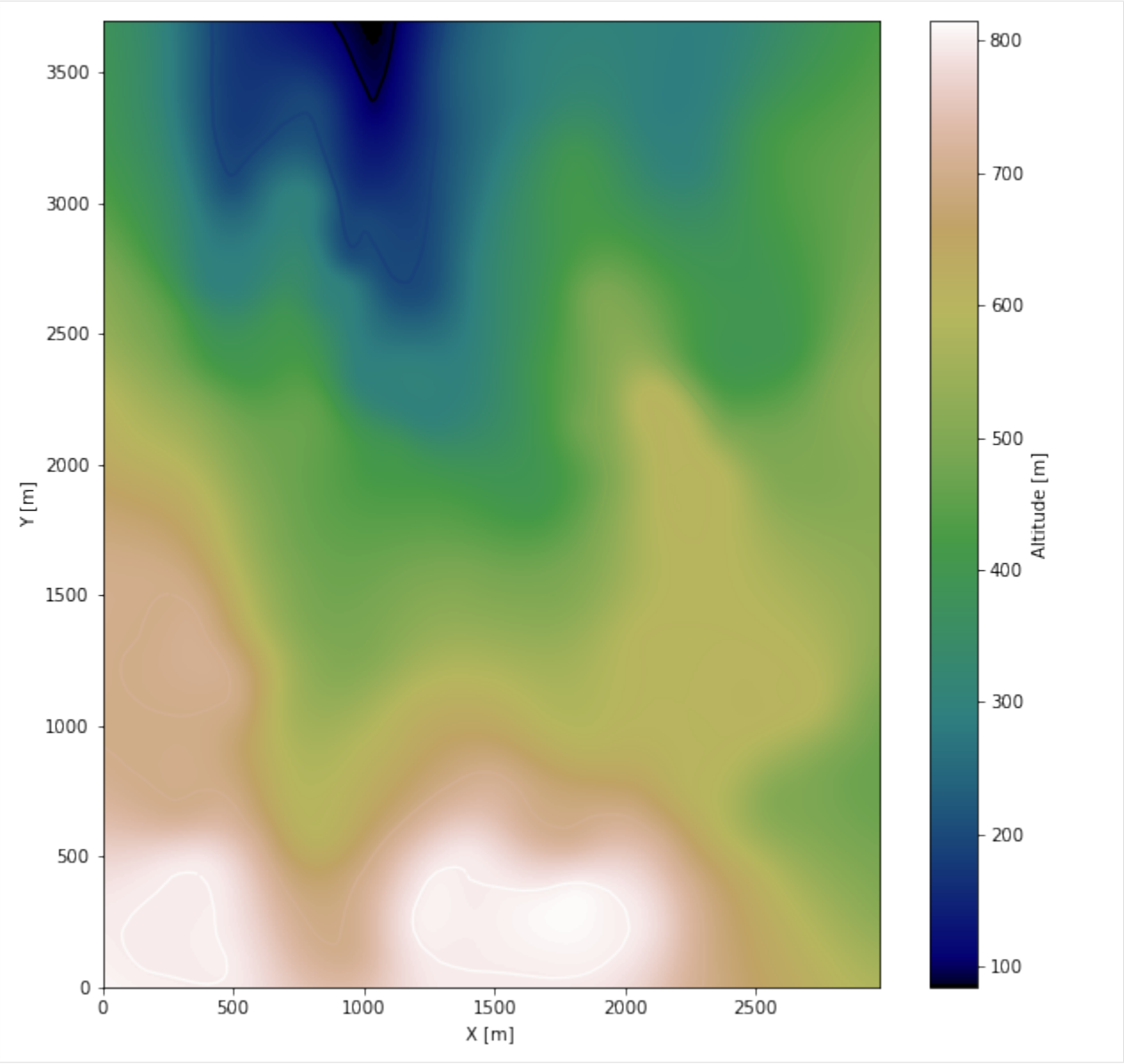
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=15)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 2973, 0, 3698], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2973)
ax.set_ylim(0, 3698)
```

[8]:
```
(0.0, 3698.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster14.tif', extent = [0, 2973, 0, 3698], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

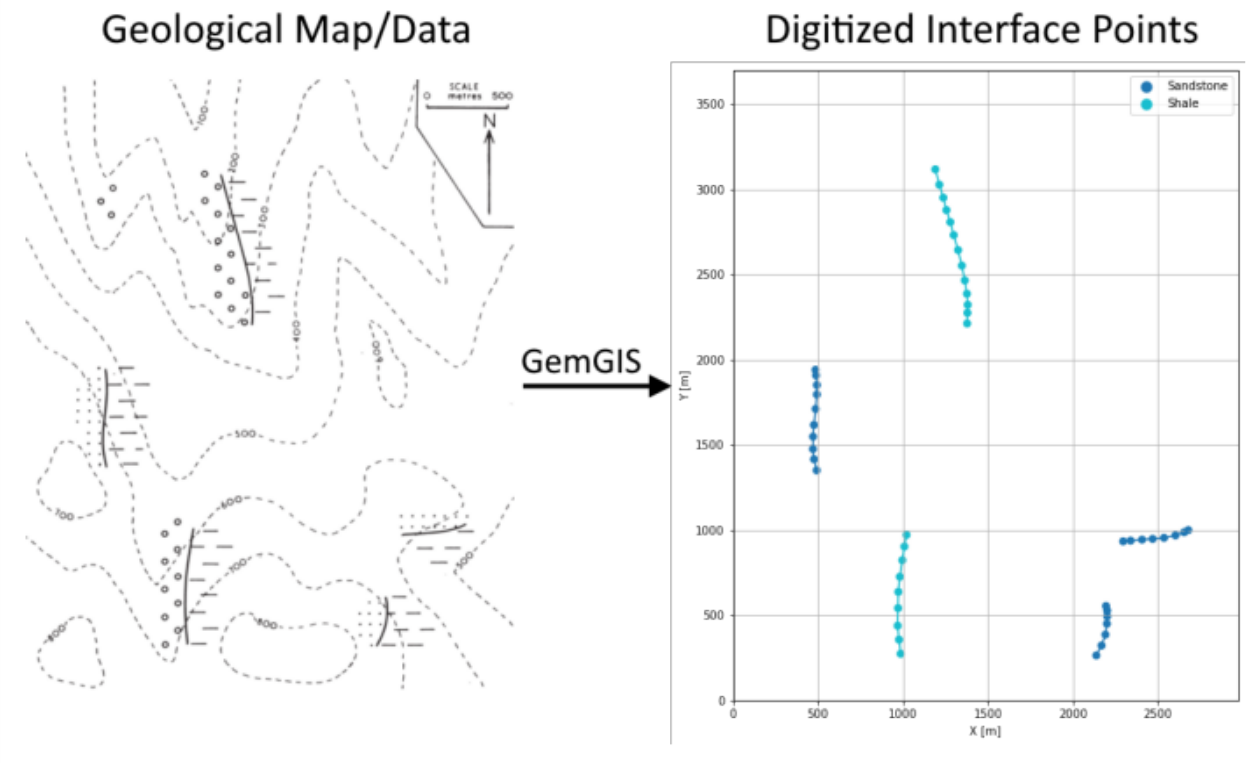
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster14.tif')
```

7.14.5 Interface Points of stratigraphic boundaries

The interface points for this three point example will be digitized as points with the respective height value as given by the contour lines and the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example14.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces14.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Sandstone	LINESTRING (491.899 1350.893, 476.463 1416.067...
1	None	Shale	LINESTRING (986.420 273.809, 976.701 356.706, ...
2	None	Sandstone	LINESTRING (2137.825 264.091, 2168.697 322.404...
3	None	Sandstone	LINESTRING (2295.614 934.124, 2339.635 938.126...
4	None	Shale	LINESTRING (1190.517 3118.020, 1213.957 3027.6...

Extracting Z coordinate from Digital Elevation Model

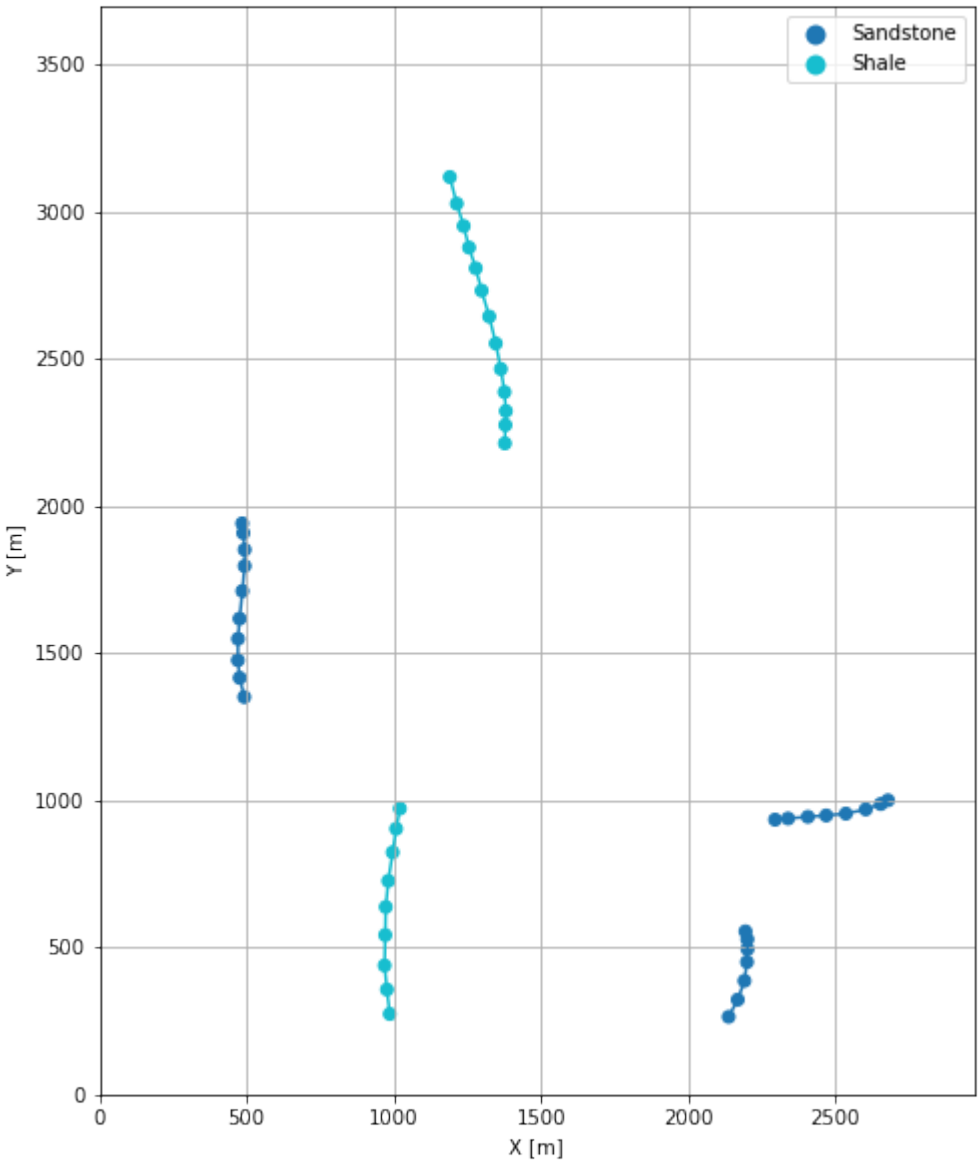
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords.head()
```

[12]:
```
   formation                  geometry      X       Y      Z
0  Sandstone  POINT (491.899 1350.893) 491.90 1350.89 663.18
1  Sandstone  POINT (476.463 1416.067) 476.46 1416.07 656.31
2  Sandstone  POINT (470.174 1476.096) 470.17 1476.10 643.79
3  Sandstone  POINT (470.746 1548.702) 470.75 1548.70 625.64
4  Sandstone  POINT (475.891 1617.306) 475.89 1617.31 607.52
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2973)
ax.set_ylim(0, 3698)
```

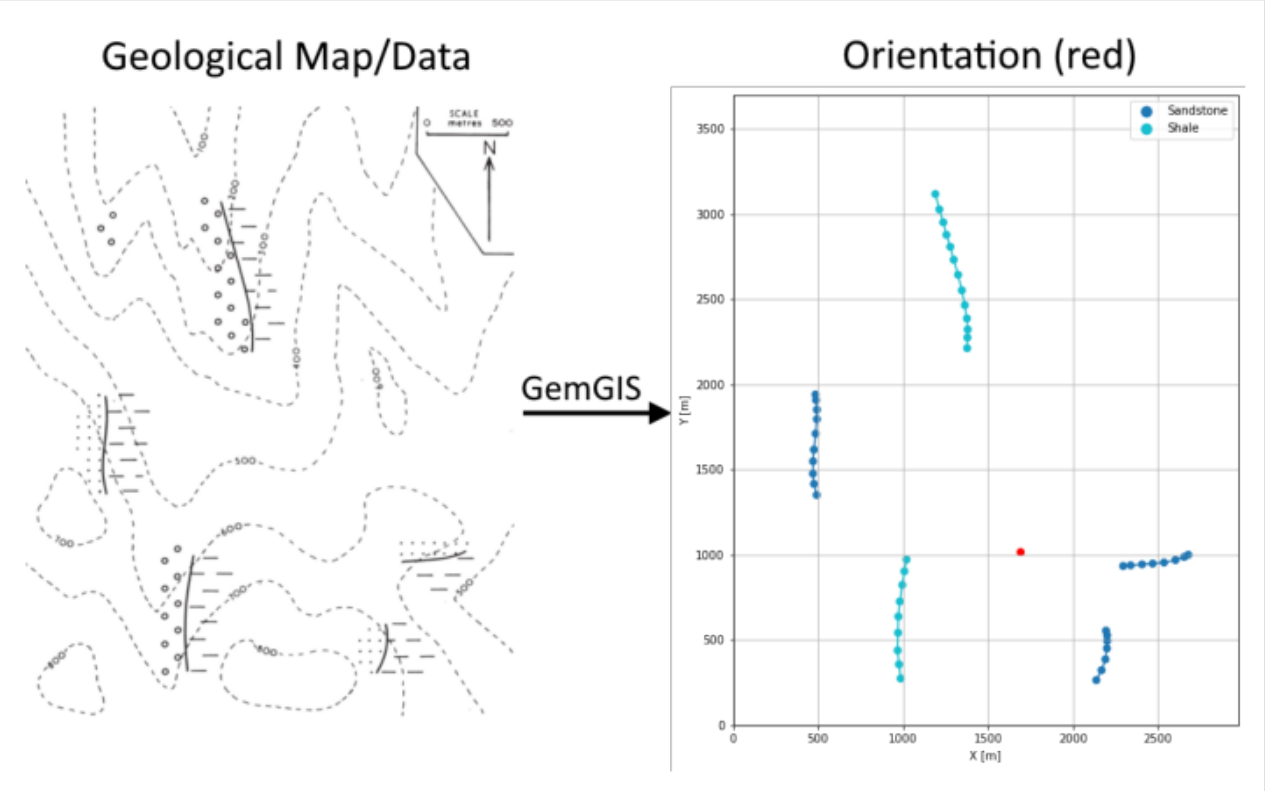
[13]:
```
(0.0, 3698.0)
```



7.14.6 Orientations from Strike Lines

For this three point example, an orientation is calculated using `gg.vector.calculate_orientation_for_three_point_problem()`.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example14.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
orientations = gpd.read_file(file_path + 'orientations14.shp')
orientations
```

[15]:
```
     id  formation    Z                  geometry
0  None  Sandstone  600  POINT (477.321 1628.597)
1  None  Sandstone  700  POINT (2201.427 477.764)
2  None  Sandstone  600  POINT (2390.088 942.556)
```

[16]:
```
orientations = gg.vector.calculate_orientation_for_three_point_problem(gdf=orientations)
orientations
```

[16]:
```
       Z  formation azimuth    dip polarity       X       Y  \
0 633.33  Sandstone -160.27 168.72        1 1689.61 1016.31

                    geometry
0  POINT (1689.612 1016.306)
```

Changing the Data Type of Fields

[17]:
```
orientations['Z'] = orientations['Z'].astype(float)
orientations['azimuth'] = orientations['azimuth'].astype(float)
orientations['dip'] = orientations['dip'].astype(float)
orientations['dip'] = 180 - orientations['dip']
orientations['azimuth'] = 180 - orientations['azimuth']
orientations['polarity'] = orientations['polarity'].astype(float)
orientations['X'] = orientations['X'].astype(float)
```

(continues on next page)

(continued from previous page)

```
orientations['Y'] = orientations['Y'].astype(float)
orientations.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Z          1 non-null      float64
 1   formation  1 non-null      object
 2   azimuth    1 non-null      float64
 3   dip        1 non-null      float64
 4   polarity   1 non-null      float64
 5   X          1 non-null      float64
 6   Y          1 non-null      float64
 7   geometry   1 non-null      geometry
dtypes: float64(6), geometry(1), object(1)
memory usage: 192.0+ bytes
```

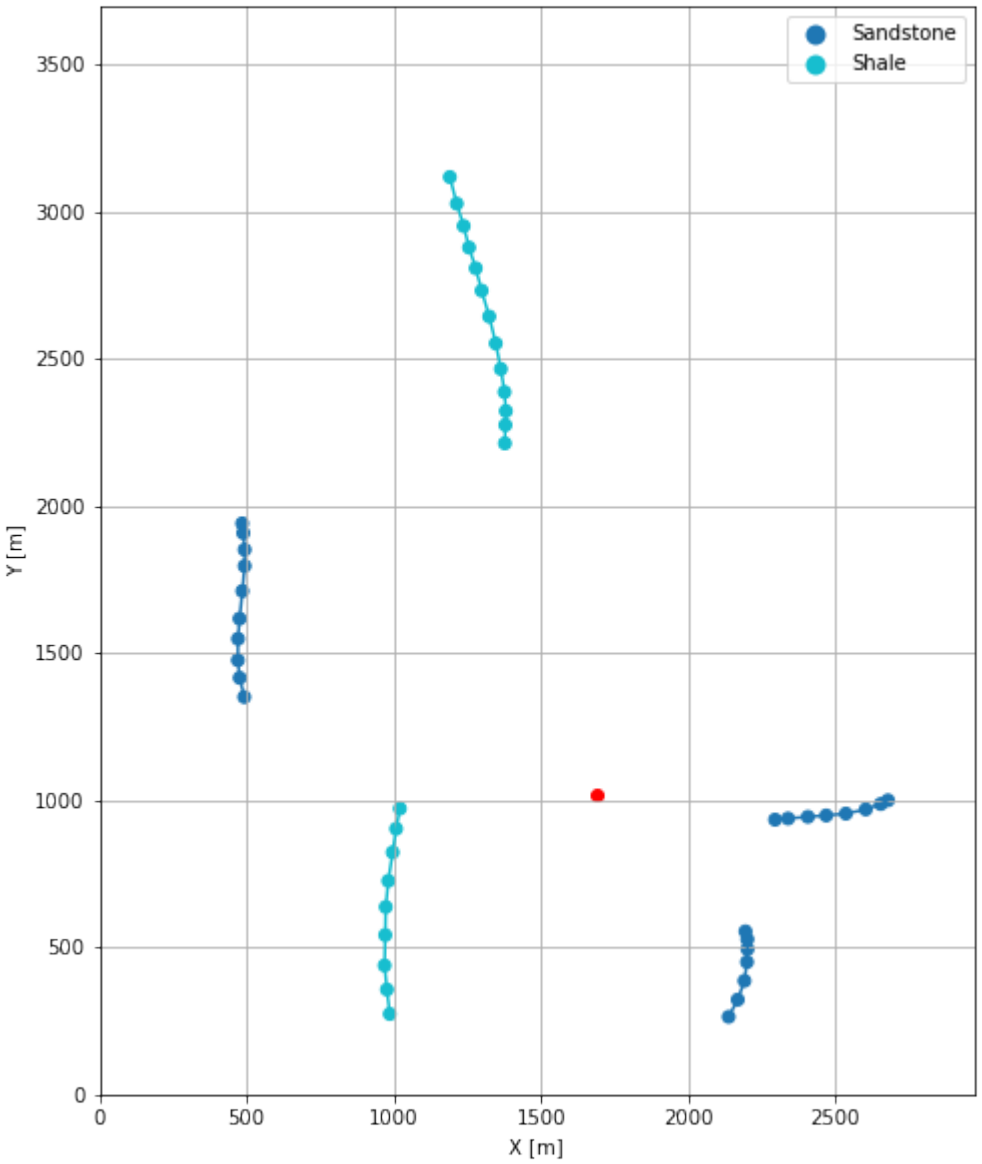
[18]:
```
orientations
```

[18]:
```
       Z  formation  azimuth   dip  polarity       X       Y  \
0 633.33  Sandstone   340.27 11.28      1.00 1689.61 1016.31

                     geometry
0  POINT (1689.612 1016.306)
```

Plotting the Orientations

[19]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 2973)
ax.set_ylim(0, 3698)
```

[19]:
```
(0.0, 3698.0)
```



7.14.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[20]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[21]:
```
geo_model = gp.create_model('Model14')
geo_model
```

[21]:
```
Model14  2022-04-05 11:06
```

Initiate Data

[22]:
```
gp.init_data(geo_model, [0, 2973, 0, 3698, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[22]:
```
Model14  2022-04-05 11:06
```

Model Surfaces

[23]:
```
geo_model.surfaces
```

[23]:
```
     surface          series  order_surfaces    color  id
0  Sandstone  Default series               1  #015482   1
1      Shale  Default series               2  #9f0052   2
```

Mapping the Stack to Surfaces

[24]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Sandstone', 'Shale'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Conglomerate')
```

[24]:
```
        surface   series  order_surfaces    color  id
0     Sandstone  Strata1               1  #015482   1
1         Shale  Strata1               2  #9f0052   2
2  Conglomerate  Strata1               3  #ffbe00   3
```

Showing the Number of Data Points

[25]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[25]:
```
        surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0     Sandstone  Strata1               1  #015482   1                 25
↪    1
```

(continues on next page)

(continued from previous page)

```
1          Shale  Strata1               2  #9f0052   2                 22                ␣
↪    0
2  Conglomerate  Strata1               3  #ffbe00   3                  0                ␣
↪    0
```

Loading Digital Elevation Model

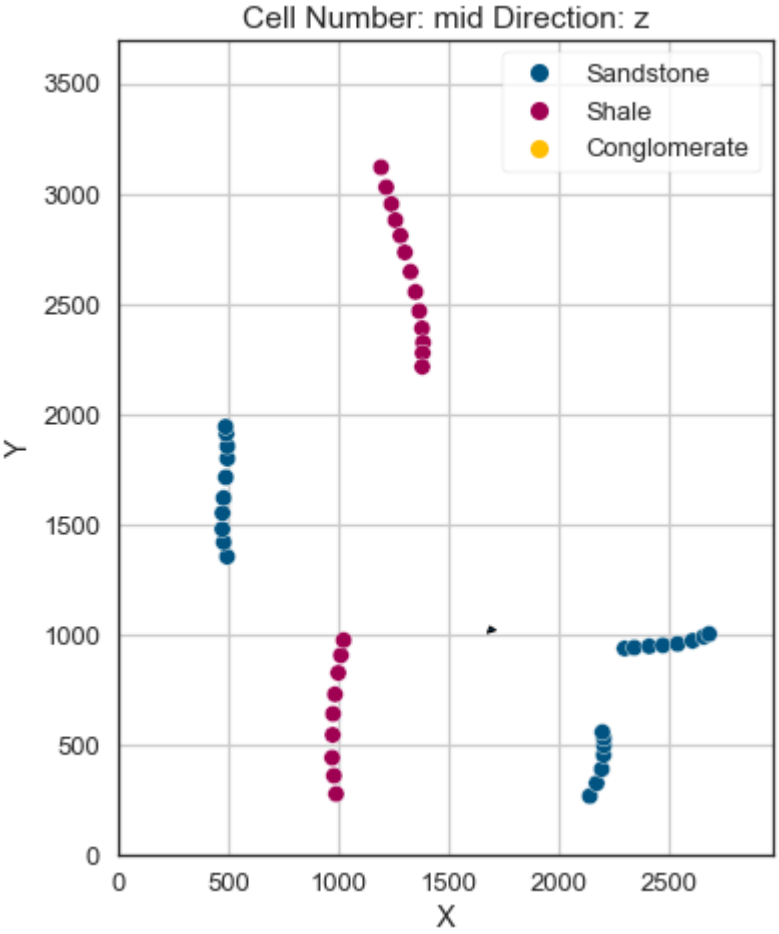
[26]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster14.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

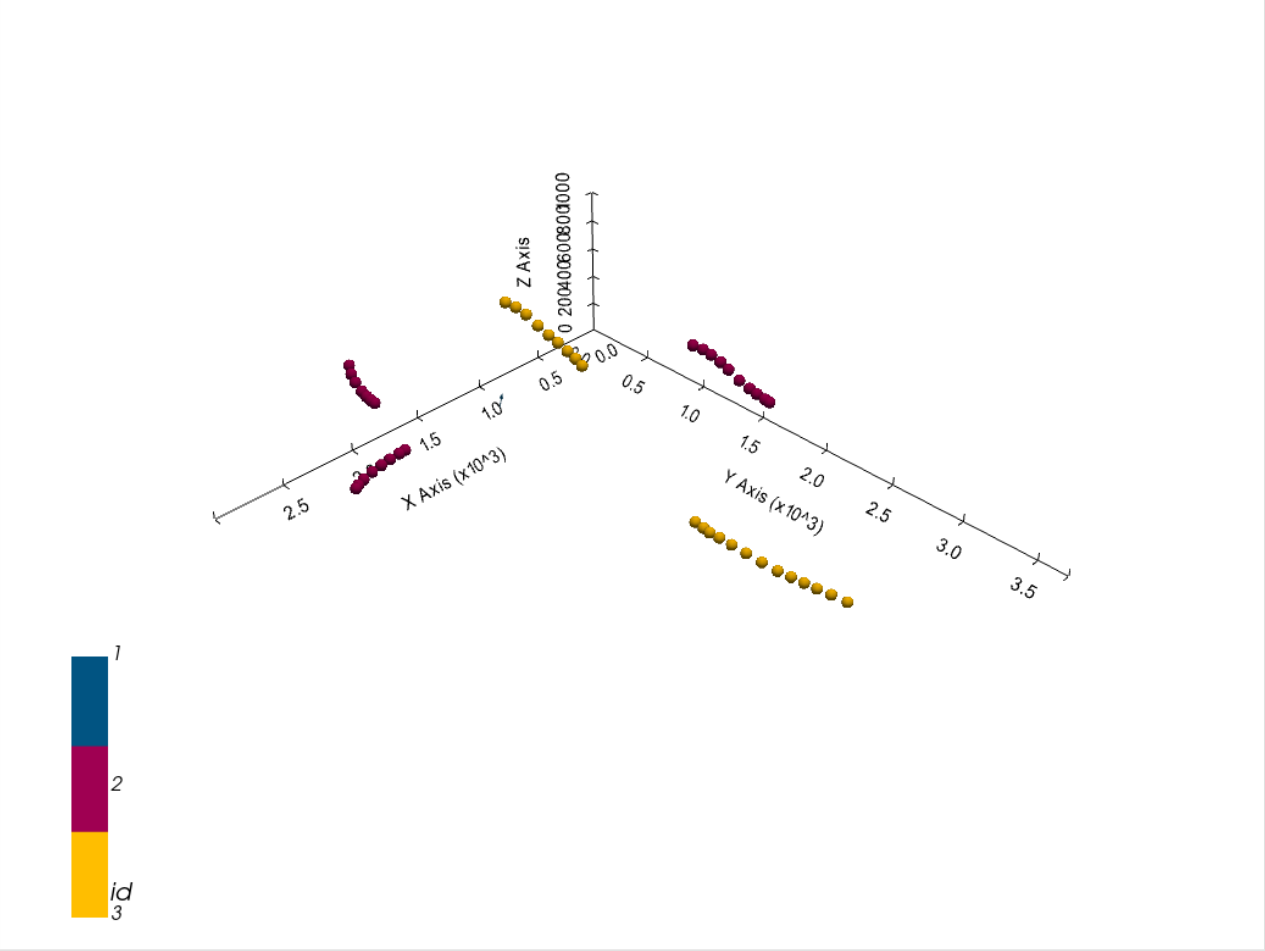
[26]:
```
Grid Object. Values:
array([[  14.865     ,   18.49      ,    5.        ],
       [  14.865     ,   18.49      ,   15.        ],
       [  14.865     ,   18.49      ,   25.        ],
       ...,
       [2965.49242424, 3660.5708502 ,  428.21966553],
       [2965.49242424, 3675.54251012,  427.20385742],
       [2965.49242424, 3690.51417004,  426.23760986]])
```

Plotting Input Data

[27]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[28]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[28]:
```
<gempy.plot.vista.GemPyToVista at 0x1b5c3679b20>
```

Setting the Interpolator

[29]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              4849.12
$C_o$            559855.55
drift equations       [3]
```

[29]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1b5bcc82a60>
```

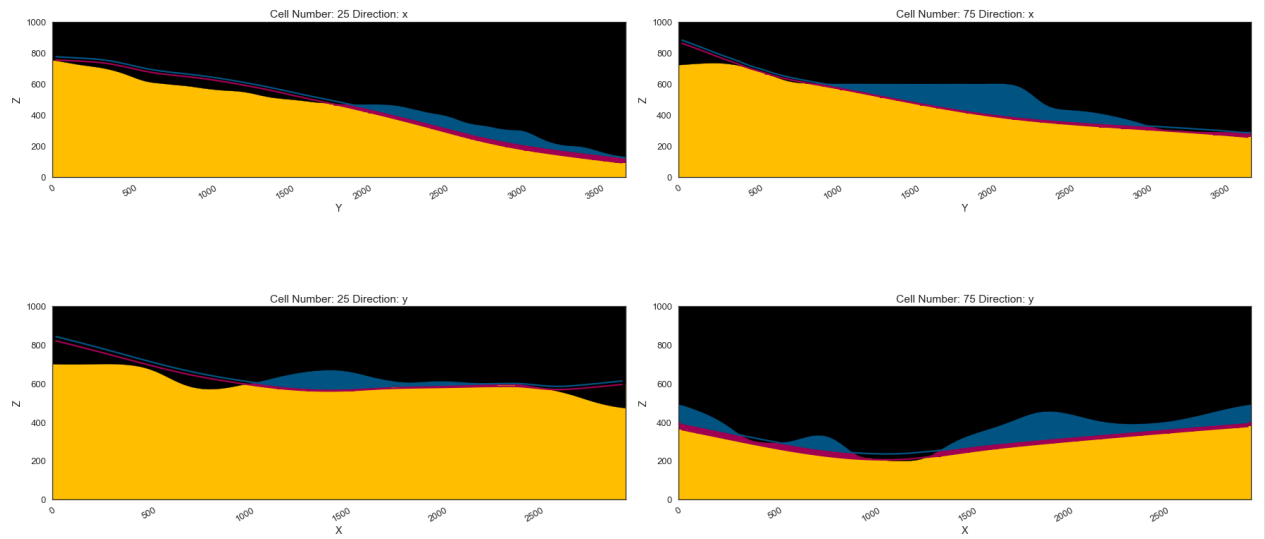
Computing Model

[30]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

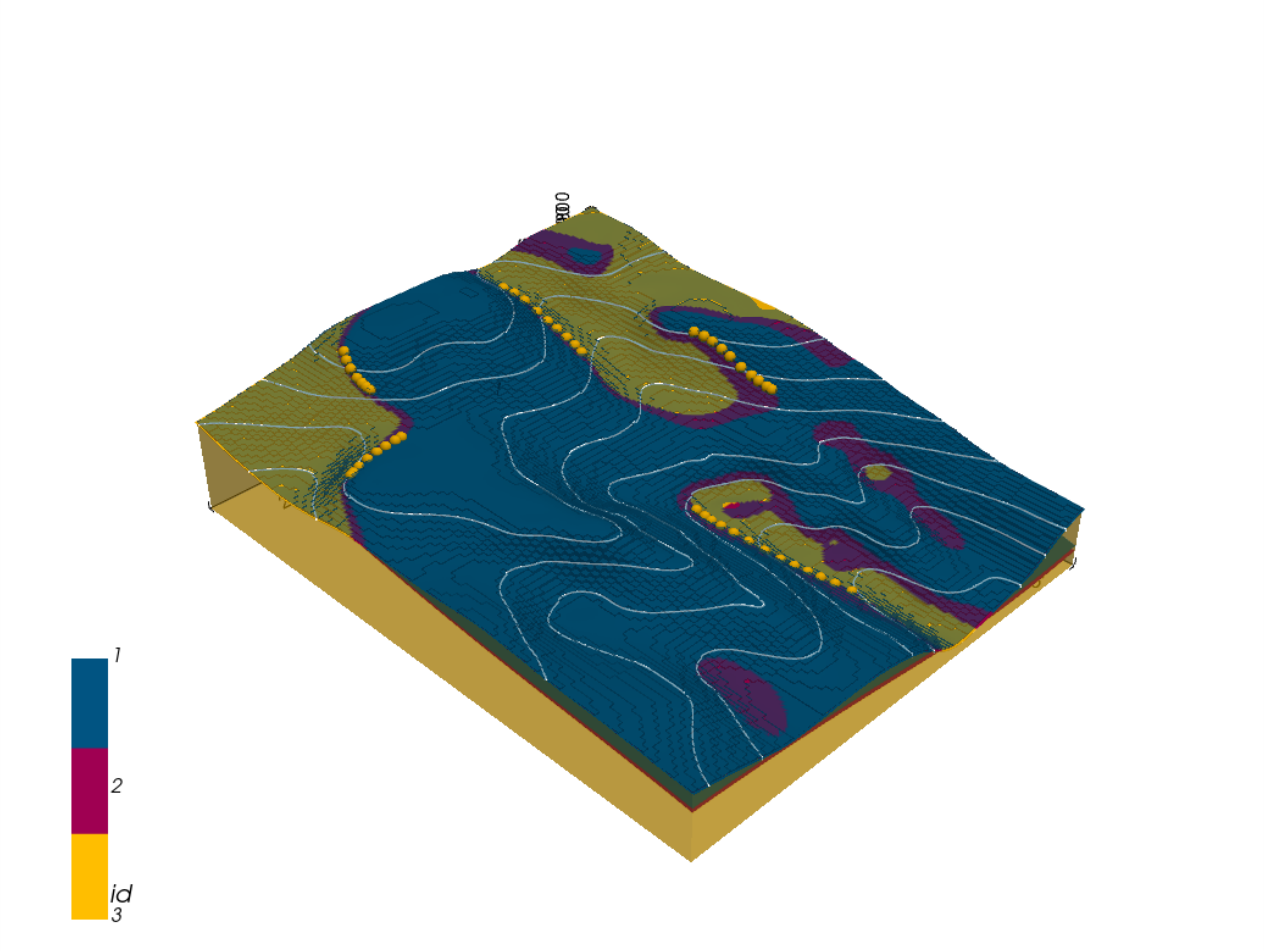
[31]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[31]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1b5c9ea4820>
```



Plotting 3D Model

[32]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.15 Example 15 - Three Point Problem

This example will show how to convert the geological map below using GemGIS to a GemPy model. This example is based on digitized data. The area is 1187 m wide (W-E extent) and 1479 m high (N-S extent). The vertical model extent varies between 100 m and 300 m. This example represents a classic "three-point-problem" of planar dipping layers (green and yellow) above an unspecified basement (purple) which are separated by a fault (blue). The interface points were not recorded at the surface but rather in boreholes at depth. The fault has an offset of 20 m but no further interface points are located beyond the fault. This will be dealt with in a two model approach.

The map has been georeferenced with QGIS. The outcrops of the layers were digitized in QGIS. The contour lines were also digitized and will be interpolated with GemGIS to create a topography for the model.

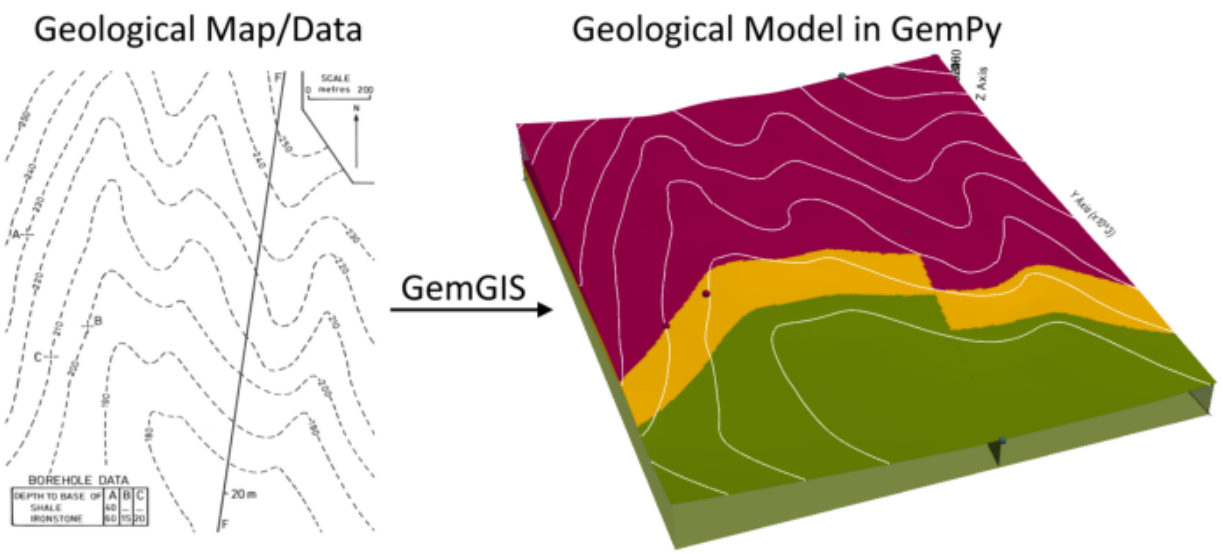
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example15.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.15.1 Licensing

7.15.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.15.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

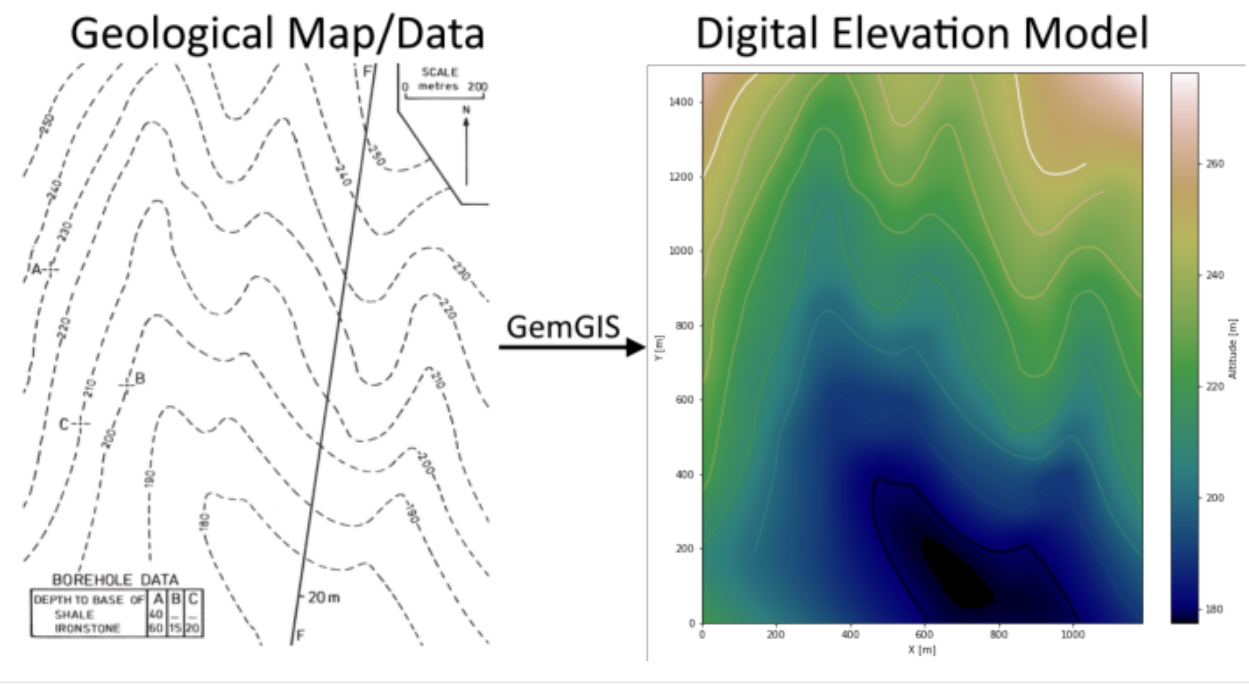
[4]:
```
file_path = 'data/example15/'
gg.download_gemgis_data.download_tutorial_data(filename="example15_three_point_problem.
↪zip", dirpath=file_path)
```

```
Downloading file 'example15_three_point_problem.zip' from 'https://rwth-aachen.sciebo.de/
↪s/AfXRsZywYDbUF34/download?path=%2Fexample15_three_point_problem.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\example\data\example15'.
```

7.15.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the SciPy Radial Basis Function interpolation wrapped in GemGIS. The respective function used for that is gg.vector.interpolate_raster().

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example15.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo15.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	180	LINESTRING (608.177 -0.021, 598.911 22.516, 58...
1	None	190	LINESTRING (323.662 216.425, 321.832 254.178, ...
2	None	200	LINESTRING (142.794 190.113, 153.433 227.980, ...
3	None	250	LINESTRING (1.395 1193.695, 20.385 1232.592, 3...
4	None	240	LINESTRING (1.623 925.311, 8.487 939.039, 13.7...

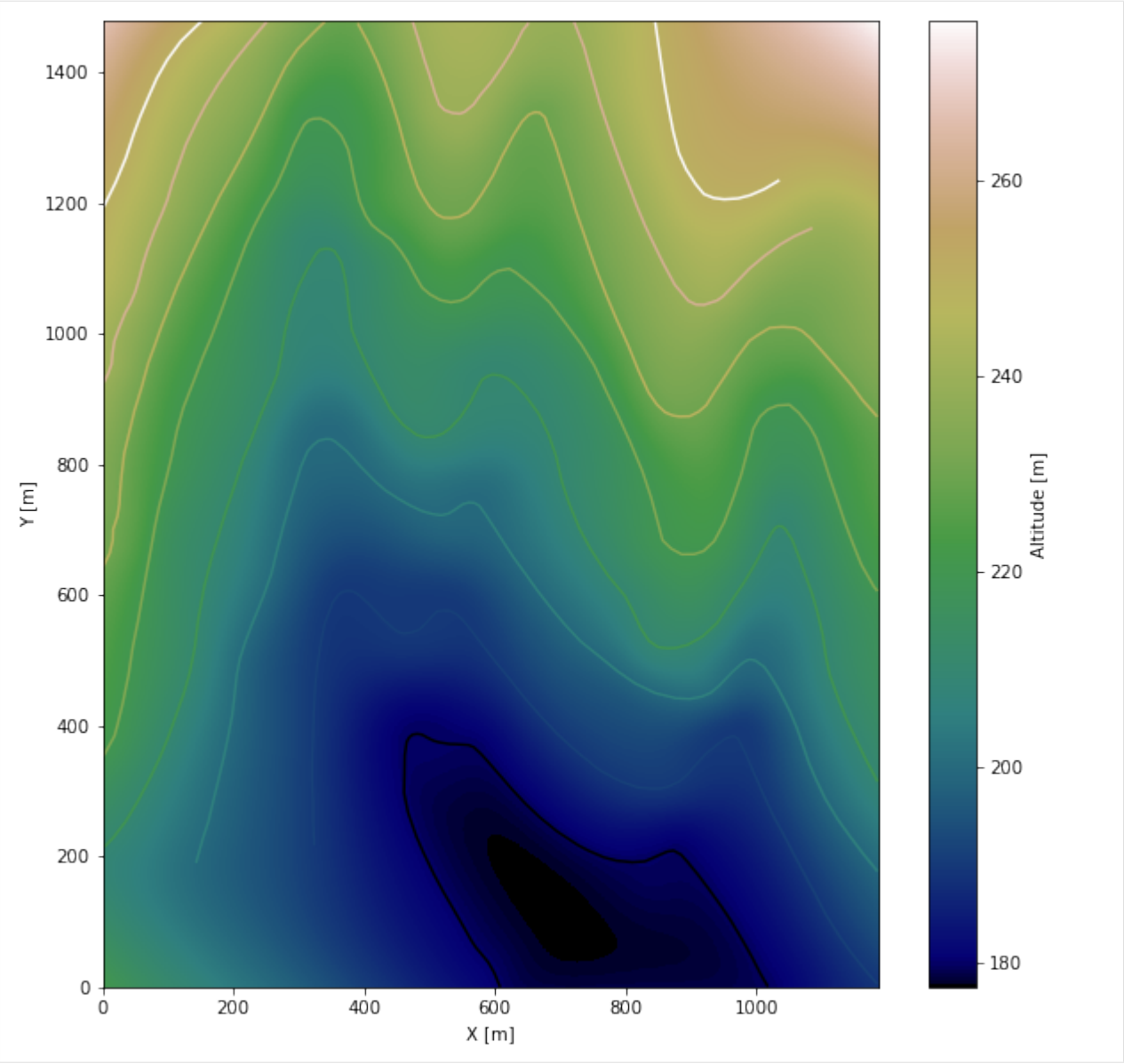
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 1187, 0, 1479], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 1187)
ax.set_ylim(0, 1479)
```

[8]:
```
(0.0, 1479.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster15.tif', extent = [0, 1187, 0, 1479], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

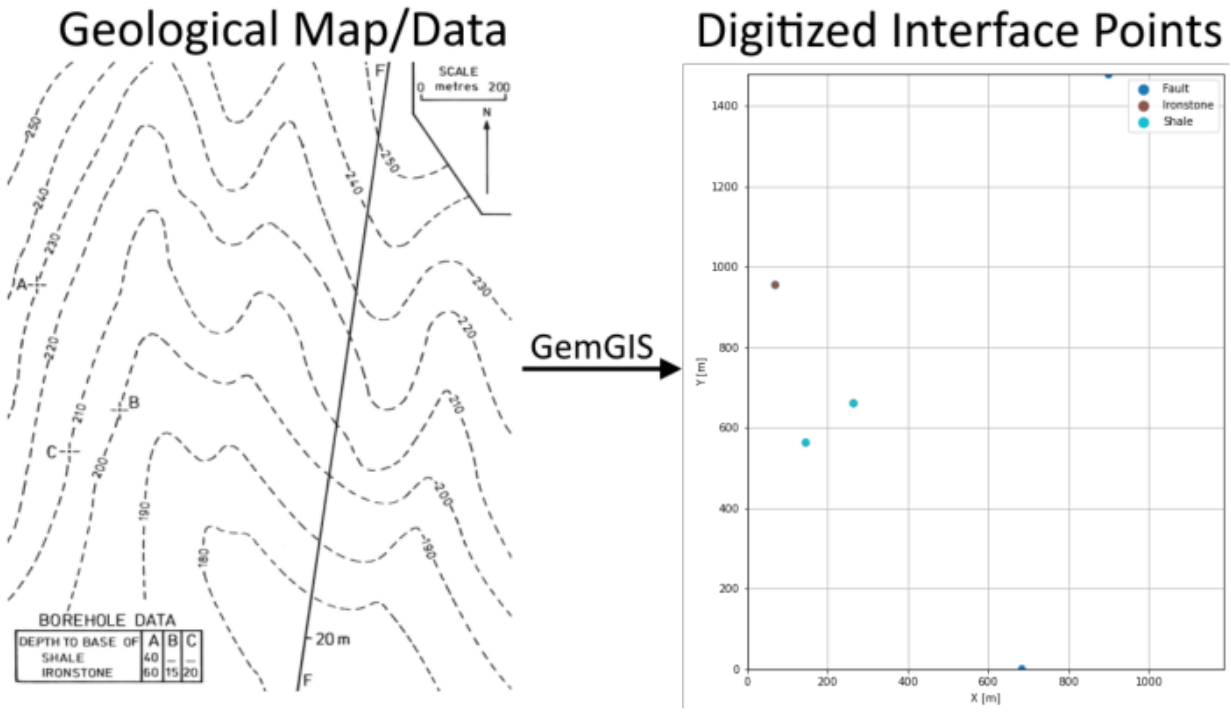
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster15.tif')
```

7.15.5 Interface Points of stratigraphic boundaries

The interface points for this three point example will be digitized as points with the respective height value as given by the borehole information and the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example15.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces15.shp')
interfaces.head()
```

[11]:

	id	formation	Z	geometry
0	None	Shale	190	POINT (69.806 954.941)
1	None	Ironstone	170	POINT (69.806 954.941)
2	None	Ironstone	190	POINT (145.769 562.774)
3	None	Ironstone	185	POINT (264.746 660.701)
4	None	Shale	210	POINT (146.226 563.346)

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=None)
interfaces_coords
```

[12]:

	formation	Z	geometry	X	Y
0	Shale	190.00	POINT (69.806 954.941)	69.81	954.94
1	Ironstone	170.00	POINT (69.806 954.941)	69.81	954.94
2	Ironstone	190.00	POINT (145.769 562.774)	145.77	562.77
3	Ironstone	185.00	POINT (264.746 660.701)	264.75	660.70
4	Shale	210.00	POINT (146.226 563.346)	146.23	563.35
5	Shale	205.00	POINT (264.746 660.701)	264.75	660.70

[13]:
```
fault = gpd.read_file(file_path + 'fault15.shp')
fault = gg.vector.extract_xyz(gdf=fault, dem=topo_raster)
fault
```

[13]:

	formation	geometry	X	Y	Z
0	Fault	POINT (683.911 0.608)	683.91	0.61	178.76
1	Fault	POINT (899.671 1477.524)	899.67	1477.52	254.62

[14]:
```
import pandas as pd

interfaces_coords = pd.concat([interfaces_coords, fault]).reset_index()
interfaces_coords
```

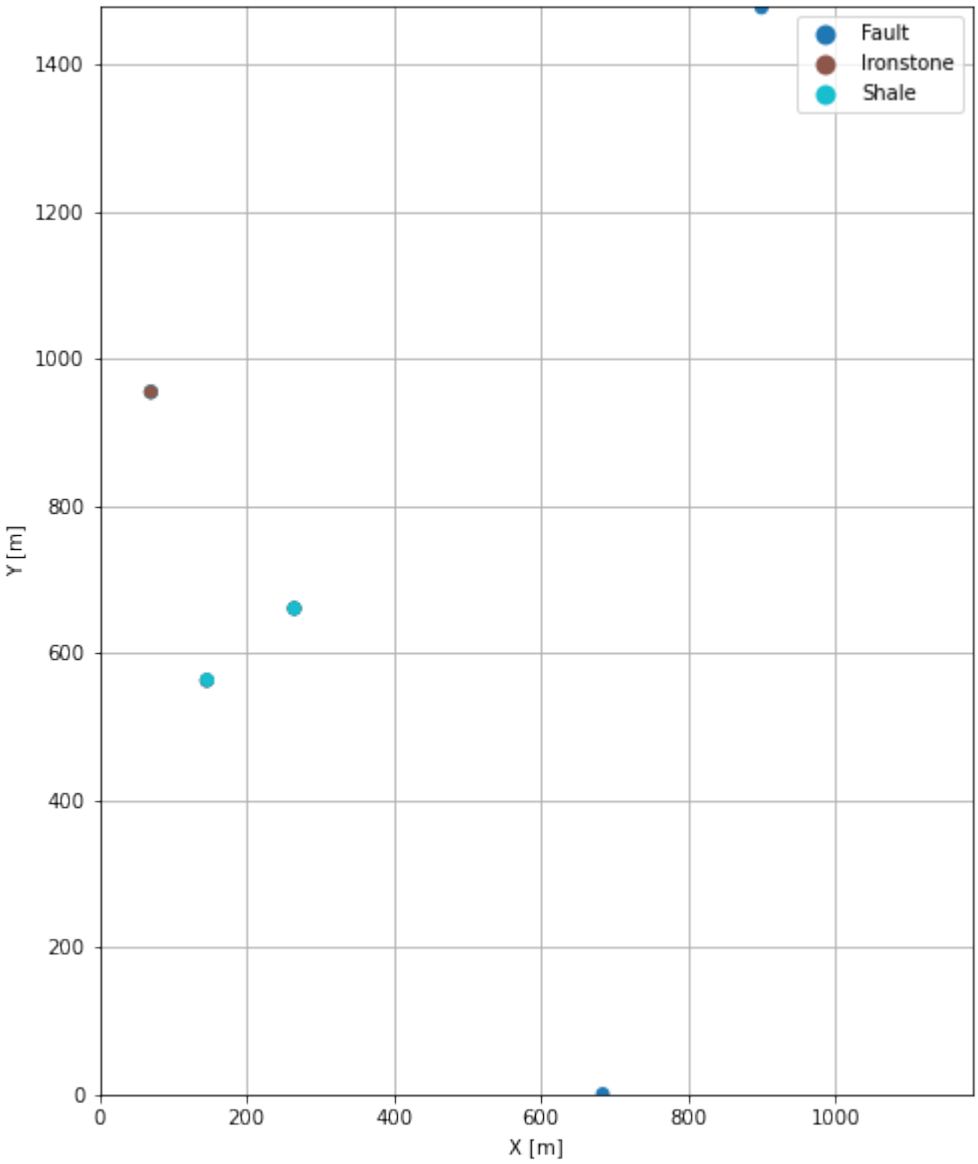
[14]:

	index	formation	Z	geometry	X	Y
0	0	Shale	190.00	POINT (69.806 954.941)	69.81	954.94
1	1	Ironstone	170.00	POINT (69.806 954.941)	69.81	954.94
2	2	Ironstone	190.00	POINT (145.769 562.774)	145.77	562.77
3	3	Ironstone	185.00	POINT (264.746 660.701)	264.75	660.70
4	4	Shale	210.00	POINT (146.226 563.346)	146.23	563.35
5	5	Shale	205.00	POINT (264.746 660.701)	264.75	660.70
6	0	Fault	178.76	POINT (683.911 0.608)	683.91	0.61
7	1	Fault	254.62	POINT (899.671 1477.524)	899.67	1477.52

Plotting the Interface Points

[15]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 1187)
ax.set_ylim(0, 1479)
```

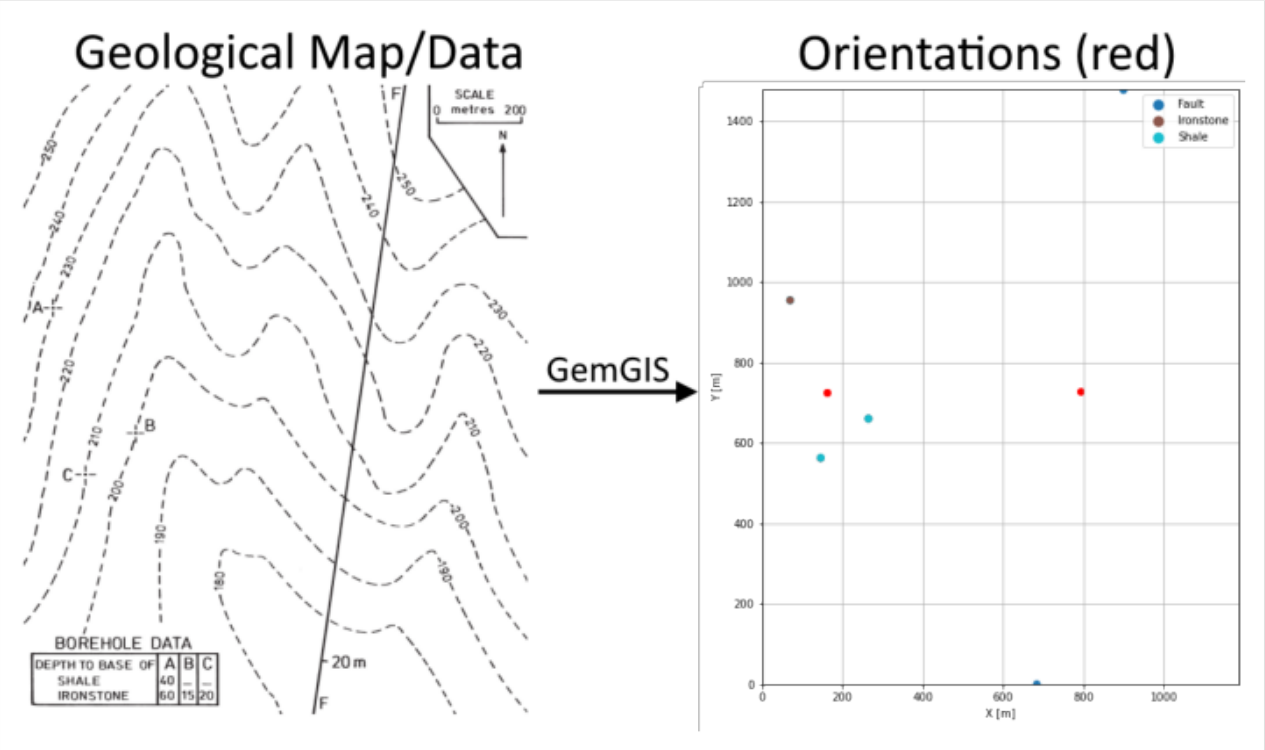
[15]:
```
(0.0, 1479.0)
```



7.15.6 Orientations from Strike Lines

For this three point example, an orientation is calculated using `gg.vector.calculate_orientation_for_three_point_problem()`.

[16]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example15.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[17]:
```
orientations1 = gpd.read_file(file_path + 'interfaces15.shp')
orientations1 = orientations1[orientations1['formation']=='Ironstone']
orientations1
```

[17]:
```
     id  formation    Z                 geometry
1  None  Ironstone  170   POINT (69.806 954.941)
2  None  Ironstone  190  POINT (145.769 562.774)
3  None  Ironstone  185  POINT (264.746 660.701)
```

[18]:
```
orientations1 = gg.vector.calculate_orientation_for_three_point_
↪problem(gdf=orientations1)
orientations1
```

[18]:
```
       Z  formation azimuth    dip polarity      X      Y  \
0 181.67  Ironstone -179.95 177.08        1 160.11 726.14

                  geometry
0  POINT (160.107 726.139)
```

Changing the Data Type of Fields

[19]:
```
orientations1['Z'] = orientations1['Z'].astype(float)
orientations1['azimuth'] = orientations1['azimuth'].astype(float)
orientations1['dip'] = orientations1['dip'].astype(float)
orientations1['dip'] = 180 - orientations1['dip']
orientations1['azimuth'] = 180 - orientations1['azimuth']
orientations1['polarity'] = orientations1['polarity'].astype(float)
orientations1['X'] = orientations1['X'].astype(float)
orientations1['Y'] = orientations1['Y'].astype(float)
orientations1.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Z          1 non-null      float64
 1   formation  1 non-null      object
 2   azimuth    1 non-null      float64
 3   dip        1 non-null      float64
 4   polarity   1 non-null      float64
 5   X          1 non-null      float64
 6   Y          1 non-null      float64
 7   geometry   1 non-null      geometry
dtypes: float64(6), geometry(1), object(1)
memory usage: 192.0+ bytes
```

[20]:
```
orientations2 = gpd.read_file(file_path + 'interfaces15.shp')
orientations2 = orientations2[orientations2['formation']=='Shale']
orientations2
```

[20]:
```
     id formation    Z                 geometry
0  None     Shale  190   POINT (69.806 954.941)
4  None     Shale  210  POINT (146.226 563.346)
5  None     Shale  205  POINT (264.746 660.701)
```

[21]:
```
orientations2 = gg.vector.calculate_orientation_for_three_point_
↪problem(gdf=orientations2)
orientations2
```

[21]:
```
       Z formation azimuth    dip polarity      X      Y  \
0 201.67     Shale -179.77 177.07        1 160.26 726.33

                   geometry
0  POINT (160.259 726.329)
```

Changing the Data Type of Fields

[22]:
```
orientations2['Z'] = orientations2['Z'].astype(float)
orientations2['azimuth'] = orientations2['azimuth'].astype(float)
orientations2['dip'] = orientations2['dip'].astype(float)
orientations2['dip'] = 180 - orientations2['dip']
orientations2['azimuth'] = 180 - orientations2['azimuth']
orientations2['polarity'] = orientations2['polarity'].astype(float)
orientations2['X'] = orientations2['X'].astype(float)
orientations2['Y'] = orientations2['Y'].astype(float)
orientations2.info()
```

```
<class 'geopandas.geodataframe.GeoDataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Z          1 non-null      float64
 1   formation  1 non-null      object
 2   azimuth    1 non-null      float64
 3   dip        1 non-null      float64
 4   polarity   1 non-null      float64
 5   X          1 non-null      float64
 6   Y          1 non-null      float64
 7   geometry   1 non-null      geometry
dtypes: float64(6), geometry(1), object(1)
memory usage: 192.0+ bytes
```

[23]:
```
orientations_fault = gpd.read_file(file_path + 'orientations15_fault.shp')
orientations_fault = gg.vector.extract_xyz(gdf=orientations_fault, dem=topo_raster)
orientations_fault
```

[23]:
```
  formation   dip  azimuth  polarity                 geometry      X      Y  \
0     Fault 90.00   280.00      1.00  POINT (792.591 727.511) 792.59 727.51

       Z
0 215.87
```

Merging Orientations

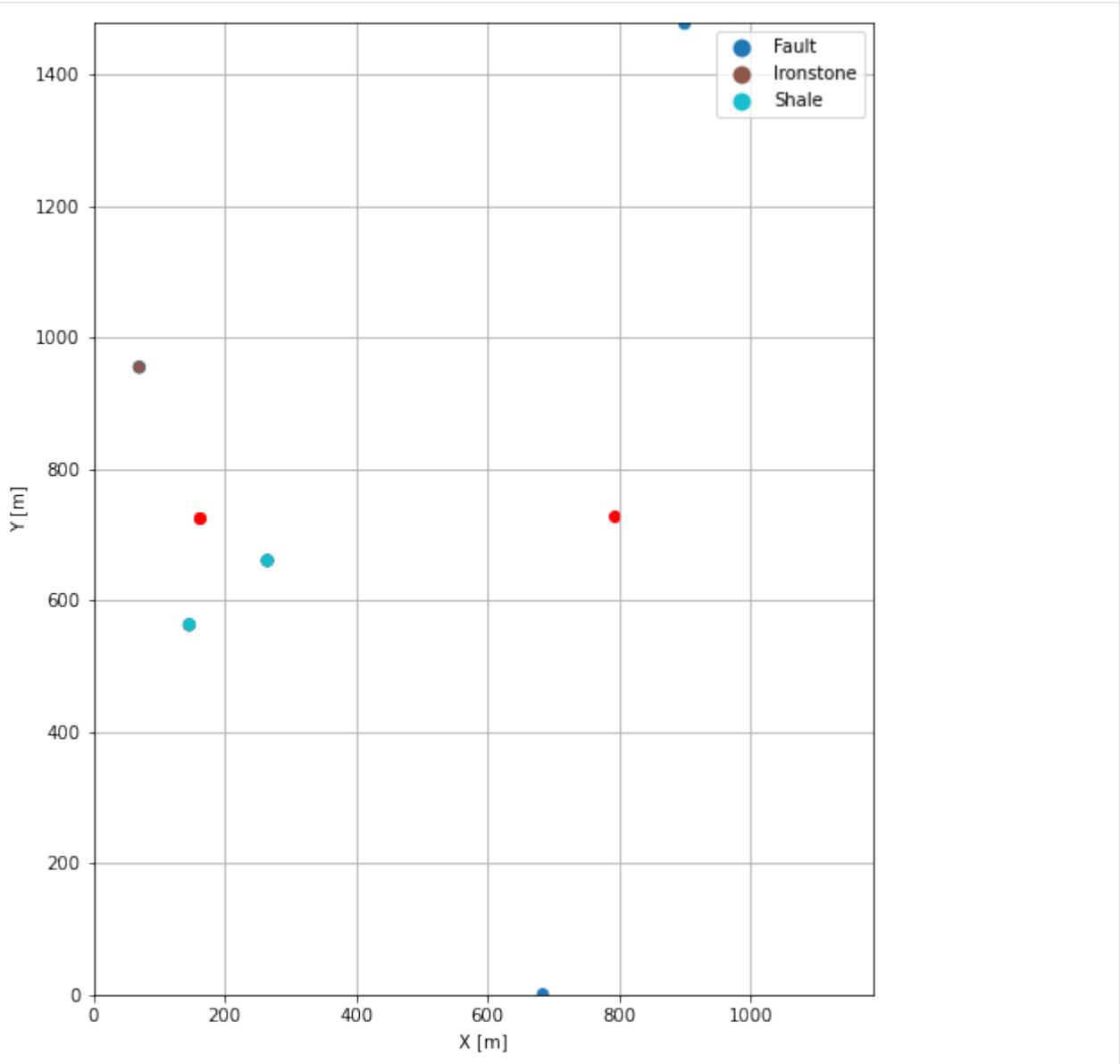
[24]:
```
orientations = pd.concat([orientations1, orientations2, orientations_fault]).reset_
↪index()
orientations
```

[24]:
```
   index      Z  formation  azimuth   dip  polarity      X      Y  \
0      0 181.67  Ironstone   359.95  2.92      1.00 160.11 726.14
1      0 201.67      Shale   359.77  2.93      1.00 160.26 726.33
2      0 215.87      Fault   280.00 90.00      1.00 792.59 727.51

                   geometry
0  POINT (160.107 726.139)
1  POINT (160.259 726.329)
2  POINT (792.591 727.511)
```

Plotting the Orientations

[25]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlim(0,1187)
plt.ylim(0,1479)
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
```

[25]:
```
Text(179.77829589465532, 0.5, 'Y [m]')
```



7.15.7 GemPy Model Construction (Part A)

The structural geological model will be constructed using the GemPy package. The first model is calculated without the fault.

[26]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating New Model

[27]:
```
geo_model = gp.create_model('Model15a')
geo_model
```

[27]:
```
Model15a  2022-04-05 11:07
```

Initiate Data

The fault interfaces and orientations will be removed from the first model to model the layers also beyond the fault. As the information is provided that the fault has an offset of 20 m, the layers will be exported, the boundaries will be digitized and the elevation will be reduced by 20 m.

[28]:
```
interfaces_coords_new=interfaces_coords[interfaces_coords['formation'] != 'Fault']
interfaces_coords_new
```

[28]:

	index	formation	Z	geometry	X	Y
0	0	Shale	190.00	POINT (69.806 954.941)	69.81	954.94
1	1	Ironstone	170.00	POINT (69.806 954.941)	69.81	954.94
2	2	Ironstone	190.00	POINT (145.769 562.774)	145.77	562.77
3	3	Ironstone	185.00	POINT (264.746 660.701)	264.75	660.70
4	4	Shale	210.00	POINT (146.226 563.346)	146.23	563.35
5	5	Shale	205.00	POINT (264.746 660.701)	264.75	660.70

[29]:
```
orientations_new=orientations[orientations['formation'] != 'Fault']
orientations_new
```

[29]:

	index	Z	formation	azimuth	dip	polarity	X	Y	\
0	0	181.67	Ironstone	359.95	2.92	1.00	160.11	726.14	
1	0	201.67	Shale	359.77	2.93	1.00	160.26	726.33	

	geometry
0	POINT (160.107 726.139)
1	POINT (160.259 726.329)

[30]:
```
gp.init_data(geo_model, [0, 1187, 0, 1479, 100, 300], [100, 100, 100],
             surface_points_df=interfaces_coords_new[interfaces_coords_new['Z'] != 0],
```

(continues on next page)

(continued from previous page)

```
            orientations_df=orientations_new,
            default_values=True)
```

```
Active grids: ['regular']
```

[30]:
```
Model15a  2022-04-05 11:07
```

Model Surfaces

[31]:
```
geo_model.surfaces
```

[31]:
```
     surface          series  order_surfaces    color  id
0      Shale  Default series               1  #015482   1
1  Ironstone  Default series               2  #9f0052   2
```

Mapping the Stack to Surfaces

[32]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Shale', 'Ironstone'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[32]:
```
     surface   series  order_surfaces    color  id
0      Shale  Strata1               1  #015482   1
1  Ironstone  Strata1               2  #9f0052   2
2   Basement  Strata1               3  #ffbe00   3
```

Showing the Number of Data Points

[33]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[33]:
```
     surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0      Shale  Strata1               1  #015482   1                  3                ␣
↪1
1  Ironstone  Strata1               2  #9f0052   2                  3                ␣
↪1
2   Basement  Strata1               3  #ffbe00   3                  0                ␣
↪0
```

Loading Digital Elevation Model

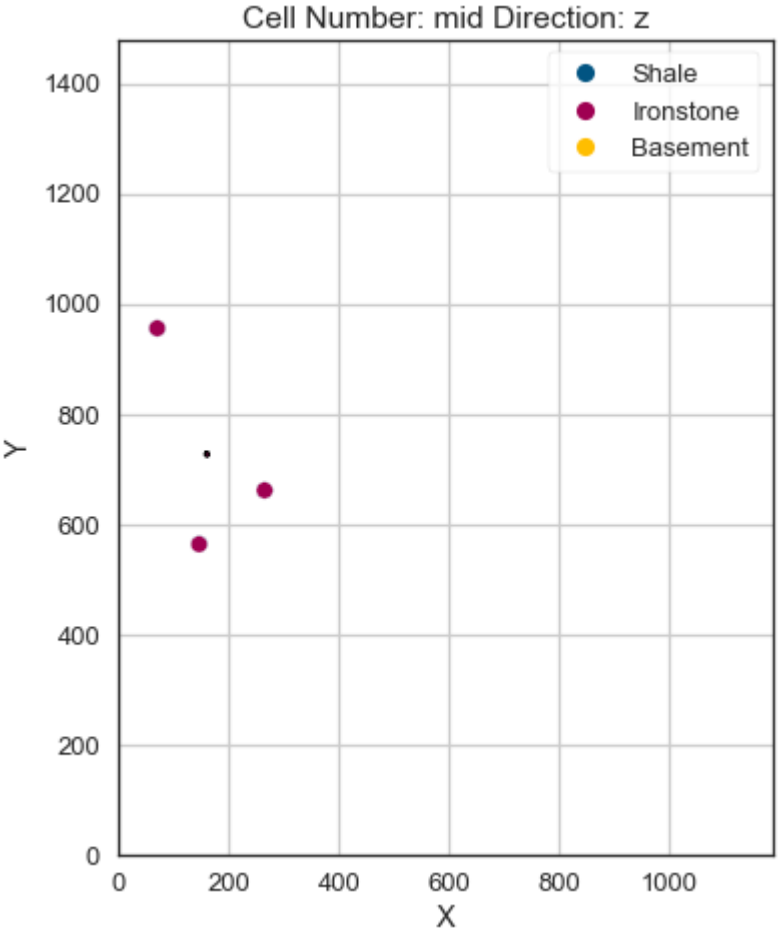
[34]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster15.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

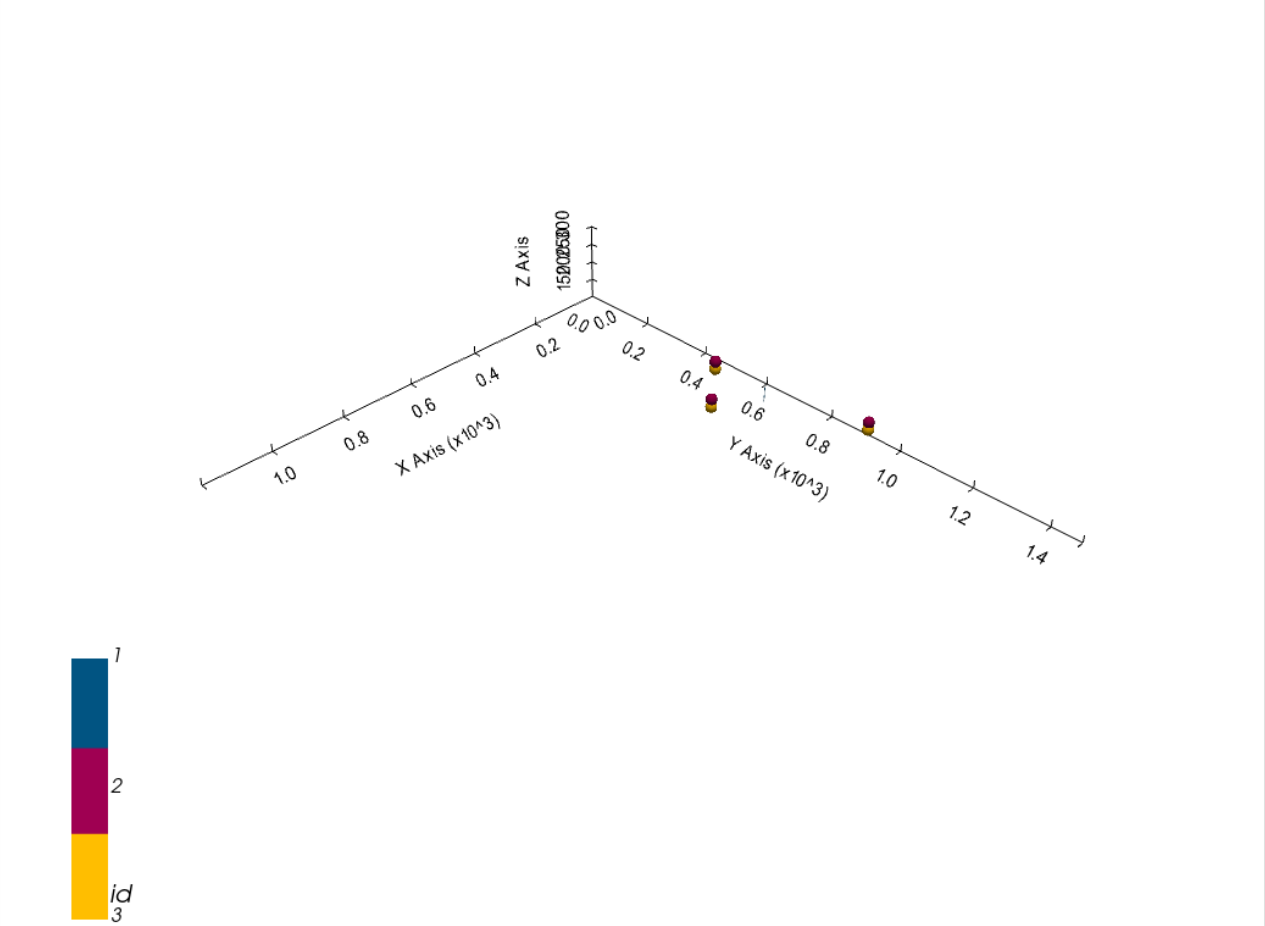
[34]:
```
Grid Object. Values:
array([[   5.935     ,    7.395     ,  101.        ],
       [   5.935     ,    7.395     ,  103.        ],
       [   5.935     ,    7.395     ,  105.        ],
       ...,
       [1184.49578059, 1466.50844595,  275.3008728 ],
       [1184.49578059, 1471.50506757,  275.80532837],
       [1184.49578059, 1476.50168919,  276.31124878]])
```

Plotting Input Data

[35]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[36]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[36]:
```
<gempy.plot.vista.GemPyToVista at 0x1e68be5d250>
```

Setting the Interpolator

[37]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range            1906.94
$C_o$           86581.19
drift equations      [3]
```

[37]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1e685ddafd0>
```

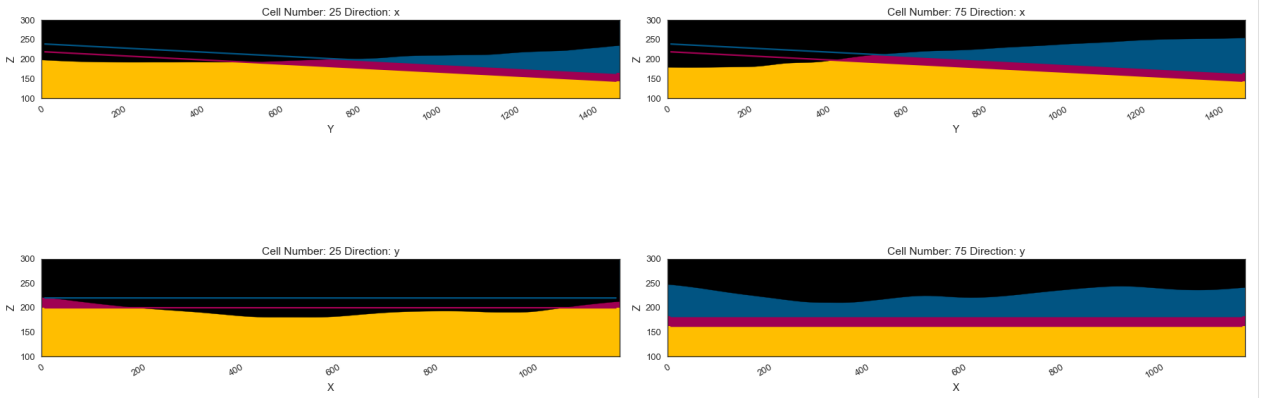
Computing Model

[38]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

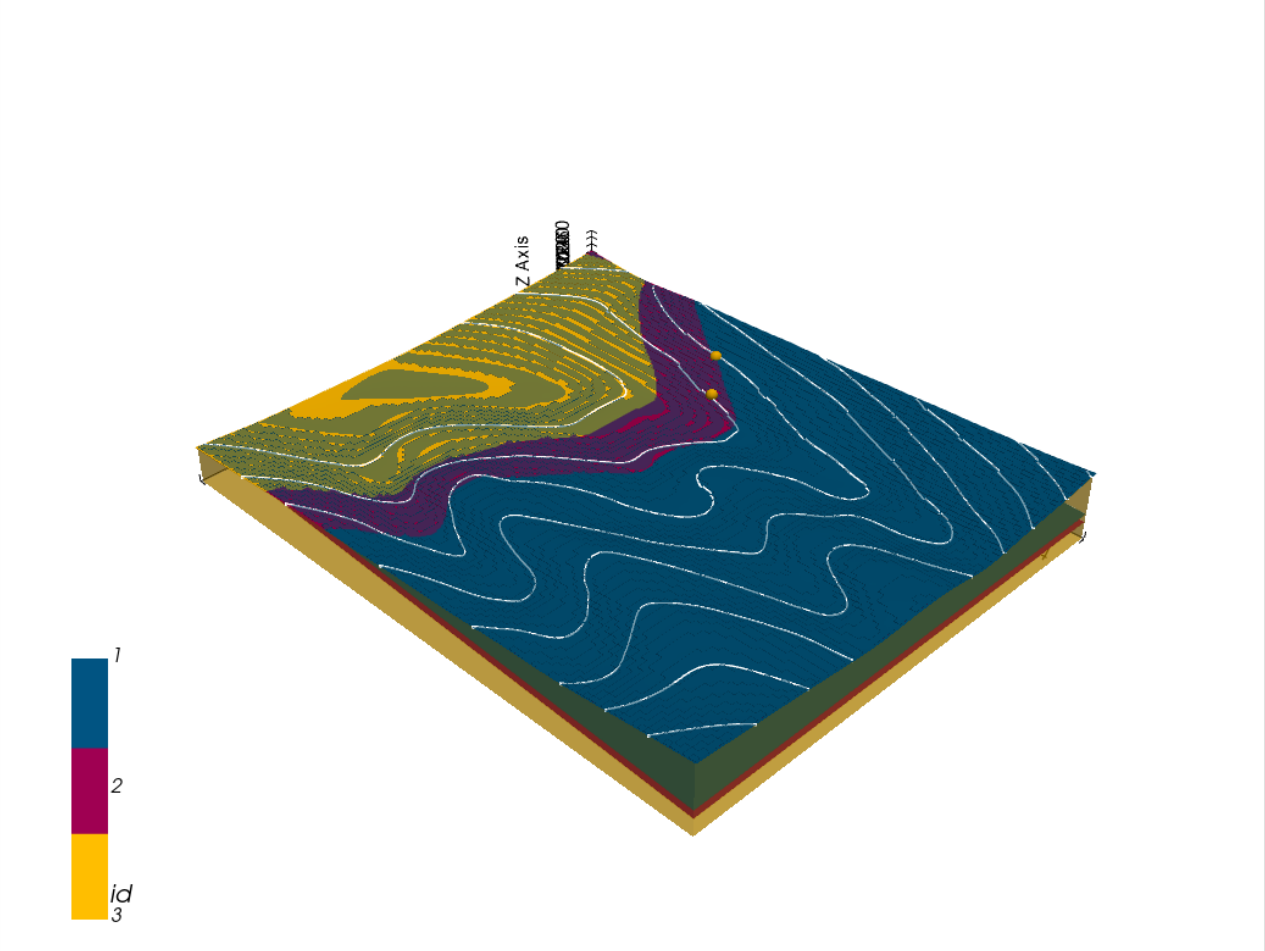
[39]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[39]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1e68a3e4a90>
```



Plotting 3D Model

[40]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



Creating Polygons from GemPy Model

A GeoDataFrame containing polygons representing the geological map can be created using `gg.post.extract_lithologies()`. This data is now being saved and the constructed layer boundaries beyond the fault in the east are being digitized. Their elevation values will be reduced by 20 m to simulate the offset of the fault. The model will then be recalculated again.

[41]:
```
gdf = gg.post.extract_lithologies(geo_model, [0, 1187, 0, 1479], 'EPSG:4326')
gdf
```

[41]:
```
   formation                                           geometry
0   Basement  POLYGON ((22.538 3.254, 22.804 2.498, 27.546 2...
1  Ironstone  POLYGON ((7.513 2.498, 12.521 2.498, 17.530 2...
2  Ironstone  POLYGON ((5.851 192.370, 7.513 194.147, 10.887...
3      Shale  POLYGON ((4.134 362.255, 7.513 365.081, 10.021...
```

[42]:
```
gdf = gpd.read_file(file_path + 'geolmap.shp')
gdf
```

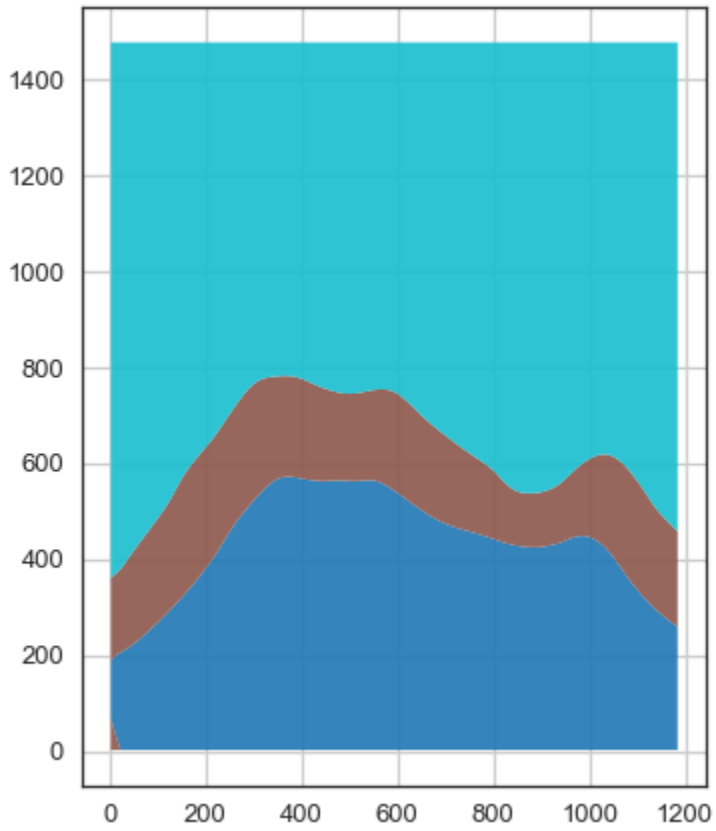
[42]:
```
       formation                                           geometry
0  Conglomerate  POLYGON ((22.538 3.254, 21.064 7.495, 19.356 1...
1     Ironstone  POLYGON ((7.513 2.498, 2.504 2.498, 2.504 7.49...
```

(continues on next page)

(continued from previous page)

```
2     Ironstone  POLYGON ((5.851 192.370, 2.504 188.069, 2.504 ...
3         Shale  POLYGON ((4.134 362.255, 2.504 360.939, 2.504 ...
```

[43]:
```
gdf.plot(column='formation',  alpha=0.9, aspect='equal')
plt.grid()
```



Preparing Interfaces beyond the fault

[44]:
```
interfaces_beyond_fault =  gpd.read_file(file_path + 'interfaces15_beyond_fault.shp')
interfaces_beyond_fault = gg.vector.extract_xyz(gdf=interfaces_beyond_fault, dem=topo_
↪raster)
interfaces_beyond_fault
```

[44]:
```
     formation                  geometry       X      Y      Z
0        Shale   POINT (778.812 601.522)  778.81 601.52 208.06
1        Shale   POINT (783.821 597.991)  783.82 597.99 208.15
2        Shale   POINT (788.829 594.245)  788.83 594.25 208.26
3        Shale   POINT (791.557 592.100)  791.56 592.10 208.71
4        Shale   POINT (797.430 587.103)  797.43 587.10 208.89
..         ...                      ...     ...    ...    ...
250  Ironstone  POINT (1169.470 270.001) 1169.47 270.00 205.13
251  Ironstone  POINT (1172.985 267.319) 1172.99 267.32 205.24
252  Ironstone  POINT (1174.479 266.187) 1174.48 266.19 205.24
253  Ironstone  POINT (1179.487 262.439) 1179.49 262.44 205.33
```

(continues on next page)

(continued from previous page)

```
254  Ironstone  POINT (1184.496 258.751) 1184.50 258.75 205.42

[255 rows x 5 columns]
```

Substracting the fault offset

[45]:
```
interfaces_beyond_fault['Z']=interfaces_beyond_fault['Z']-20
interfaces_beyond_fault
```

[45]:
```
     formation                  geometry       X      Y      Z
0        Shale   POINT (778.812 601.522)  778.81 601.52 188.06
1        Shale   POINT (783.821 597.991)  783.82 597.99 188.15
2        Shale   POINT (788.829 594.245)  788.83 594.25 188.26
3        Shale   POINT (791.557 592.100)  791.56 592.10 188.71
4        Shale   POINT (797.430 587.103)  797.43 587.10 188.89
..         ...                      ...     ...    ...    ...
250  Ironstone  POINT (1169.470 270.001) 1169.47 270.00 185.13
251  Ironstone  POINT (1172.985 267.319) 1172.99 267.32 185.24
252  Ironstone  POINT (1174.479 266.187) 1174.48 266.19 185.24
253  Ironstone  POINT (1179.487 262.439) 1179.49 262.44 185.33
254  Ironstone  POINT (1184.496 258.751) 1184.50 258.75 185.42

[255 rows x 5 columns]
```

Mergin old and new interfaces

[46]:
```
interfaces_coords = pd.concat([interfaces_coords, interfaces_beyond_fault.sample(n=50)]).
↪reset_index()
interfaces_coords
```

[46]:
```
    level_0  index  formation      Z                 geometry       X       Y
0         0   0.00      Shale 190.00    POINT (69.806 954.941)   69.81  954.94
1         1   1.00  Ironstone 170.00    POINT (69.806 954.941)   69.81  954.94
2         2   2.00  Ironstone 190.00   POINT (145.769 562.774)  145.77  562.77
3         3   3.00  Ironstone 185.00   POINT (264.746 660.701)  264.75  660.70
4         4   4.00      Shale 210.00   POINT (146.226 563.346)  146.23  563.35
5         5   5.00      Shale 205.00   POINT (264.746 660.701)  264.75  660.70
6         6   0.00      Fault 178.76     POINT (683.911 0.608)  683.91    0.61
7         7   1.00      Fault 254.62  POINT (899.671 1477.524)  899.67 1477.52
8       177    NaN  Ironstone 176.27   POINT (949.099 437.906)  949.10  437.91
9       199    NaN  Ironstone 177.36  POINT (1039.795 417.218) 1039.79  417.22
10      156    NaN  Ironstone 176.90   POINT (853.939 427.352)  853.94  427.35
11      127    NaN      Shale 194.60  POINT (1169.470 472.641) 1169.47  472.64
12        4    NaN      Shale 188.89   POINT (797.430 587.103)  797.43  587.10
13      230    NaN  Ironstone 182.46  POINT (1114.378 320.613) 1114.38  320.61
14      155    NaN  Ironstone 176.77   POINT (848.930 428.164)  848.93  428.16
15      196    NaN  Ironstone 177.05  POINT (1034.243 423.251) 1034.24  423.25
16      110    NaN      Shale 191.82  POINT (1124.395 527.284) 1124.39  527.28
17        0    NaN      Shale 188.06   POINT (778.812 601.522)  778.81  601.52
18       20    NaN      Shale 190.81   POINT (838.823 547.130)  838.82  547.13
```

(continues on next page)

(continued from previous page)

```
19       74    NaN      Shale 187.12  POINT (1034.243 618.288) 1034.24  618.29
20       92    NaN      Shale 188.89  POINT (1085.673 582.106) 1085.67  582.11
21      138    NaN  Ironstone 175.84   POINT (783.821 447.680)  783.82  447.68
22      119    NaN      Shale 193.19  POINT (1144.428 499.461) 1144.43  499.46
23       94    NaN      Shale 189.24  POINT (1089.653 577.110) 1089.65  577.11
24       77    NaN      Shale 187.33  POINT (1044.259 616.032) 1044.26  616.03
25      123    NaN      Shale 193.80  POINT (1154.445 488.152) 1154.45  488.15
26       53    NaN      Shale 189.23   POINT (965.782 582.106)  965.78  582.11
27        8    NaN      Shale 189.40   POINT (807.567 577.110)  807.57  577.11
28      194    NaN  Ironstone 176.60  POINT (1024.226 432.683) 1024.23  432.68
29      160    NaN  Ironstone 177.25   POINT (868.964 425.396)  868.96  425.40
30      181    NaN  Ironstone 176.12   POINT (969.133 446.025)  969.13  446.03
31       55    NaN      Shale 189.01   POINT (971.197 587.103)  971.20  587.10
32      233    NaN  Ironstone 183.05  POINT (1122.155 312.289) 1122.16  312.29
33      241    NaN  Ironstone 184.02  POINT (1142.482 292.302) 1142.48  292.30
34      144    NaN  Ironstone 176.33   POINT (803.854 440.894)  803.85  440.89
35      252    NaN  Ironstone 185.24  POINT (1174.479 266.187) 1174.48  266.19
36      152    NaN  Ironstone 176.80   POINT (833.905 431.359)  833.91  431.36
37        3    NaN      Shale 188.71   POINT (791.557 592.100)  791.56  592.10
38      129    NaN      Shale 194.85  POINT (1174.933 467.184) 1174.93  467.18
39       46    NaN      Shale 190.13   POINT (944.754 562.120)  944.75  562.12
40      169    NaN  Ironstone 176.79   POINT (914.040 427.799)  914.04  427.80
41      211    NaN  Ironstone 179.46  POINT (1069.302 377.432) 1069.30  377.43
42      186    NaN  Ironstone 175.71   POINT (989.167 448.328)  989.17  448.33
43      206    NaN  Ironstone 178.75  POINT (1059.017 392.235) 1059.02  392.23
44      149    NaN  Ironstone 176.38   POINT (823.888 434.155)  823.89  434.15
45      212    NaN  Ironstone 179.93  POINT (1072.891 372.248) 1072.89  372.25
46      225    NaN  Ironstone 181.85  POINT (1103.757 332.275) 1103.76  332.28
47      187    NaN  Ironstone 175.83   POINT (994.175 447.621)  994.18  447.62
48       42    NaN      Shale 190.77   POINT (931.344 552.127)  931.34  552.13
49      109    NaN      Shale 191.38  POINT (1121.307 532.140) 1121.31  532.14
50      108    NaN      Shale 191.45  POINT (1119.386 535.097) 1119.39  535.10
51       38    NaN      Shale 191.03   POINT (919.049 545.720)  919.05  545.72
52      192    NaN  Ironstone 176.51  POINT (1018.269 437.204) 1018.27  437.20
53      121    NaN      Shale 193.50  POINT (1149.437 493.655) 1149.44  493.66
54      254    NaN  Ironstone 185.42  POINT (1184.496 258.751) 1184.50  258.75
55      222    NaN  Ironstone 181.17  POINT (1094.344 343.135) 1094.34  343.14
56      184    NaN  Ironstone 175.73   POINT (979.150 448.200)  979.15  448.20
57      209    NaN  Ironstone 178.98  POINT (1064.293 384.655) 1064.29  384.65
```

7.15.8 GemPy Model Construction (Part B)

The structural geological model will be constructed using the `GemPy` package.

[47]:
```
import gempy as gp
```

Creating new Model

[48]:
```
geo_model = gp.create_model('Model15b')
geo_model
```

[48]:
```
Model15b  2022-04-05 11:08
```

Initiate Data

[49]:
```
gp.init_data(geo_model, [0, 1187, 0, 1479, 100, 300], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[49]:
```
Model15b  2022-04-05 11:08
```

Model Surfaces

[50]:
```
geo_model.surfaces
```

[50]:

	surface	series	order_surfaces	color	id
0	Shale	Default series	1	#015482	1
1	Ironstone	Default series	2	#9f0052	2
2	Fault	Default series	3	#ffbe00	3

Mapping the Stack to Surfaces

[51]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1' : ('Fault'),
                          'Strata1': ('Shale', 'Ironstone'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_is_fault(['Fault1'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[51]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault1	1	Fault	True	True	False
Strata1	2	Erosion	True	False	False

Showing the Number of Data Points

[52]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[52]:
```
     surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
2      Fault   Fault1               1  #527682   1                  2                ␣
↪1
0      Shale  Strata1               1  #9f0052   2                 25                ␣
↪1
1  Ironstone  Strata1               2  #ffbe00   3                 31                ␣
↪1
3   Basement  Strata1               3  #728f02   4                  0                ␣
↪0
```

Loading Digital Elevation Model

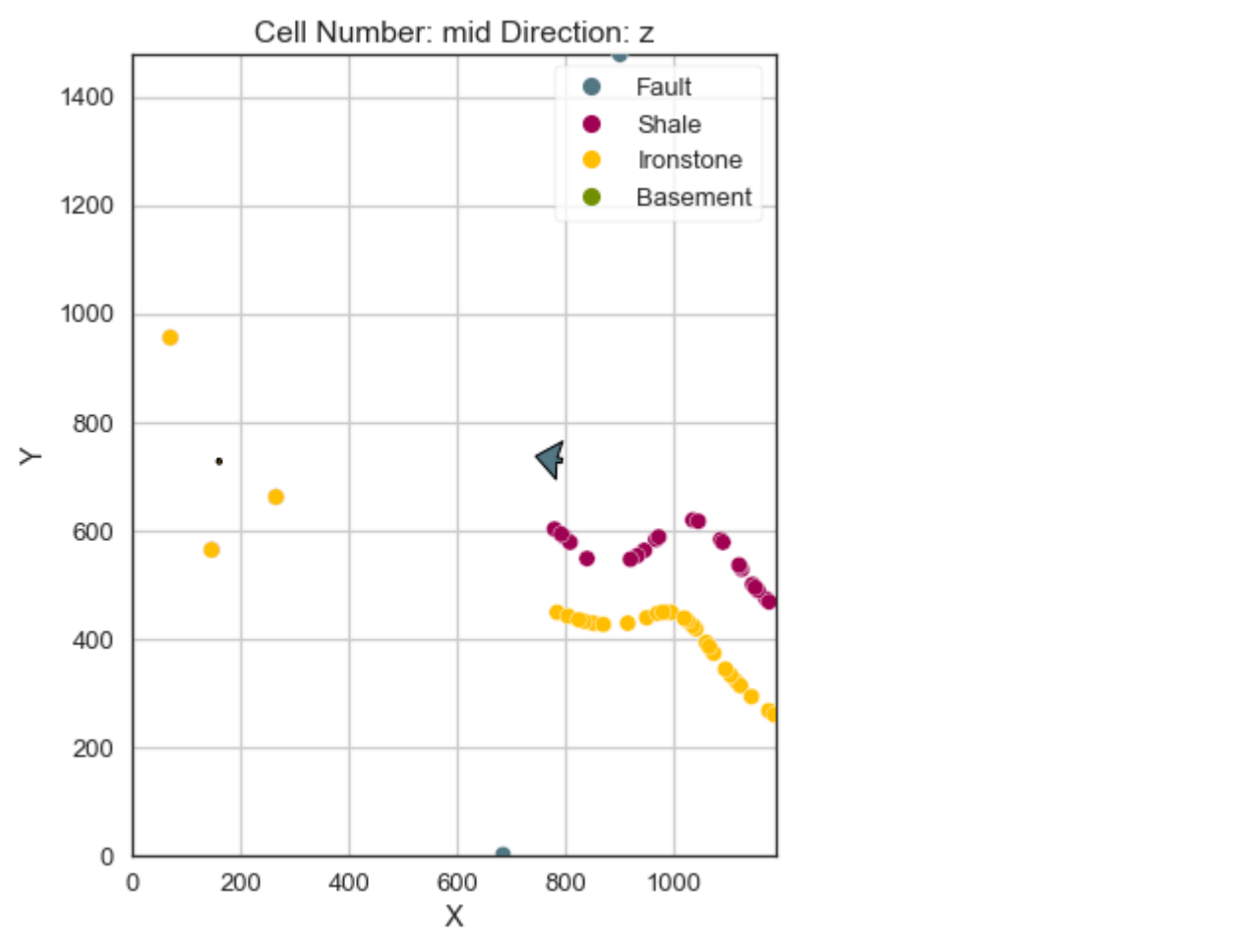
[53]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster15.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

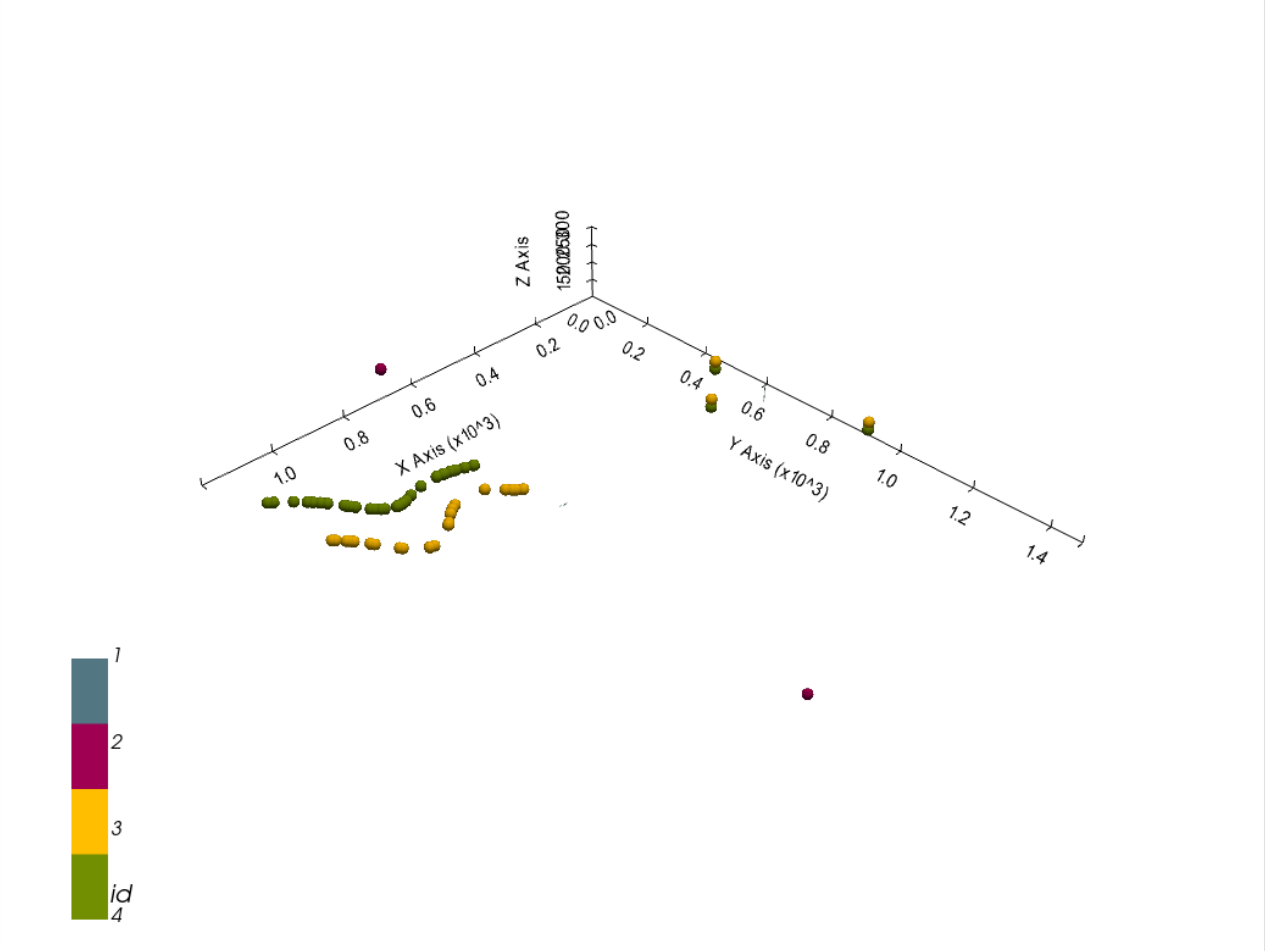
[53]:
```
Grid Object. Values:
array([[   5.935     ,    7.395     ,  101.        ],
       [   5.935     ,    7.395     ,  103.        ],
       [   5.935     ,    7.395     ,  105.        ],
       ...,
       [1184.49578059, 1466.50844595,  275.3008728 ],
       [1184.49578059, 1471.50506757,  275.80532837],
       [1184.49578059, 1476.50168919,  276.31124878]])
```

Plotting Input Data

[54]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[55]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[55]:
```
<gempy.plot.vista.GemPyToVista at 0x1e6886403d0>
```

Setting the Interpolator

[56]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                   values
range             1906.94
$C_o$            86581.19
drift equations    [3, 3]
```

[56]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1e687f7d370>
```

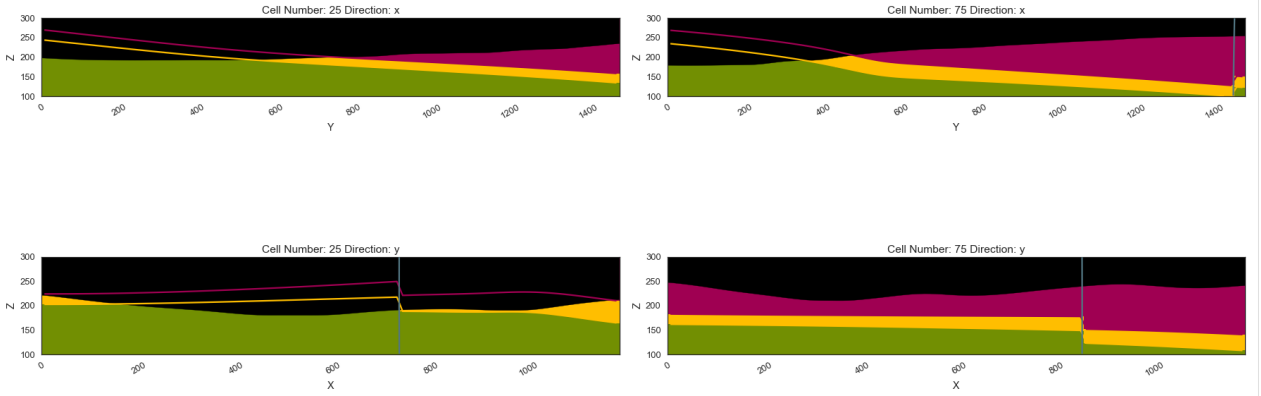
Computing Model

[57]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

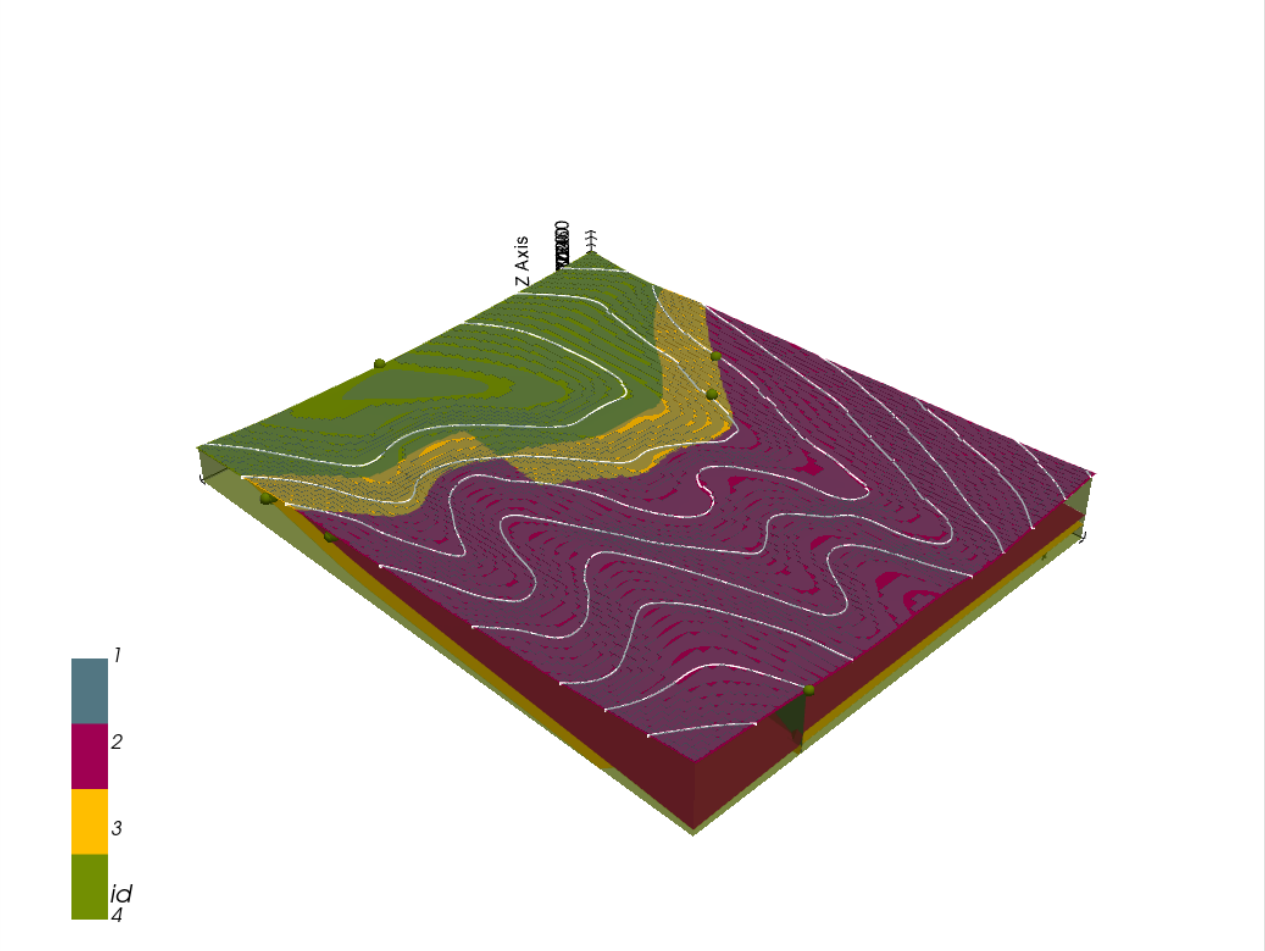
[58]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[58]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1e6b7bacbb0>
```



Plotting 3D Model

[59]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.16 Example 16 - Unconformal Faulted Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3968 m wide (W-E extent) and 2731 m high (N-S extent). The model represents folded and faulted layers (red, yellow, green) as well as layers separated by an unconfirmity (light red to light green) above an unspecified basement (light blue). The vertical model extent varies between 0 m and 1000 m.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

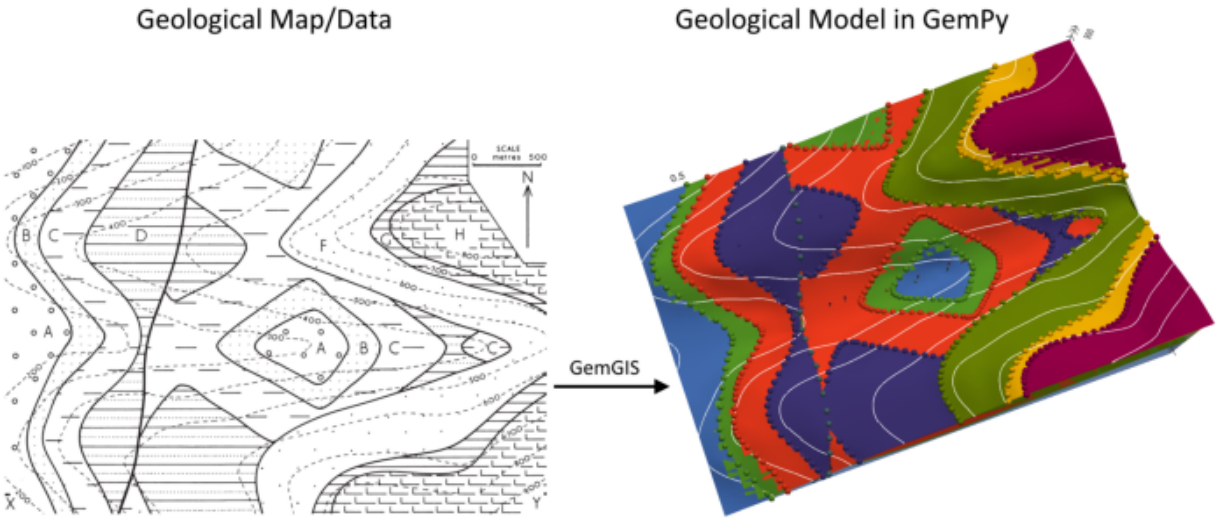
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example16.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.16.1 Licensing

7.16.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.16.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

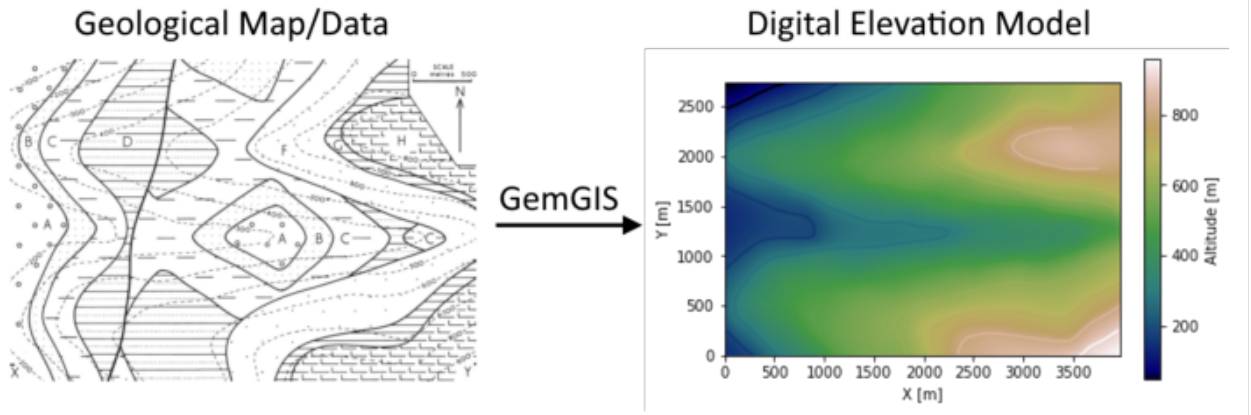
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example16/'
gg.download_gemgis_data.download_tutorial_data(filename="example16_all_features.zip",␣
↪dirpath=file_path)
```

7.16.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example16.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo16.shp')
topo.head()
```

[6]:
```
     id    Z                                           geometry
0  None  200  LINESTRING (1.591 265.171, 32.840 230.967, 79...
1  None  200  LINESTRING (2.436 851.918, 33.473 889.712, 73...
2  None  100  LINESTRING (1.803 2469.855, 62.188 2493.502, 1...
3  None  200  LINESTRING (2.014 2215.436, 17.849 2238.028, 6...
4  None  900  LINESTRING (3559.448 2.095, 3595.764 16.875, 3...
```

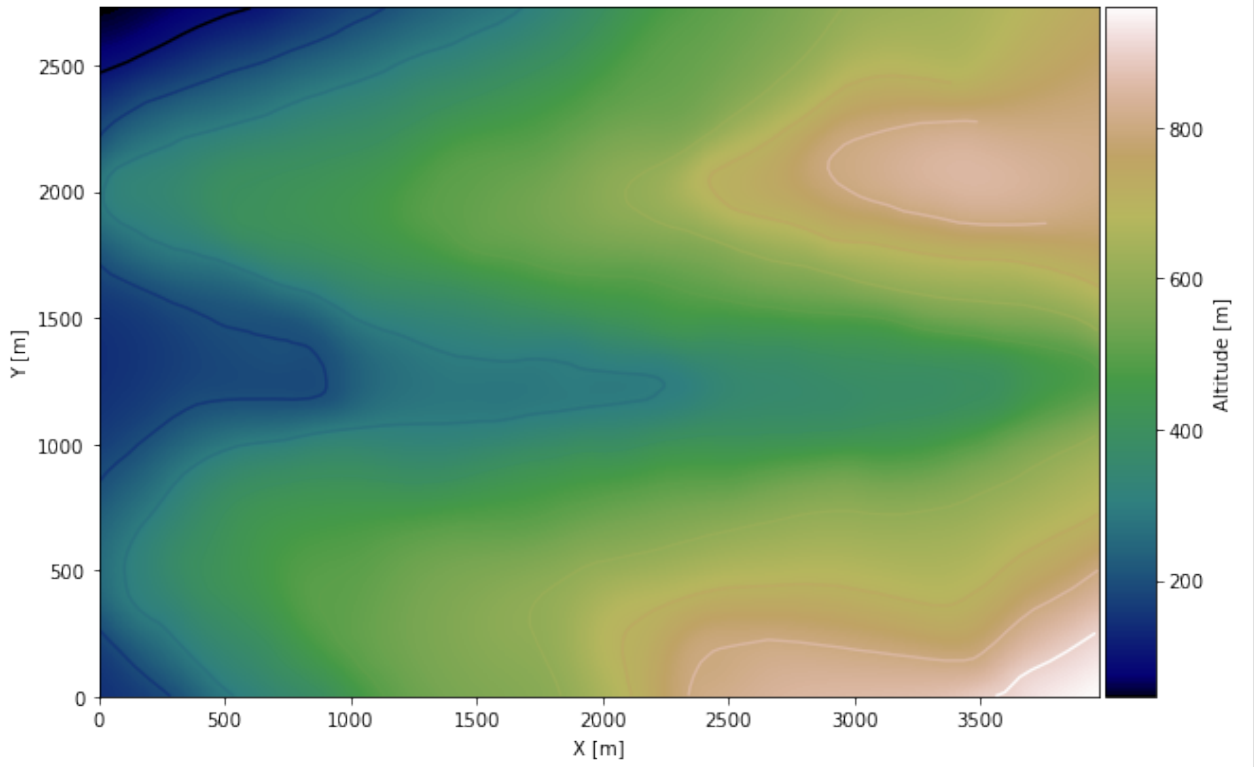
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable
fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 3968, 0, 2731], cmap='gist_earth
↪')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3968)
ax.set_ylim(0, 2731)
```

[8]:
```
(0.0, 2731.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

$\text{gg.raster.save}_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster16.tif', extent = [0, 3968, 0, 2731], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

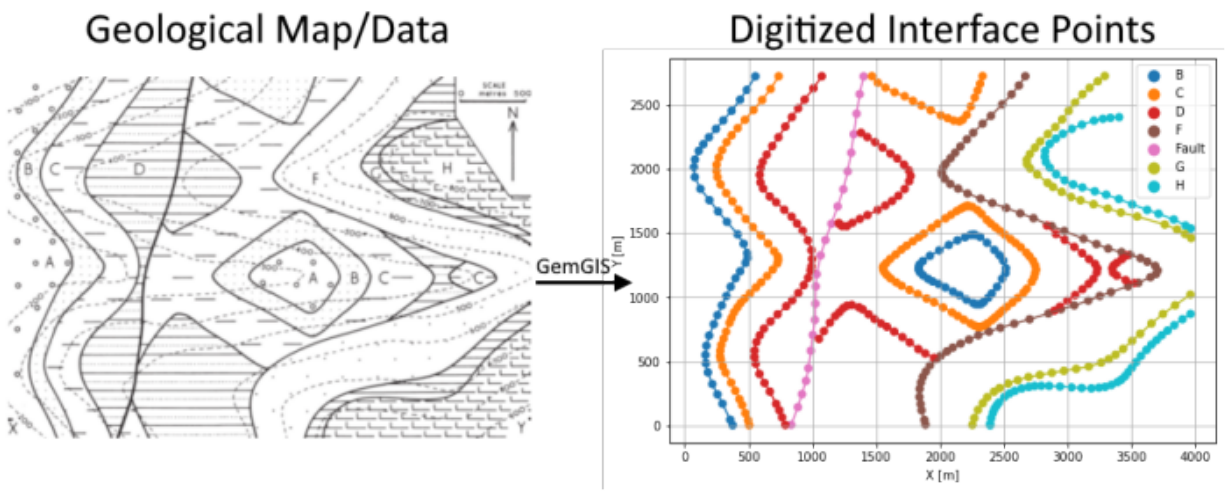
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster16.tif')
```

7.16.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example16.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces16.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Fault	LINESTRING (1401.636 2725.330, 1370.388 2586.8...
1	None	H	LINESTRING (3403.207 2406.937, 3334.799 2401.8...
2	None	G	LINESTRING (3290.883 2728.708, 3235.987 2687.3...

(continues on next page)

(continued from previous page)

```
3  None         D  LINESTRING (3461.481 1326.764, 3425.165 1298.0...
4  None         D  LINESTRING (2837.362 1560.703, 2908.304 1494.8...
```

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Fault', 'B',
↪'C', 'D', 'G', 'F', 'H'])]
interfaces_coords.head()
```

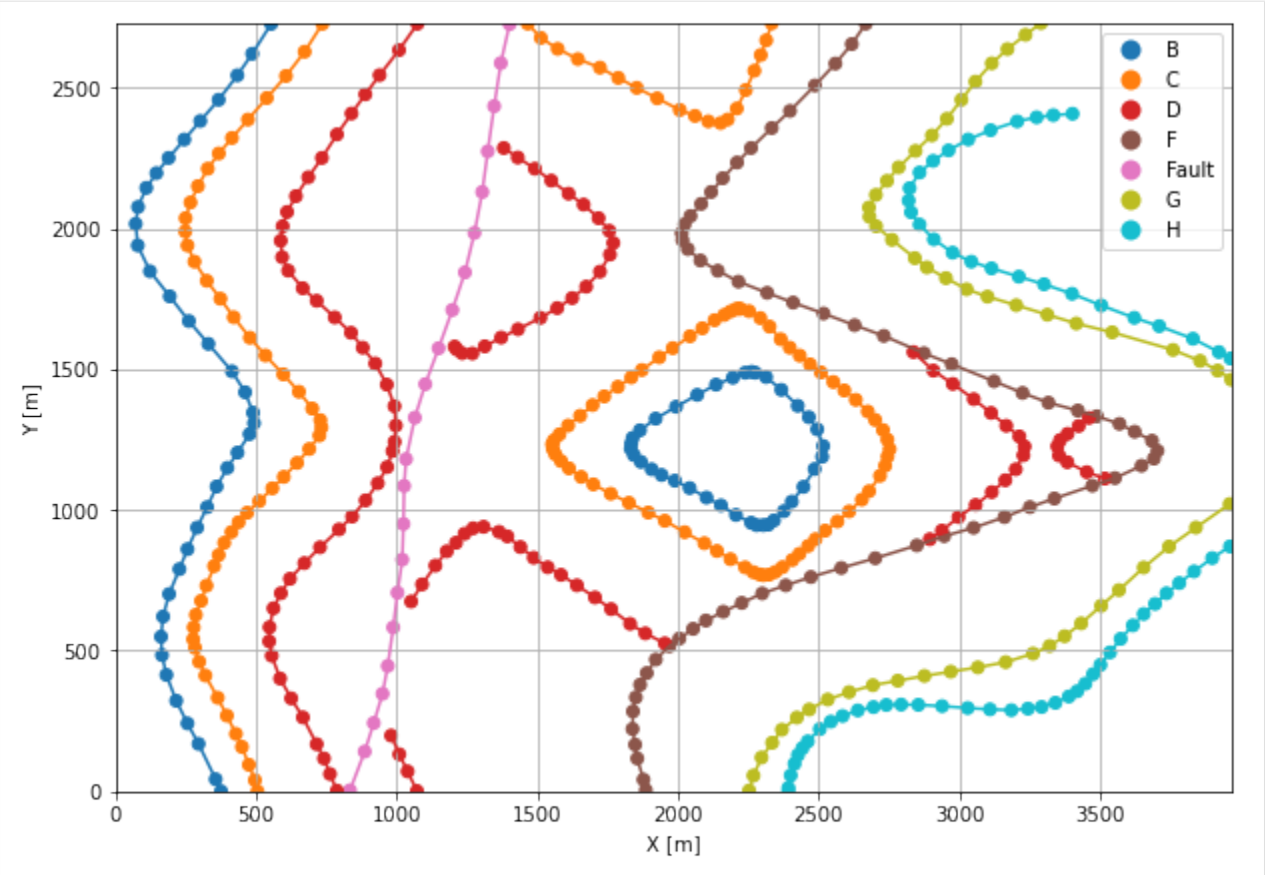
[12]:

	formation	geometry	X	Y	Z
351	H	POINT (3476.260 416.767)	3476.26	416.77	699.70
44	H	POINT (3711.466 1652.758)	3711.47	1652.76	664.07
359	H	POINT (3109.728 291.774)	3109.73	291.77	747.18
358	H	POINT (3185.737 288.818)	3185.74	288.82	745.14
357	H	POINT (3244.855 294.308)	3244.86	294.31	734.68

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3968)
ax.set_ylim(0, 2731)
```

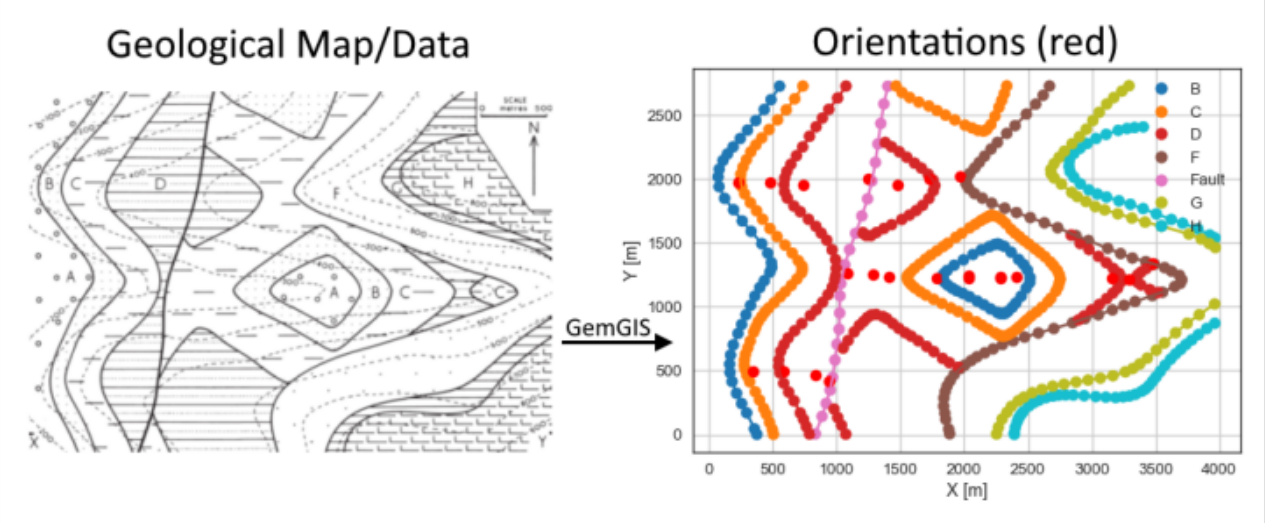
[13]:
```
(0.0, 2731.0)
```



7.16.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example16.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes16.shp')
strikes.head()
```

[15]:
```
   id formation    Z                                        geometry
0   1    Fault2  300  LINESTRING (1369.455 2575.828, 1108.966 1461.029)
1   2    Fault2  400  LINESTRING (1331.187 2325.368, 1176.529 1650.260)
2   1    Fault1  400      LINESTRING (1021.872 873.278, 846.365 36.387)
3   2    Fault1  500      LINESTRING (973.575 482.148, 931.611 286.847)
4   2        B4  300     LINESTRING (92.345 2105.786, 115.570 1859.812)
```

Calculate Orientations for each formation

[16]:
```
orientations_f1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'Fault1'].sort_values(by='Z', ascending=True).reset_index())
orientations_f1
```

[16]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 72.24   281.86 450.00  POINT (943.356 419.665)      1.00    Fault1 943.36

       Y
0 419.67
```

[17]:
```
orientations_f2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'Fault2'].sort_values(by='Z', ascending=True).reset_index())
orientations_f2
```

[17]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 78.84   283.09 350.00  POINT (1246.534 2003.121)      1.00    Fault2

        X       Y
0 1246.53 2003.12
```

[18]:
```
orientations_f3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'Fault3'].sort_values(by='Z', ascending=True).reset_index())
orientations_f3
```

[18]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 72.65   281.34 350.00  POINT (1084.157 1262.429)      1.00    Fault3

        X       Y
0 1084.16 1262.43
```

[19]:
```
orientations_b1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B1'].sort_values(by='Z', ascending=True).reset_index())
orientations_b1
```

[19]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 20.35    85.12 250.00  POINT (347.292 491.913)      1.00        B1 347.29

       Y
0 491.91
```

[20]:
```
orientations_b2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B2'].sort_values(by='Z', ascending=True).reset_index())
orientations_b2
```

[20]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 21.95   265.66 350.00  POINT (2029.784 1223.500)      1.00        B2

        X       Y
0 2029.78 1223.50
```

[21]:
```
orientations_b3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B3'].sort_values(by='Z', ascending=True).reset_index())
orientations_b3
```

[21]:
```
   dip  azimuth      Z                   geometry  polarity formation       X  \
0 0.00    85.61 400.00  POINT (2282.355 1220.333)      1.00        B3 2282.36

        Y
0 1220.33
```

[22]:
```
orientations_b4 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B4'].sort_values(by='Z', ascending=True).reset_index())
orientations_b4
```

[22]:
```
    dip  azimuth      Z                   geometry  polarity formation      X  \
0 21.87    85.18 250.00  POINT (230.639 1970.923)      1.00        B4 230.64

        Y
0 1970.92
```

[23]:
```
orientations_c1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C1'].sort_values(by='Z', ascending=True).reset_index())
orientations_c1
```

[23]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 22.48    85.96 250.00  POINT (585.612 498.643)      1.00        C1 585.61
```

(continues on next page)

(continued from previous page)

```
       Y
0 498.64
```

[24]:
```
orientations_c2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C2'].sort_values(by='Z', ascending=True).reset_index())
orientations_c2
```

[24]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.41   264.42 350.00  POINT (1784.602 1228.779)      1.00        C2
1  0.00     0.00 400.00  POINT (2032.159 1240.391)      1.00        C2

        X       Y
0 1784.60 1228.78
1 2032.16 1240.39
```

[25]:
```
orientations_c3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C3'].sort_values(by='Z', ascending=True).reset_index())
orientations_c3
```

[25]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 11.47    85.21 450.00  POINT (2406.530 1229.571)      1.00        C3
1  0.00     0.00 500.00  POINT (2284.863 1233.002)      1.00        C3

        X       Y
0 2406.53 1229.57
1 2284.86 1233.00
```

[26]:
```
orientations_c4 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C4'].sort_values(by='Z', ascending=True).reset_index())
orientations_c4
```

[26]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.59   265.72 350.00  POINT (1723.636 2004.441)      1.00        C4
1 22.15   265.73 450.00  POINT (1968.554 2021.331)      1.00        C4

        X       Y
0 1723.64 2004.44
1 1968.55 2021.33
```

[27]:
```
orientations_c5 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C5'].sort_values(by='Z', ascending=True).reset_index())
orientations_c5
```

[27]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 21.62    85.70 250.00  POINT (475.557 1973.562)      1.00        C5 475.56

        Y
0 1973.56
```

[28]:
```
orientations_d1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D1'].sort_values(by='Z', ascending=True).reset_index())
orientations_d1
```

[28]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 24.97    86.28 350.00  POINT (837.128 468.952)      1.00        D1 837.13

       Y
0 468.95
```

[29]:
```
orientations_d2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D2'].sort_values(by='Z', ascending=True).reset_index())
orientations_d2
```

[29]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0  0.00    85.37 400.00  POINT (3288.420 1212.416)      1.00        D2
1 11.25     0.00 450.00  POINT (3159.890 1219.542)      1.00        D2

        X       Y
0 3288.42 1212.42
1 3159.89 1219.54
```

[30]:
```
orientations_d3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D3'].sort_values(by='Z', ascending=True).reset_index())
orientations_d3
```

[30]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 22.81   266.11 450.00  POINT (1481.621 1955.879)      1.00        D3

        X       Y
0 1481.62 1955.88
```

[31]:
```
orientations_d4 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D4'].sort_values(by='Z', ascending=True).reset_index())
orientations_d4
```

[31]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0  0.00   265.65 400.00  POINT (1283.945 1251.212)      1.00        D4
1 11.48     0.00 450.00  POINT (1407.460 1235.905)      1.00        D4

        X       Y
0 1283.94 1251.21
1 1407.46 1235.90
```

[32]:
```
orientations_d5 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D5'].sort_values(by='Z', ascending=True).reset_index())
orientations_d5
```

[32]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 23.00    84.84 350.00  POINT (735.783 1952.448)      1.00        D5 735.78

        Y
0 1952.45
```

[33]:
```
orientations_g = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'G'].sort_values(by='Z', ascending=True).reset_index())
orientations_g
```

[33]:
```
   dip  azimuth      Z                   geometry  polarity formation       X  \
0 5.85    89.87 650.00  POINT (3475.011 1339.361)      1.00         G 3475.01

        Y
0 1339.36
```

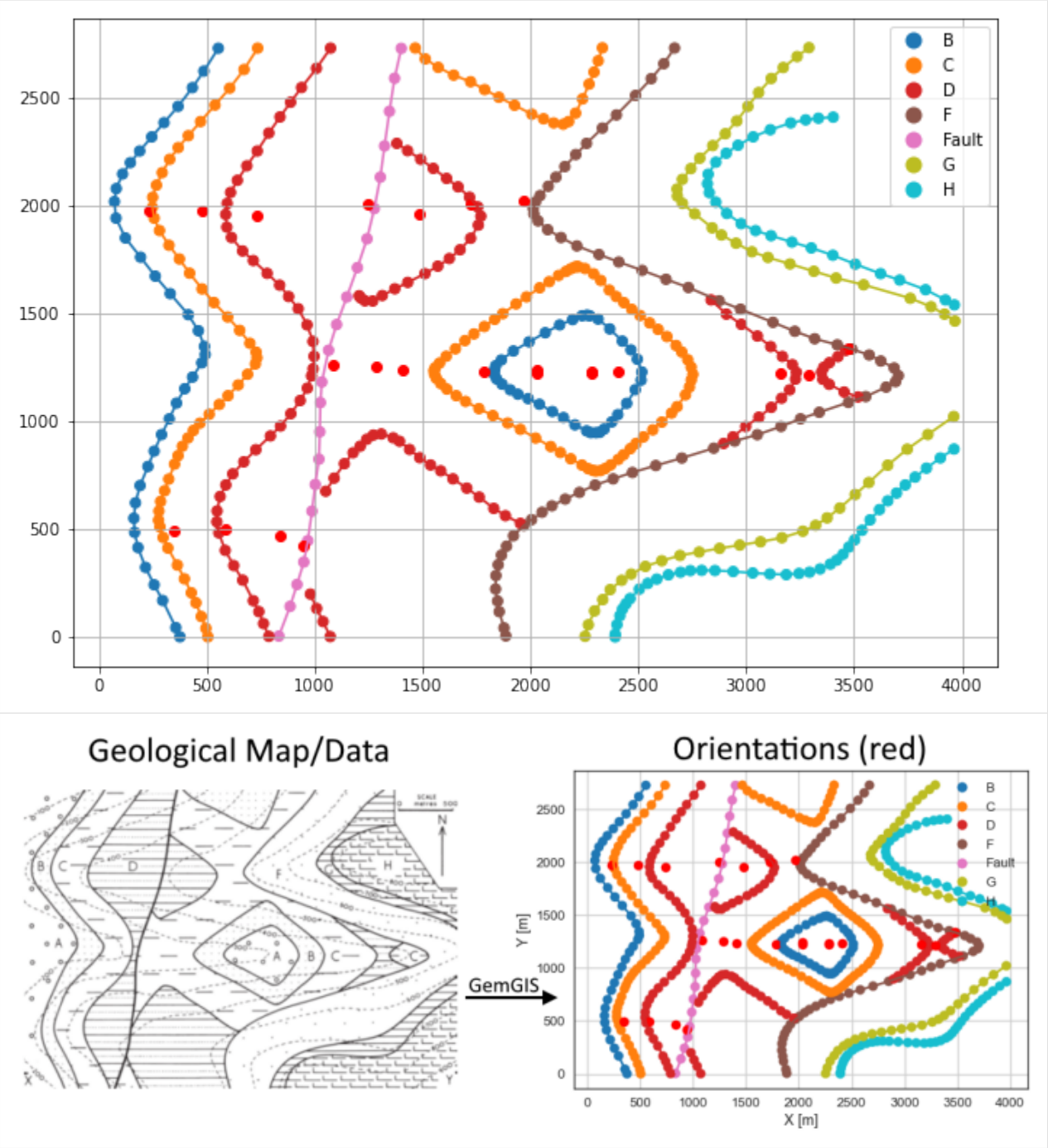
Merging Orientations

[34]:
```
import pandas as pd
orientations = pd.concat([orientations_f1, orientations_f2, orientations_f3,␣
↪orientations_b1, orientations_b2, orientations_b3, orientations_b4, orientations_c1,␣
↪orientations_c2, orientations_c3, orientations_c4, orientations_c5, orientations_d1,␣
↪orientations_d2, orientations_d3, orientations_d4, orientations_d5, orientations_g]).
↪reset_index()
orientations['formation'] = ['Fault', 'Fault', 'Fault', 'B', 'B', 'B', 'B', 'C', 'C', 'C
↪', 'C', 'C', 'C', 'C', 'C', 'D', 'D', 'D', 'D', 'D', 'D', 'D', 'G']
orientations = orientations[orientations['formation'].isin(['Fault', 'B', 'C', 'D', 'G
↪'])]
orientations.head()
```

[34]:
```
   index   dip  azimuth      Z                   geometry  polarity formation  \
0      0 72.24   281.86 450.00    POINT (943.356 419.665)      1.00     Fault
1      0 78.84   283.09 350.00  POINT (1246.534 2003.121)      1.00     Fault
2      0 72.65   281.34 350.00  POINT (1084.157 1262.429)      1.00     Fault
3      0 20.35    85.12 250.00    POINT (347.292 491.913)      1.00         B
4      0 21.95   265.66 350.00  POINT (2029.784 1223.500)      1.00         B

        X       Y
0  943.36  419.67
1 1246.53 2003.12
2 1084.16 1262.43
3  347.29  491.91
4 2029.78 1223.50
```

Plotting the Orientations

[35]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example16.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```


B
C
D
F
Fault
G
H
2500
2000
1500
1000
500
0
0
500
1000
1500
2000
2500
3000
3500
4000
Geological Map/Data
Orientations (red)
GemGIS
Y [m]
X [m]

7.16.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[36]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[37]:
```
geo_model = gp.create_model('Model16')
geo_model
```

[37]:
```
Model16  2022-04-05 11:10
```

Initiate Data

[38]:
```
gp.init_data(geo_model, [0, 3968, 0, 2731, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[38]:
```
Model16  2022-04-05 11:10
```

Model Surfaces

[39]:
```
geo_model.surfaces
```

[39]:

	surface	series	order_surfaces	color	id
0	H	Default series	1	#015482	1
1	G	Default series	2	#9f0052	2
2	Fault	Default series	3	#ffbe00	3
3	F	Default series	4	#728f02	4
4	D	Default series	5	#443988	5
5	C	Default series	6	#ff3f20	6
6	B	Default series	7	#5DA629	7

Mapping the Stack to Surfaces

[40]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('Fault'),
                          'Strata1': ('H', 'G', 'F'),
                          'Strata2': ('D', 'C', 'B'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('A')
geo_model.set_is_fault(['Fault1'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[40]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault1	1	Fault	True	True	False
Strata1	2	Erosion	True	False	False
Strata2	3	Erosion	True	False	False

Showing the Number of Data Points

[41]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[41]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
2	Fault	Fault1	1	#527682	1	22	3
0	H	Strata1	1	#9f0052	2	63	0
1	G	Strata1	2	#ffbe00	3	54	1
3	F	Strata1	3	#728f02	4	66	0
4	D	Strata2	1	#443988	5	116	7
5	C	Strata2	2	#ff3f20	6	139	8
6	B	Strata2	3	#5DA629	7	75	4
7	A	Strata2	4	#4878d0	8	0	0

Loading Digital Elevation Model

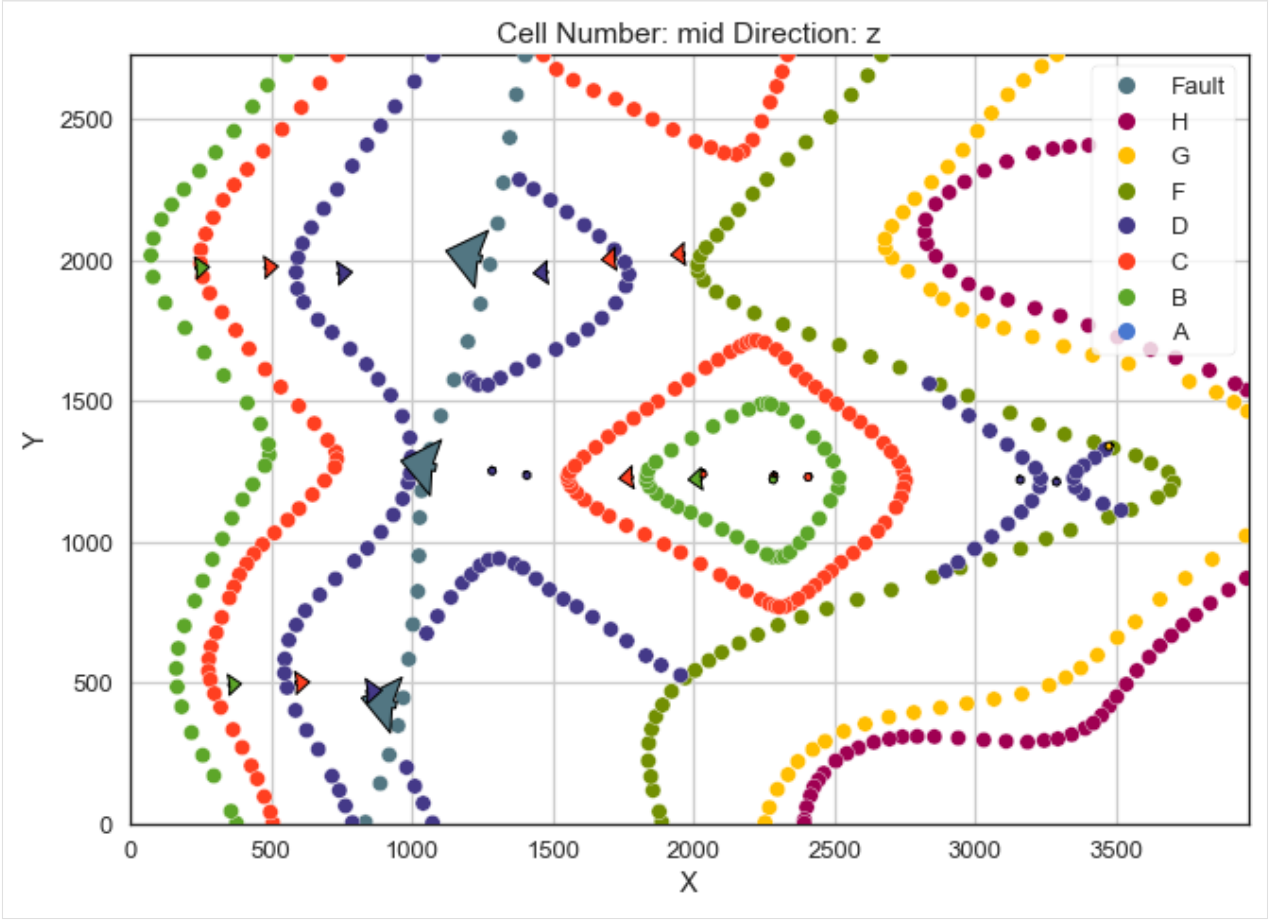
[42]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster16.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

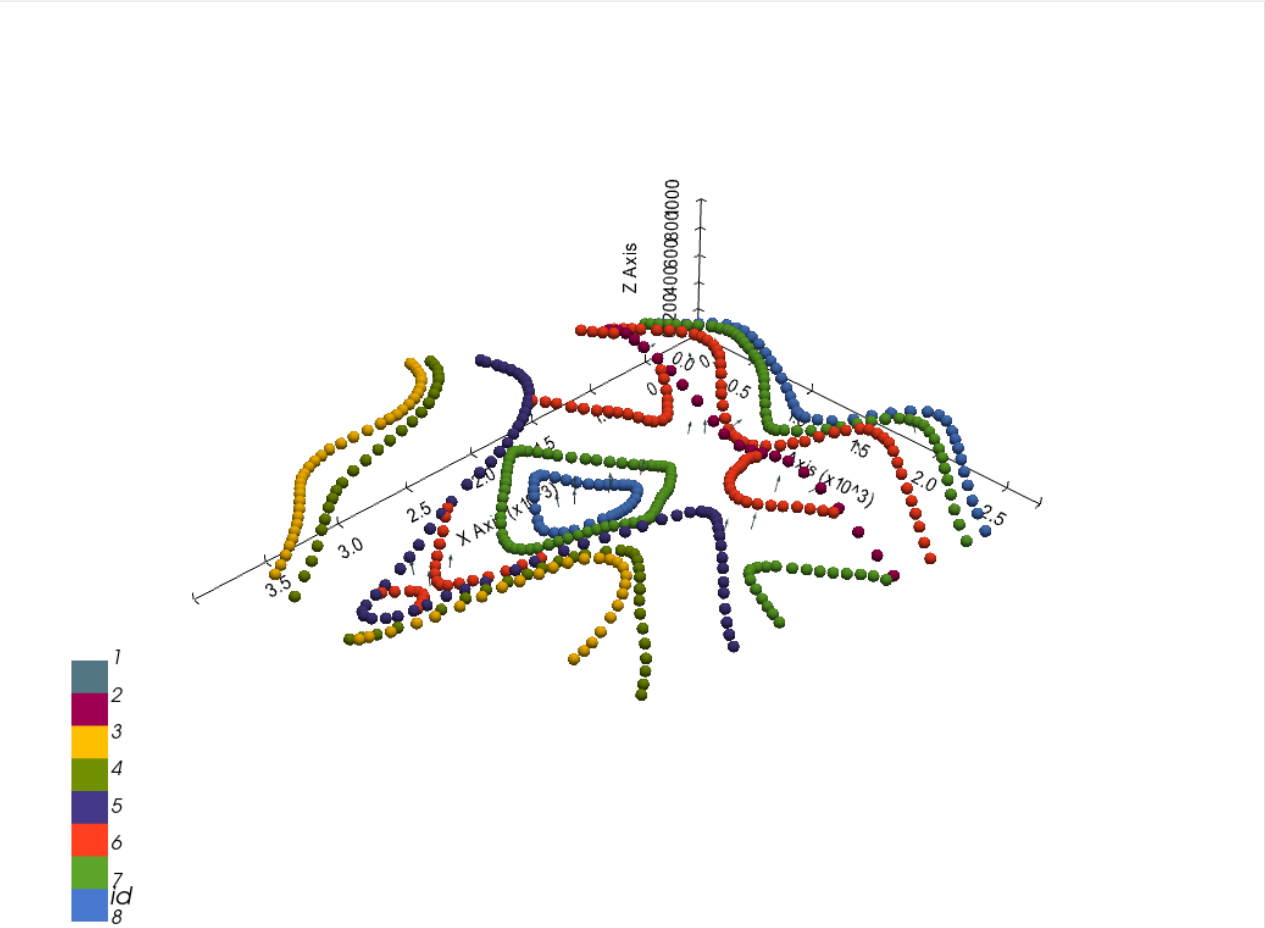
[42]:
```
Grid Object. Values:
array([[  19.84      ,   13.655     ,    5.        ],
       [  19.84      ,   13.655     ,   15.        ],
       [  19.84      ,   13.655     ,   25.        ],
       ...,
       [3963.00251889, 2705.99084249,  744.06182861],
       [3963.00251889, 2715.99450549,  742.93560791],
       [3963.00251889, 2725.9981685 ,  741.82879639]])
```

Plotting Input Data

[43]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[44]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[44]:
```
<gempy.plot.vista.GemPyToVista at 0x20a4d850f40>
```

Setting the Interpolator

[45]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                      values
range               4919.69
$C_o$             576271.07
drift equations  [3, 3, 3]
```

[45]:
```
<gempy.core.interpolator.InterpolatorModel at 0x20a4b61e9a0>
```

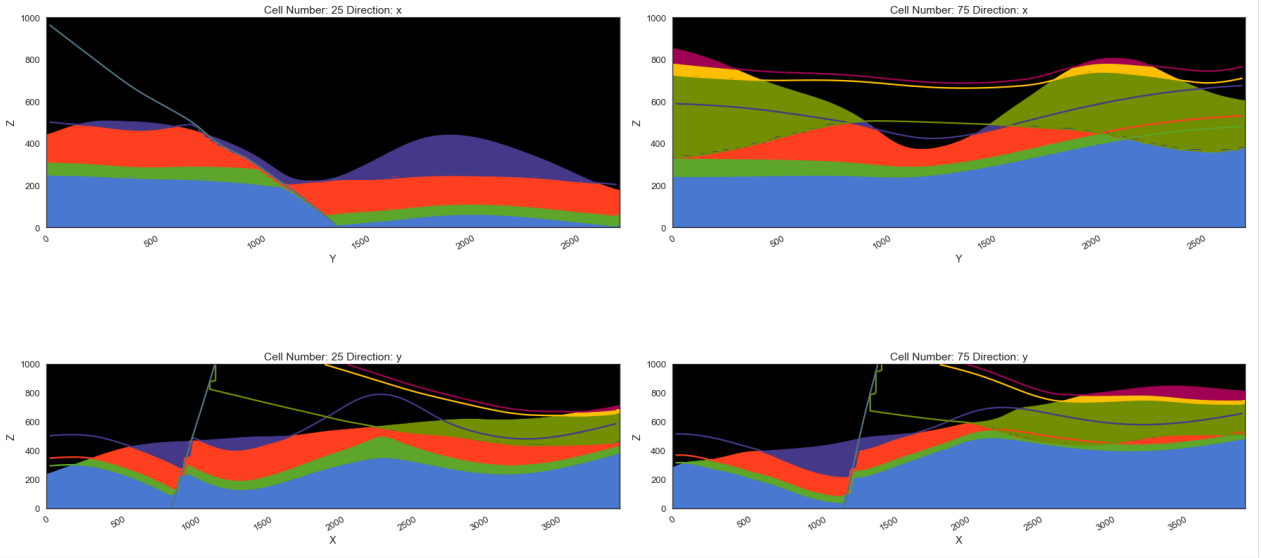
Computing Model

[46]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

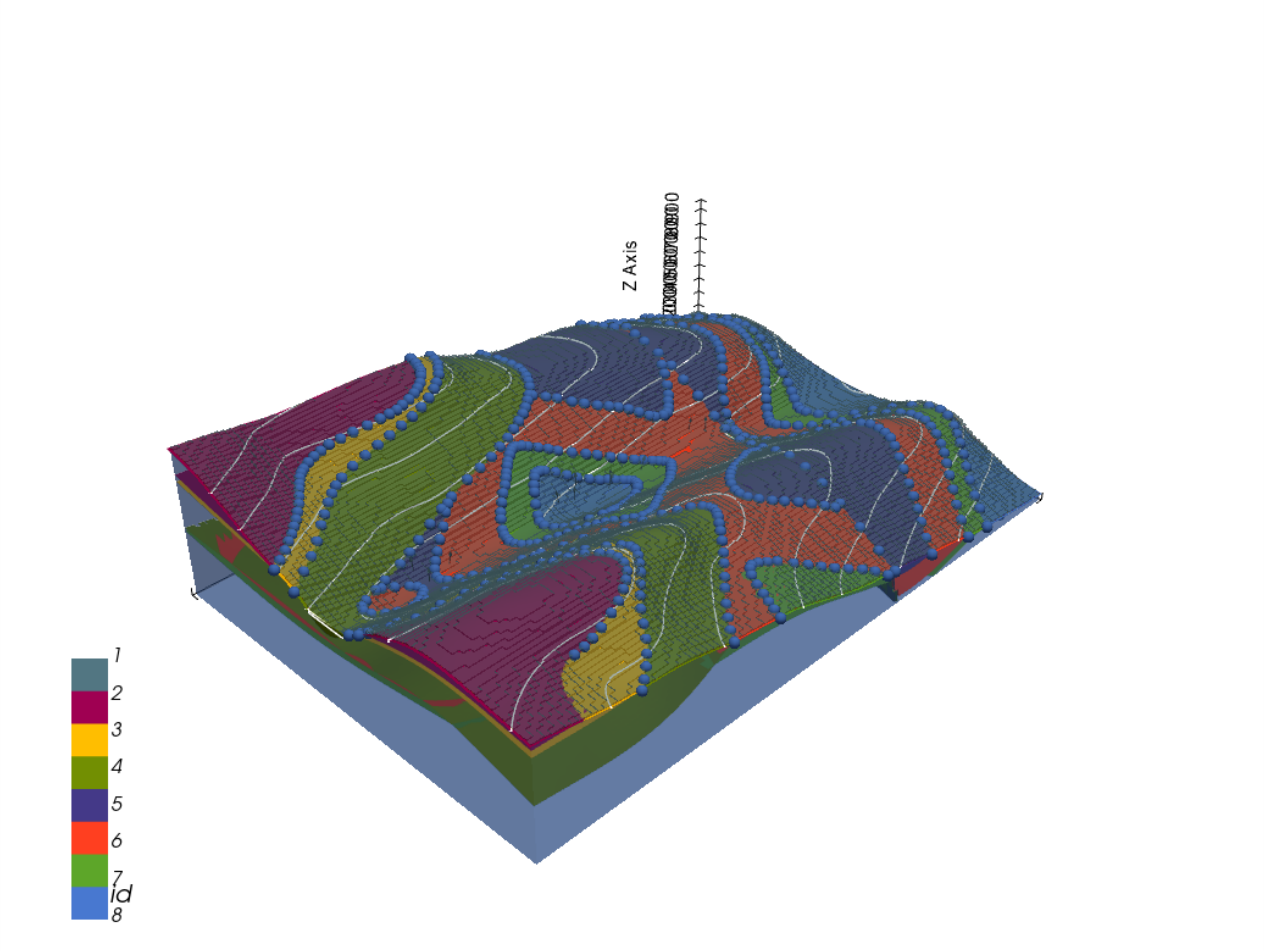
[47]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[47]:
```
<gempy.plot.visualization_2d.Plot2D at 0x20a4c1c94f0>
```



Plotting 3D Model

[48]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.17 Example 17 - Three Point Problem and Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3892 m wide (W-E extent) and 2683 m high (N-S extent). The model represents a coal seam that was encountered at the surface and in boreholes. A second coal seam is located 300 m vertically above the first one. The vertical model extent varies between 0 m and 1000 m.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

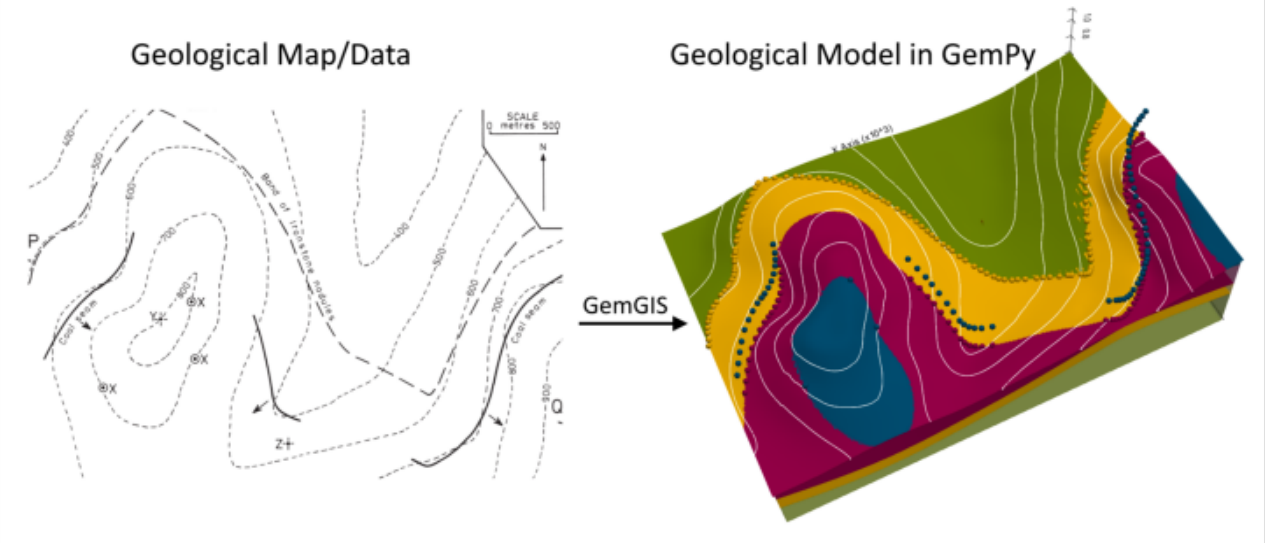
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example17.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.17.1 Licensing

7.17.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.17.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

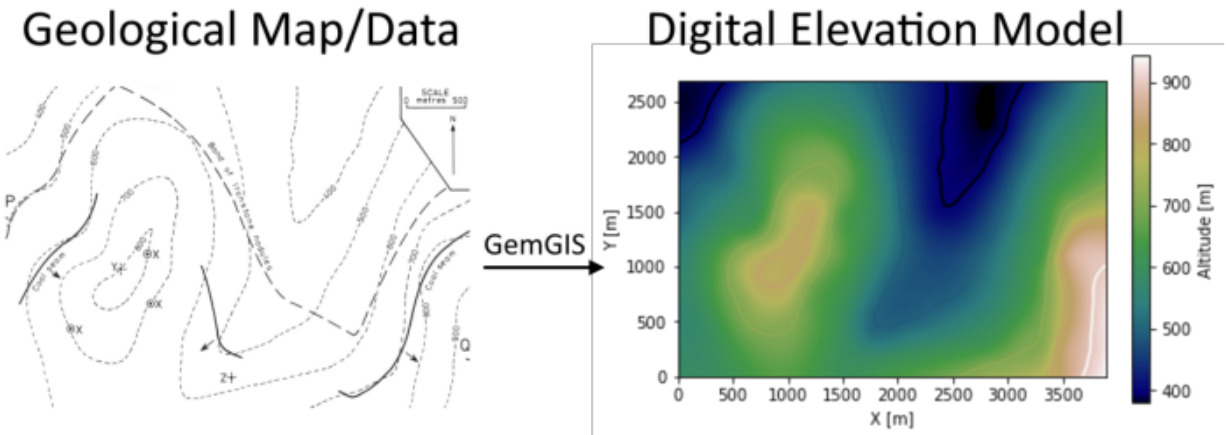
[4]:
```
file_path = 'data/example17/'
gg.download_gemgis_data.download_tutorial_data(filename="example17_three_point_problem.
↪zip", dirpath=file_path)
```

```
Downloading file 'example17_three_point_problem.zip' from 'https://rwth-aachen.sciebo.de/
↪s/AfXRsZywYDbUF34/download?path=%2Fexample17_three_point_problem.zip' to 'C:\Users\
↪ale93371\Documents\gemgis\docs\getting_started\example\data\example17'.
```

7.17.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example17.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo17.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	600	LINESTRING (399.582 9.701, 373.041 104.254, 35...
1	None	900	LINESTRING (3680.737 2.651, 3702.716 47.854, 3...
2	None	800	LINESTRING (3390.857 3.480, 3398.322 17.995, 3...
3	None	700	LINESTRING (2564.763 4.725, 2631.530 33.339, 2...
4	None	800	LINESTRING (1224.018 1463.246, 1211.577 1384.4...

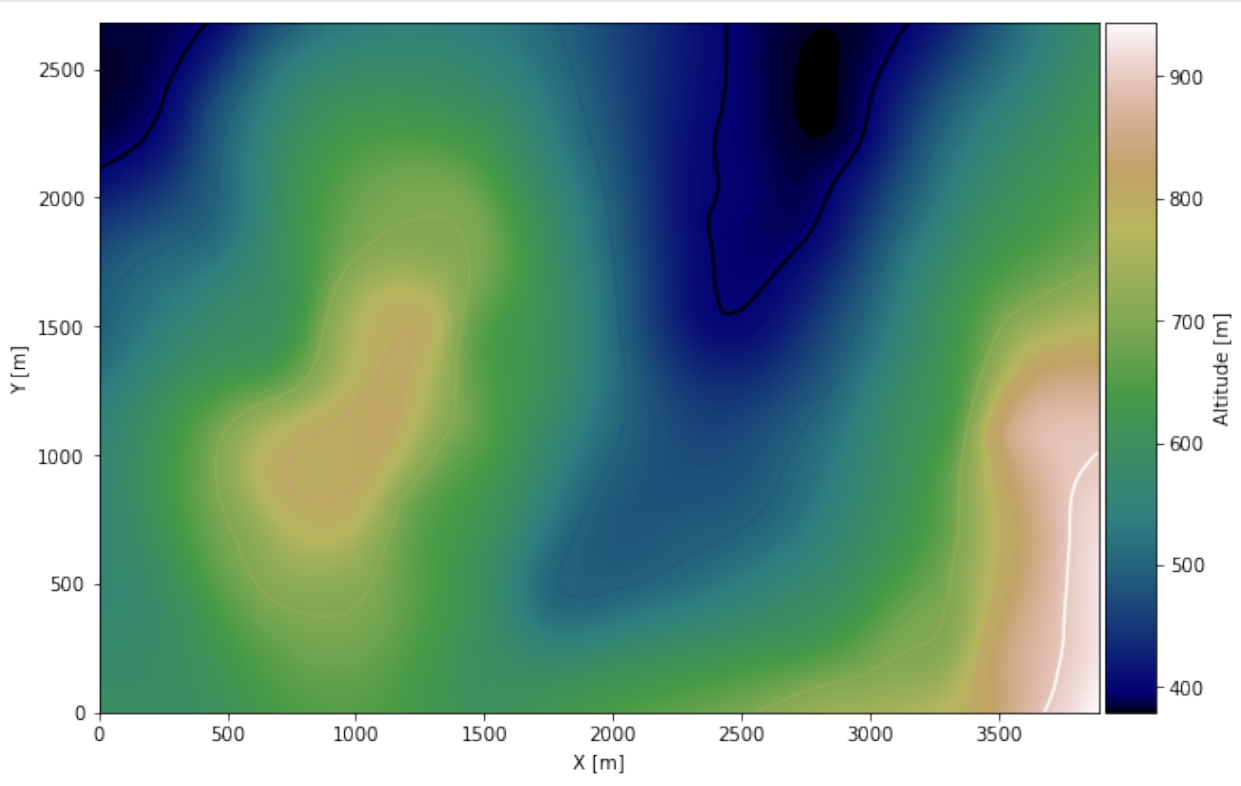
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 3892, 0, 2683], cmap='gist_earth
↪')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3892)
ax.set_ylim(0, 2683)
```

[8]:
```
(0.0, 2683.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster17.tif', extent = [0, 3892, 0, 2683], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

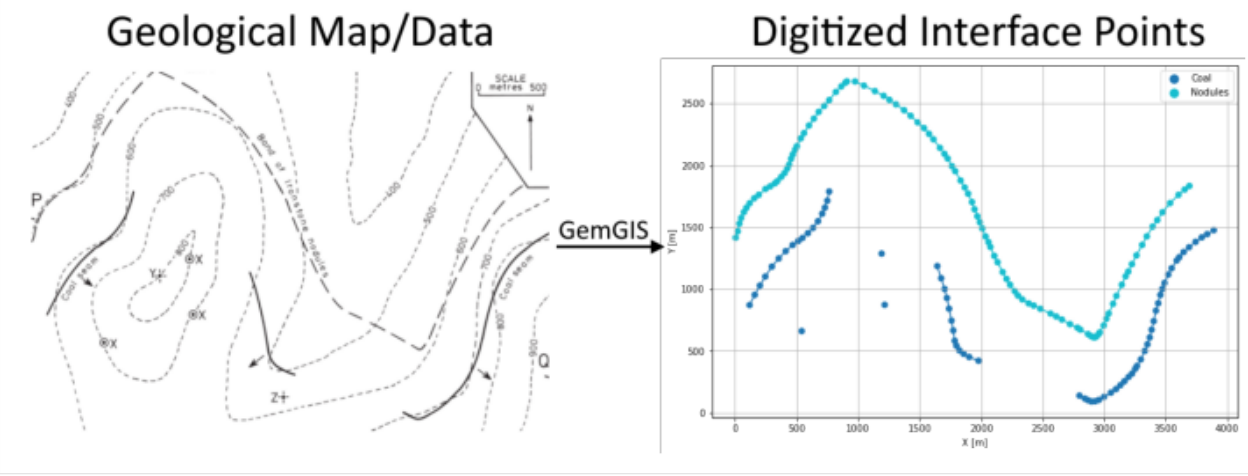
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster17.tif')
```

7.17.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example17.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces17.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Coal	LINESTRING (117.997 870.009, 159.675 952.743, ...
1	None	Nodules	LINESTRING (4.782 1414.932, 21.578 1466.563, 3...
2	None	Nodules	LINESTRING (974.572 2678.334, 1047.353 2644.74...

(continues on next page)

(continued from previous page)

```
3  None      Coal  LINESTRING (1643.286 1185.393, 1675.011 1087.1...
4  None      Coal  LINESTRING (2798.761 137.845, 2842.927 113.585...
```

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=True)
interfaces_coords.head()
```

[12]:
```
    formation                  geometry       X      Y      Z
0        Coal   POINT (117.997 870.009)  118.00 870.01 583.24
141      Coal  POINT (3095.794 190.409) 3095.79 190.41 703.05
140      Coal  POINT (3054.116 161.483) 3054.12 161.48 705.77
139      Coal  POINT (2999.064 127.892) 2999.06 127.89 708.86
138      Coal  POINT (2956.764 103.010) 2956.76 103.01 710.91
```

[13]:
```
points = gpd.read_file(file_path + 'points17.shp')
points
```

[13]:
```
     id formation    Z                   geometry
0  None      Coal  500  POINT (1191.049 1286.166)
1  None      Coal  400   POINT (1215.931 871.875)
2  None      Coal  400    POINT (541.619 659.753)
```

[14]:
```
points_coords = gg.vector.extract_xy(gdf=points)
points_coords
```

[14]:
```
  formation      Z                   geometry       X       Y
0      Coal 500.00  POINT (1191.049 1286.166) 1191.05 1286.17
1      Coal 400.00   POINT (1215.931 871.875) 1215.93  871.87
2      Coal 400.00    POINT (541.619 659.753)  541.62  659.75
```

[15]:
```
import pandas as pd

interfaces_coords = pd.concat([interfaces_coords, points_coords])
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Coal',
↪'Nodules'])].reset_index()
interfaces_coords.head()
```

[15]:
```
   index formation                  geometry       X      Y      Z
0      0      Coal   POINT (117.997 870.009)  118.00 870.01 583.24
1    141      Coal  POINT (3095.794 190.409) 3095.79 190.41 703.05
2    140      Coal  POINT (3054.116 161.483) 3054.12 161.48 705.77
3    139      Coal  POINT (2999.064 127.892) 2999.06 127.89 708.86
4    138      Coal  POINT (2956.764 103.010) 2956.76 103.01 710.91
```

Creating data for the coal seam 300 m heigher

[16]:
```
interfaces_coal300 = interfaces_coords[interfaces_coords['formation']=='Coal']
interfaces_coal300['Z'] = interfaces_coal300['Z']+300
interfaces_coal300['formation'] = 'Coal300'
interfaces_coal300
```

[16]:
```
     index formation                   geometry       X       Y       Z
0        0   Coal300    POINT (117.997 870.009)  118.00  870.01  883.24
1      141   Coal300   POINT (3095.794 190.409) 3095.79  190.41 1003.05
2      140   Coal300   POINT (3054.116 161.483) 3054.12  161.48 1005.77
3      139   Coal300   POINT (2999.064 127.892) 2999.06  127.89 1008.86
4      138   Coal300   POINT (2956.764 103.010) 2956.76  103.01 1010.91
..     ...       ...                       ...     ...     ...     ...
64       1   Coal300    POINT (159.675 952.743)  159.68  952.74  892.40
65       7   Coal300   POINT (468.216 1355.837)  468.22 1355.84  906.89
169      0   Coal300  POINT (1191.049 1286.166) 1191.05 1286.17  800.00
170      1   Coal300   POINT (1215.931 871.875) 1215.93  871.87  700.00
171      2   Coal300    POINT (541.619 659.753)  541.62  659.75  700.00

[69 rows x 6 columns]
```

Merging the interface data

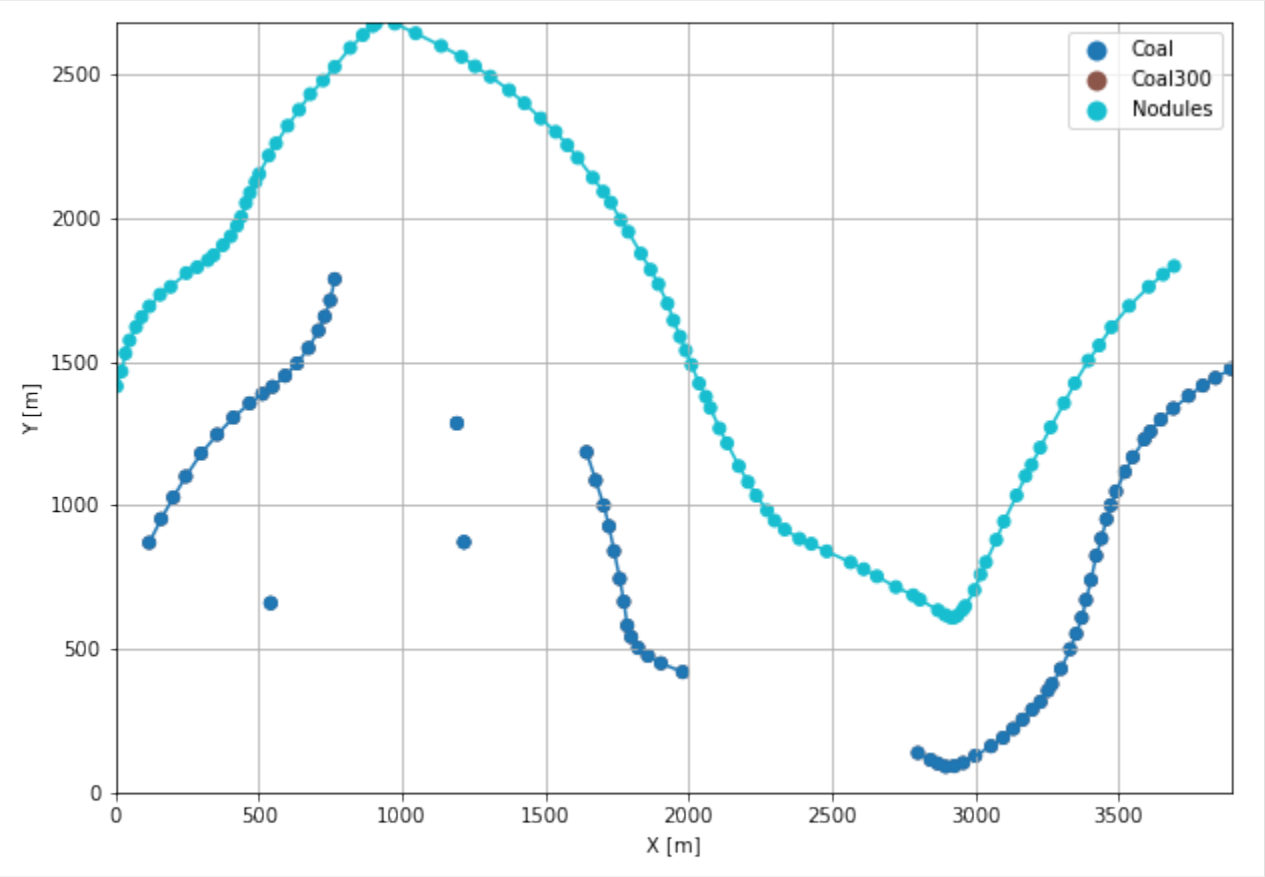
[17]:
```
interfaces_coords = pd.concat([interfaces_coal300, interfaces_coords])
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Coal300',
↪'Coal', 'Nodules'])].reset_index()
interfaces_coords.head()
```

[17]:
```
   level_0  index formation                   geometry       X      Y       Z
0        0      0   Coal300    POINT (117.997 870.009)  118.00 870.01  883.24
1        1    141   Coal300   POINT (3095.794 190.409) 3095.79 190.41 1003.05
2        2    140   Coal300   POINT (3054.116 161.483) 3054.12 161.48 1005.77
3        3    139   Coal300   POINT (2999.064 127.892) 2999.06 127.89 1008.86
4        4    138   Coal300   POINT (2956.764 103.010) 2956.76 103.01 1010.91
```

Plotting the Interface Points

[18]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
points_coords.plot(ax=ax, column='formation', legend=False, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3892)
ax.set_ylim(0, 2683)
```

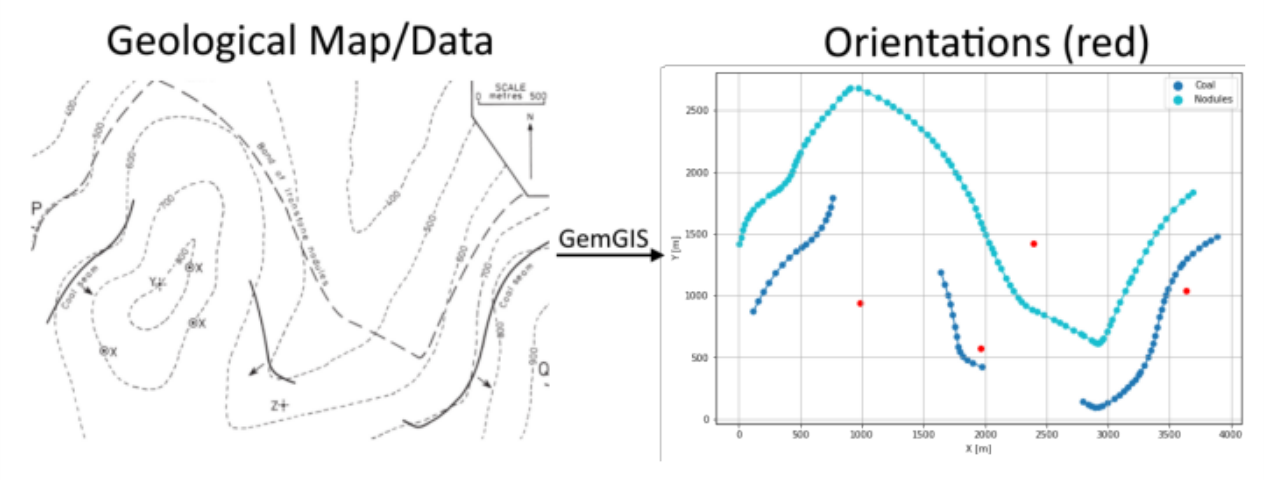
[18]:
```
(0.0, 2683.0)
```



7.17.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[19]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example17.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[20]:
```
strikes = gpd.read_file(file_path + 'strikes17.shp')
strikes.head()
```

[20]:
```
   id formation    Z                                           geometry
0   2     Coal2  800  LINESTRING (3799.965 1421.775, 3474.006 1008.728)
1   1     Coal2  700   LINESTRING (3158.000 249.816, 4109.750 1469.052)
2   1     Coal1  500    LINESTRING (1793.202 564.267, 1911.393 443.898)
3   2     Coal1  600   LINESTRING (1667.546 1093.328, 2477.467 175.791)
4   1   Nodules  500   LINESTRING (2038.915 1418.665, 2585.083 789.763)
```

Calculate Orientations for each formation

[21]:
```
orientations_coal1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal1'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal1
```

[21]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 22.04   228.51 550.00  POINT (1962.402 569.321)      1.00     Coal1 1962.40

       Y
0 569.32
```

[22]:
```
orientations_coal2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal2'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal2
```

[22]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 24.93   128.01 750.00  POINT (3635.430 1037.343)      1.00     Coal2

        X       Y
0 3635.43 1037.34
```

[23]:
```
orientations_nodules = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Nodules'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_nodules
```

[23]:
```
    dip  azimuth      Z                  geometry  polarity formation  \
0 10.45   228.04 550.00  POINT (2395.977 1421.931)      1.00   Nodules

        X       Y
0 2395.98 1421.93
```

[24]:
```
orientations_coal3 = gg.vector.calculate_orientation_for_three_point_problem(gdf=points)
orientations_coal3['Z'] = orientations_coal3['Z'].astype(float)
orientations_coal3['azimuth'] = orientations_coal3['azimuth'].astype(float)
orientations_coal3['dip'] = orientations_coal3['dip'].astype(float)
orientations_coal3['dip'] = 180 - orientations_coal3['dip']
orientations_coal3['azimuth'] = 180 - orientations_coal3['azimuth']
orientations_coal3['polarity'] = orientations_coal3['polarity'].astype(float)
orientations_coal3['X'] = orientations_coal3['X'].astype(float)
orientations_coal3['Y'] = orientations_coal3['Y'].astype(float)
orientations_coal3
```

[24]:
```
       Z formation  azimuth   dip  polarity      X      Y  \
0 433.33      Coal   197.46 13.95      1.00 982.87 939.26

                  geometry
0  POINT (982.867 939.265)
```

Merging Orientations

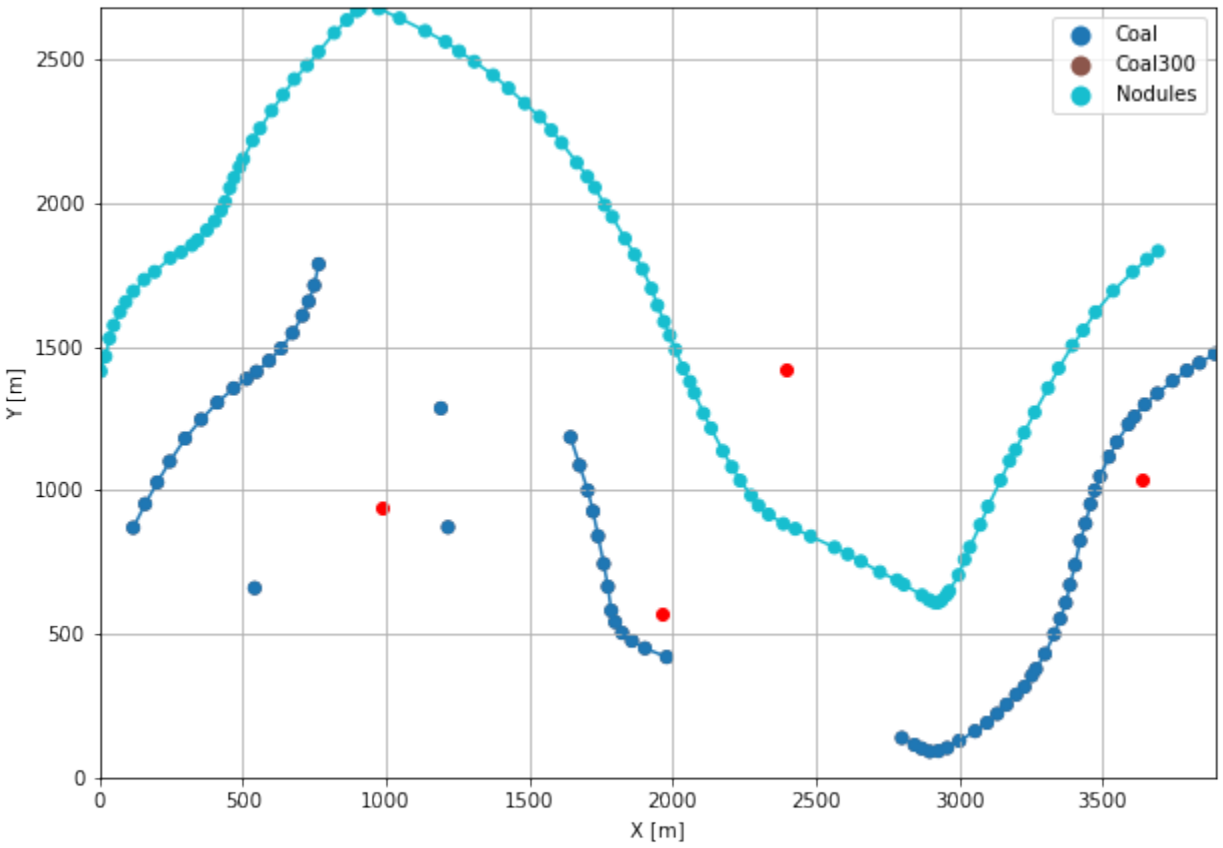
[25]:
```
import pandas as pd
orientations = pd.concat([orientations_coal1, orientations_coal2, orientations_coal3,␣
↪orientations_nodules]).reset_index()
orientations['formation'] = ['Coal', 'Coal', 'Coal', 'Nodules']
orientations = orientations[orientations['formation'].isin(['Coal', 'Nodules'])]
orientations
```

[25]:
```
   index   dip  azimuth      Z                  geometry  polarity formation  \
0      0 22.04   228.51 550.00   POINT (1962.402 569.321)      1.00      Coal
1      0 24.93   128.01 750.00  POINT (3635.430 1037.343)      1.00      Coal
2      0 13.95   197.46 433.33    POINT (982.867 939.265)      1.00      Coal
3      0 10.45   228.04 550.00  POINT (2395.977 1421.931)      1.00   Nodules

        X       Y
0 1962.40  569.32
1 3635.43 1037.34
2  982.87  939.26
3 2395.98 1421.93
```

Plotting the Orientations

[26]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3892)
ax.set_ylim(0, 2683)
```

[26]:
```
(0.0, 2683.0)
```



7.17.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[27]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
```

(continues on next page)

(continued from previous page)

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[28]:
```
geo_model = gp.create_model('Model17')
geo_model
```

[28]:
```
Model17  2022-04-05 11:11
```

Initiate Data

[29]:
```
gp.init_data(geo_model, [0, 3892, 0, 2683, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[29]:
```
Model17  2022-04-05 11:11
```

Model Surfaces

[30]:
```
geo_model.surfaces
```

[30]:

	surface	series	order_surfaces	color	id
0	Coal300	Default series	1	#015482	1
1	Coal	Default series	2	#9f0052	2
2	Nodules	Default series	3	#ffbe00	3

Mapping the Stack to Surfaces

[31]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Coal300', 'Coal', 'Nodules'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[31]:

	surface	series	order_surfaces	color	id
0	Coal300	Strata1	1	#015482	1
1	Coal	Strata1	2	#9f0052	2
2	Nodules	Strata1	3	#ffbe00	3
3	Basement	Strata1	4	#728f02	4

Showing the Number of Data Points

[32]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[32]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Coal300	Strata1	1	#015482	1	69	0
1	Coal	Strata1	2	#9f0052	2	69	3
2	Nodules	Strata1	3	#ffbe00	3	103	1
3	Basement	Strata1	4	#728f02	4	0	0

Loading Digital Elevation Model

[33]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster17.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[33]:
```
Grid Object. Values:
array([[  19.46      ,   13.415     ,    5.        ],
       [  19.46      ,   13.415     ,   15.        ],
       [  19.46      ,   13.415     ,   25.        ],
       ...,
       [3886.99742931, 2657.97201493,  594.15032959],
       [3886.99742931, 2667.98320896,  593.5199585 ],
       [3886.99742931, 2677.99440299,  592.91101074]])
```

Defining Custom Section

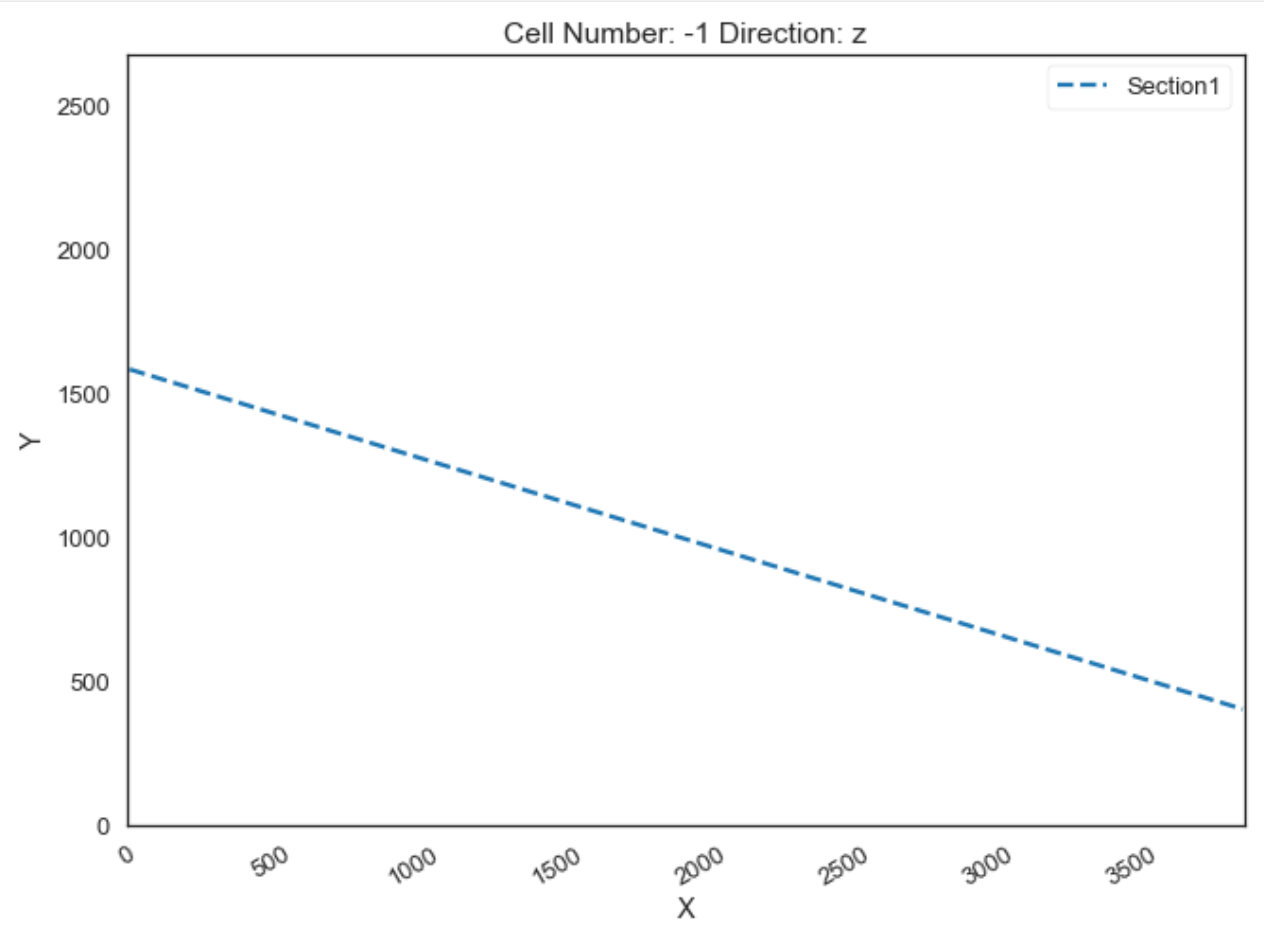
[34]:
```
custom_section = gpd.read_file(file_path + 'customsection17.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='name')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[34]:
```
                                          start                                 ␣
↪stop resolution    dist
Section1  [6.648494638704051, 1585.9987115807176]  [3888.2970423641327, 402.
↪8423753990243]  [100, 80] 4057.96
```

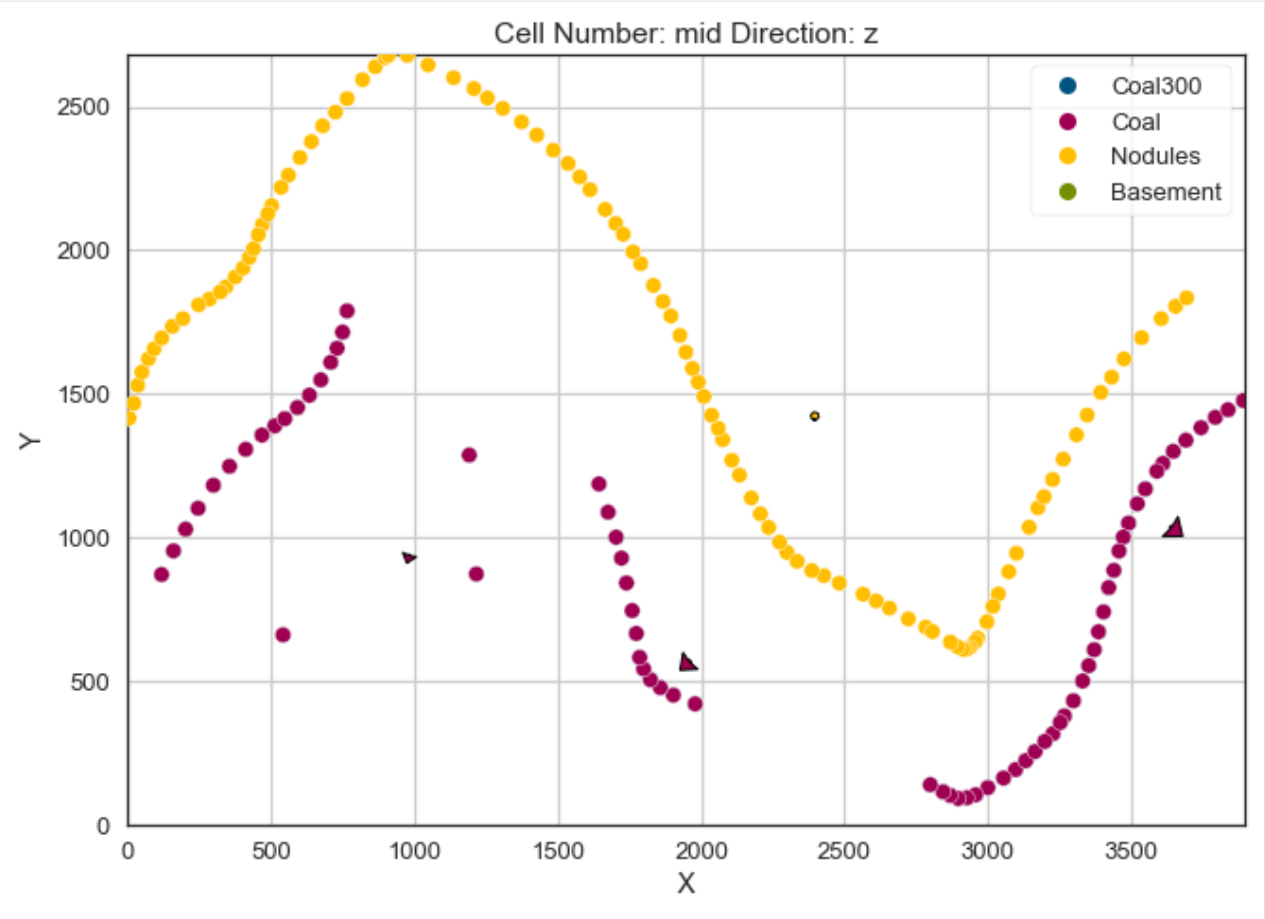
[35]:
```
gp.plot.plot_section_traces(geo_model)
```

[35]:
```
<gempy.plot.visualization_2d.Plot2D at 0x18708b29ac0>
```

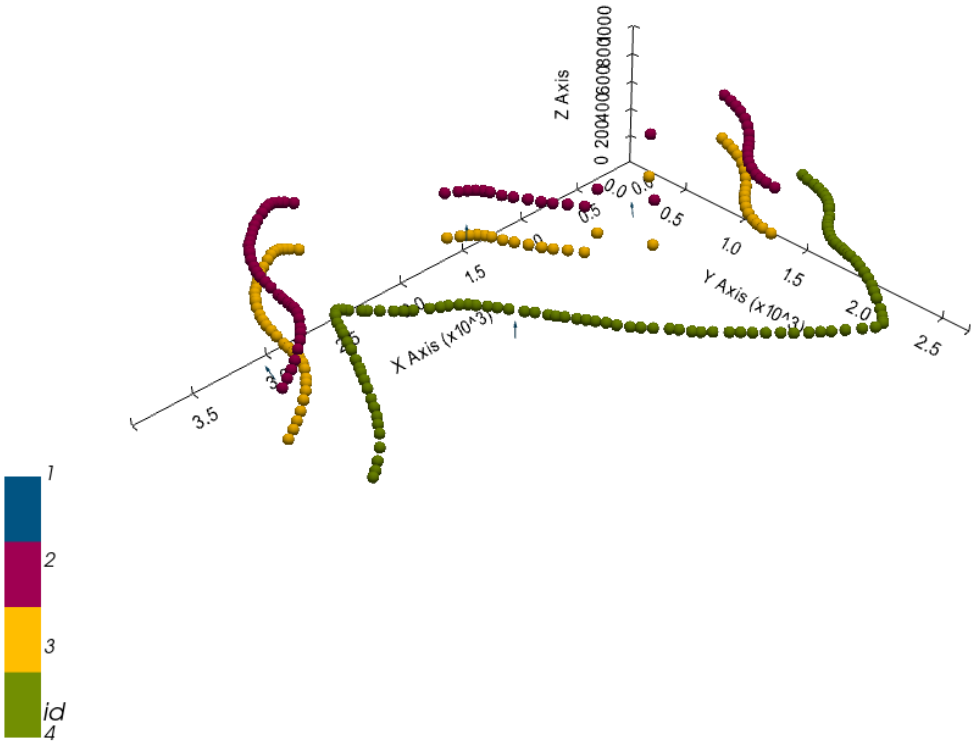


Plotting Input Data

[36]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[37]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[37]:
```
<gempy.plot.vista.GemPyToVista at 0x187073560a0>
```

Setting the Interpolator

[38]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              4831.79
$C_o$            555860.79
drift equations        [3]
```

[38]:
```
<gempy.core.interpolator.InterpolatorModel at 0x18706891c70>
```

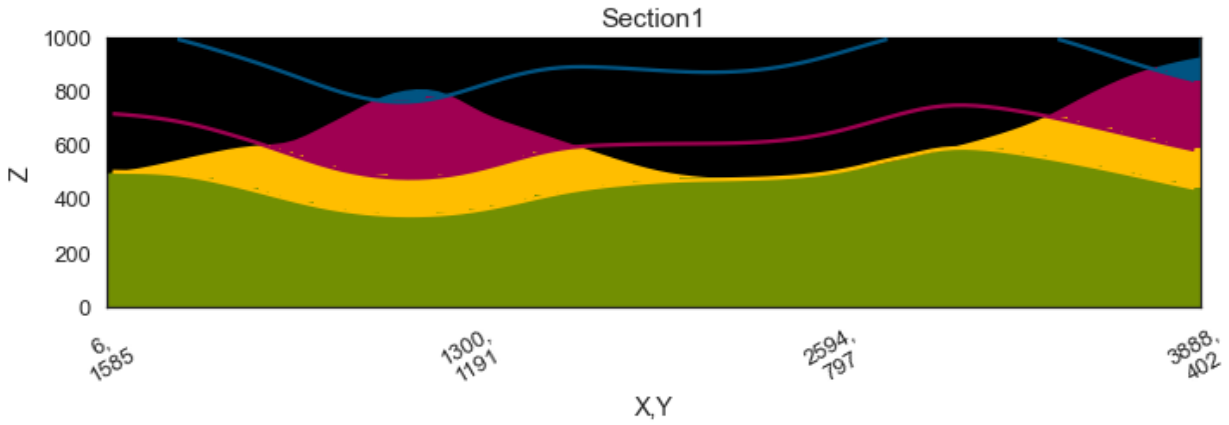
Computing Model

[39]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

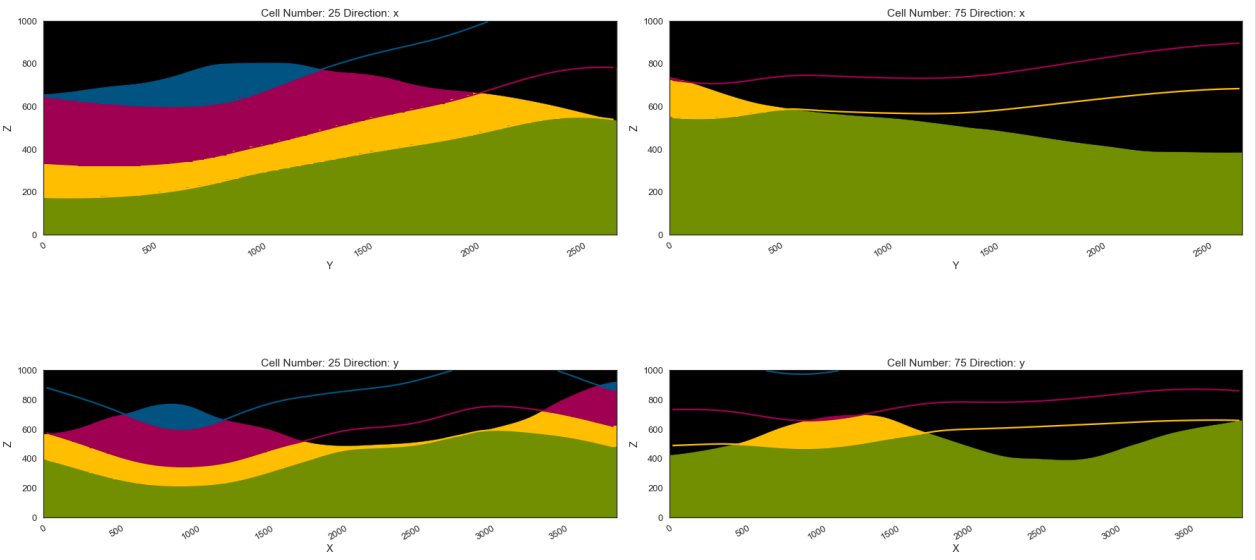
[40]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[40]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1870b8fa760>
```



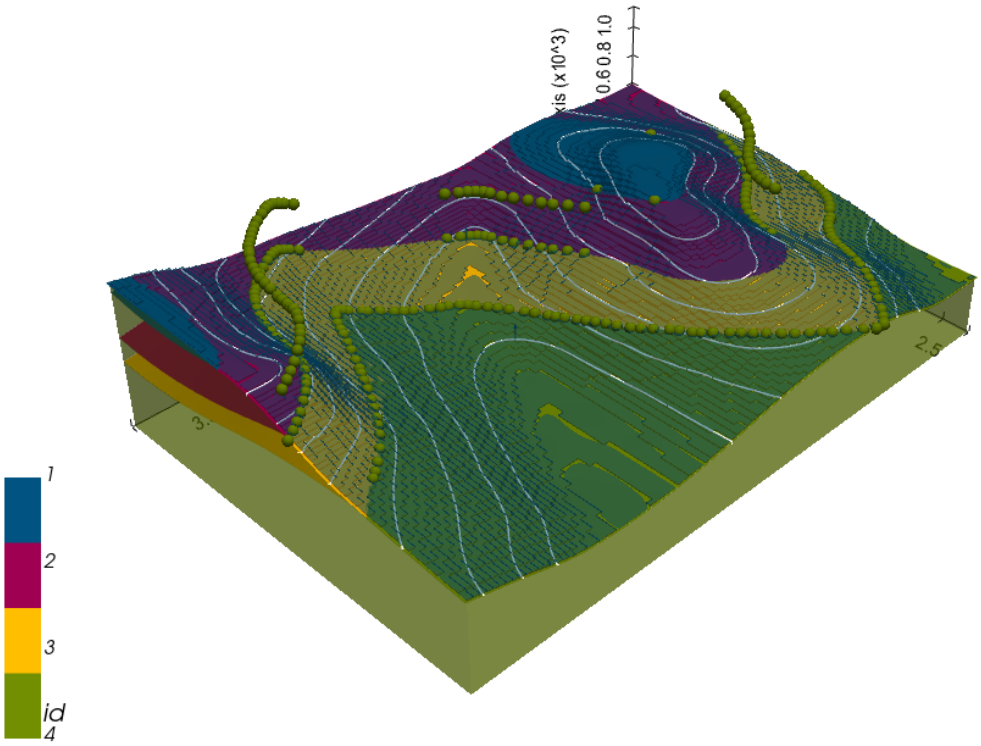
[41]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[41]:
```
<gempy.plot.visualization_2d.Plot2D at 0x187089cd670>
```



Plotting 3D Model

[42]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:
```
```

7.18 Example 18 - Faulted Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 4022 m wide (W-E extent) and 2761 m high (N-S extent). The vertical model extent varies between 0 m and 1000 m. The model represents folded layers which are cut by a roughly N-S trending fault.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

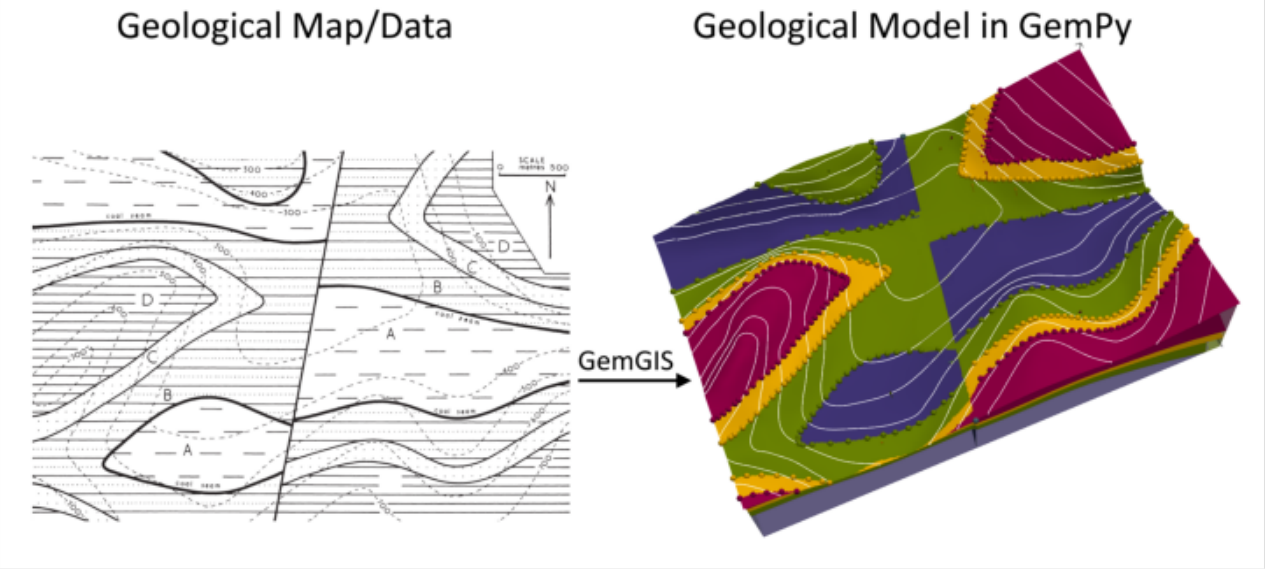
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/cover_example18.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.18.1 Licensing

7.18.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.18.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

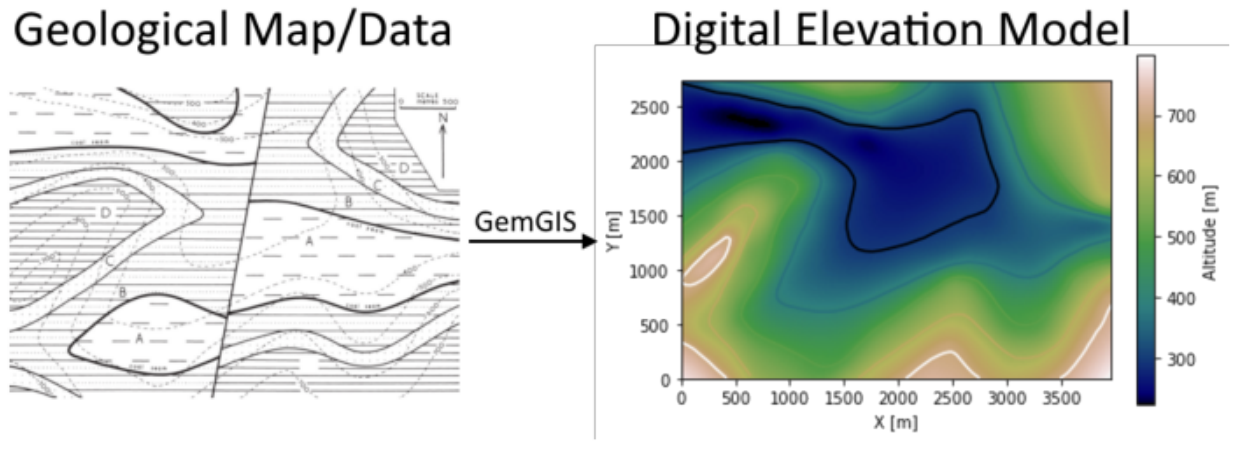
[4]:
```
file_path = 'data/example18/'
gg.download_gemgis_data.download_tutorial_data(filename="example18_faulted_folded_layers.
↪zip", dirpath=file_path)
```

```
Downloading file 'example18_faulted_folded_layers.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2Fexample18_faulted_folded_layers.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\example\data\example18'.
```

7.18.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example18.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo18.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	700	LINESTRING (3.286 332.303, 62.614 304.560, 120...
1	None	700	LINESTRING (27.188 946.063, 60.480 983.196, 98...

(continues on next page)

(continued from previous page)

```
2  None  500  LINESTRING (1024.228 2750.211, 1101.055 2721.1...
3  None  400  LINESTRING (317.849 2757.466, 375.042 2739.113...
4  None  300  LINESTRING (4.567 2715.638, 60.480 2702.407, 1...
```

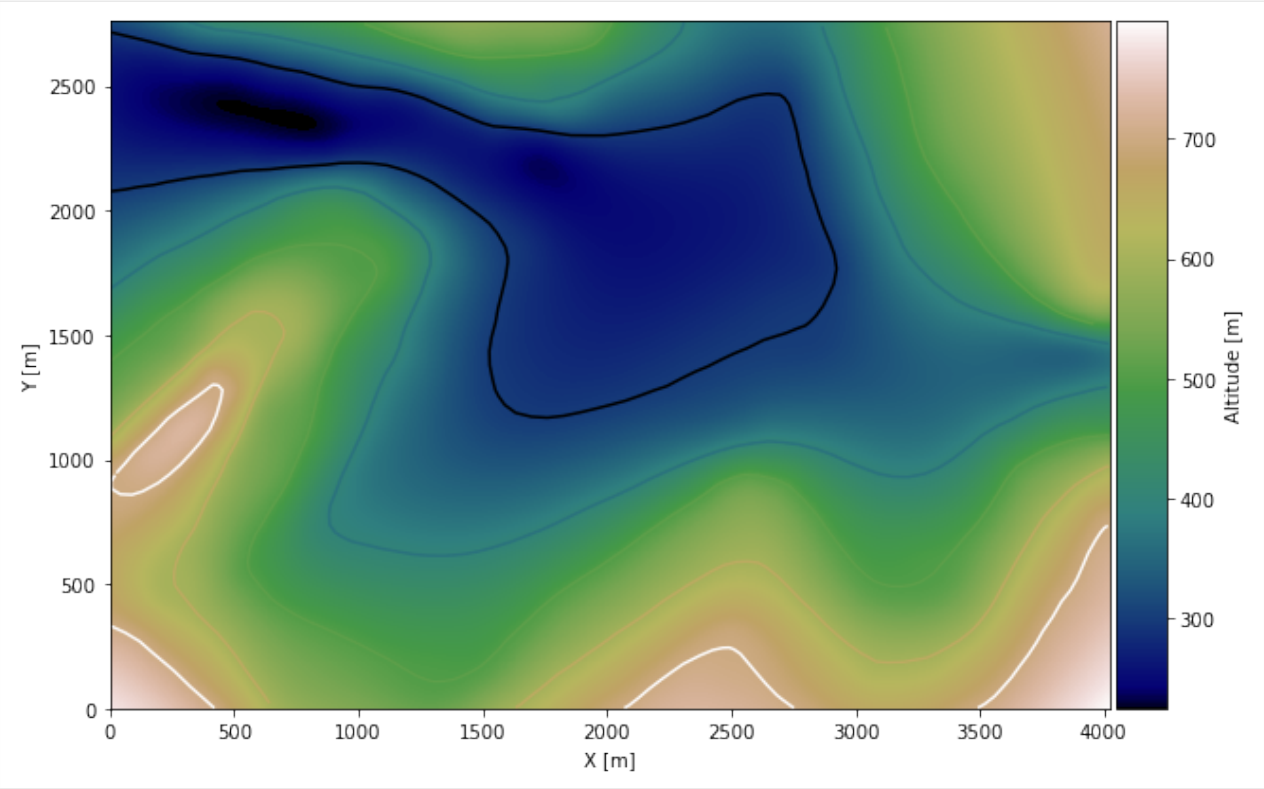
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable
fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 4022, 0, 2761], cmap='gist_earth
↪')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4022)
ax.set_ylim(0, 2761)
```

[8]:
```
(0.0, 2761.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster18.tif', extent = [0, 4022, 0, 2761], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster18.tif')
```

7.18.5 Interface Points of stratigraphic boundaries

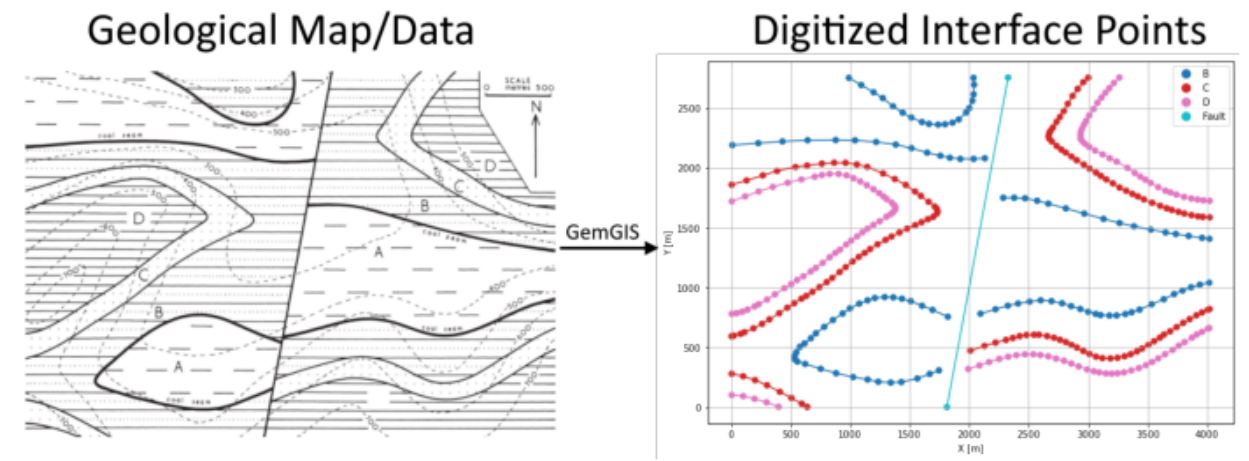
The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example18.png')
plt.figure(figsize=(10, 10))
```

(continues on next page)

(continued from previous page)

```
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces18.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Fault	LINESTRING (2326.868 2757.893, 1812.982 1.522)
1	None	D	LINESTRING (4.567 1721.586, 117.246 1763.414, ...
2	None	C	LINESTRING (3.713 1860.728, 129.197 1896.580, ...
3	None	C	LINESTRING (4.140 279.378, 100.600 258.891, 20...
4	None	D	LINESTRING (4.140 101.823, 100.600 92.433, 196...

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Fault', 'D',
↪'C', 'B'])]
interfaces_coords.head()
```

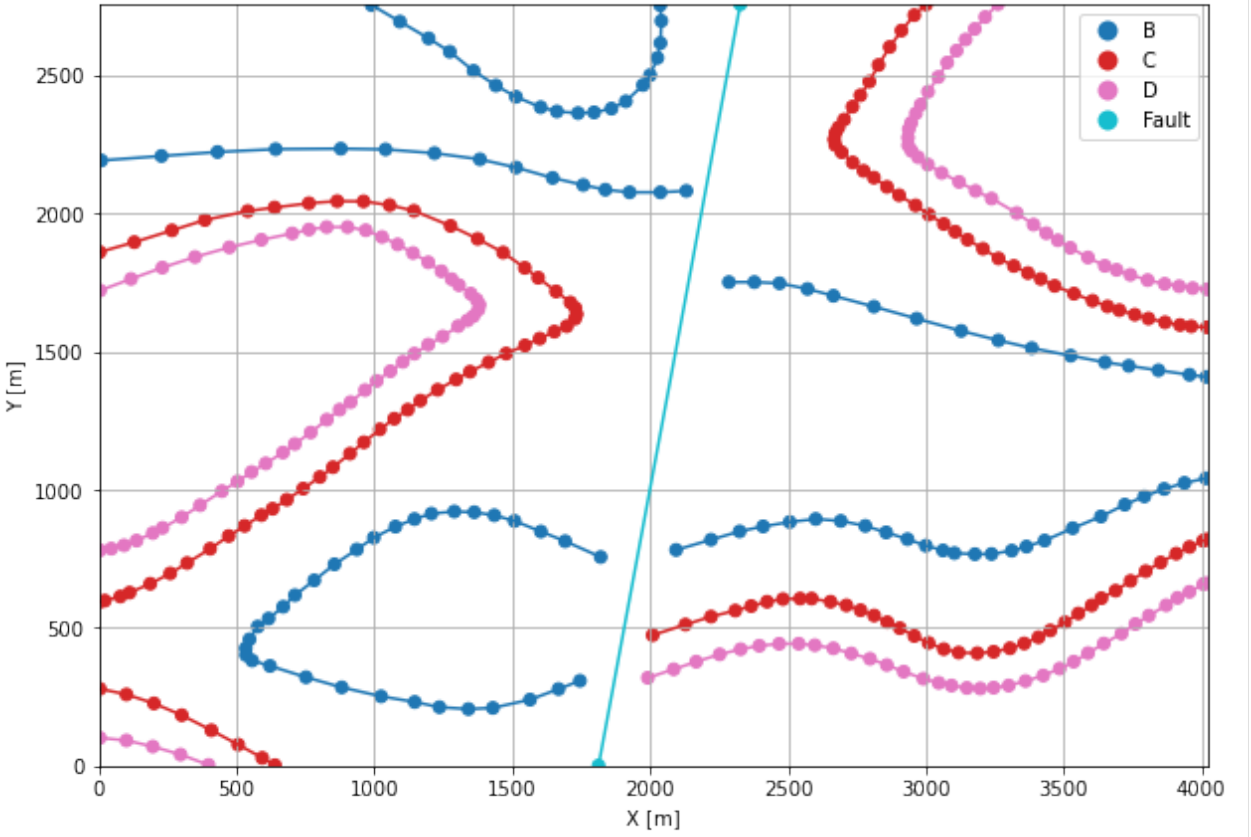
[12]:

	formation	geometry	X	Y	Z
0	Fault	POINT (2326.868 2757.893)	2326.87	2757.89	396.78
1	Fault	POINT (1812.982 1.522)	1812.98	1.52	645.29
263	D	POINT (3437.014 1929.018)	3437.01	1929.02	488.43
256	D	POINT (3007.638 2177.425)	3007.64	2177.42	371.88
257	D	POINT (3057.575 2147.121)	3057.57	2147.12	390.98

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4022)
ax.set_ylim(0, 2761)
```

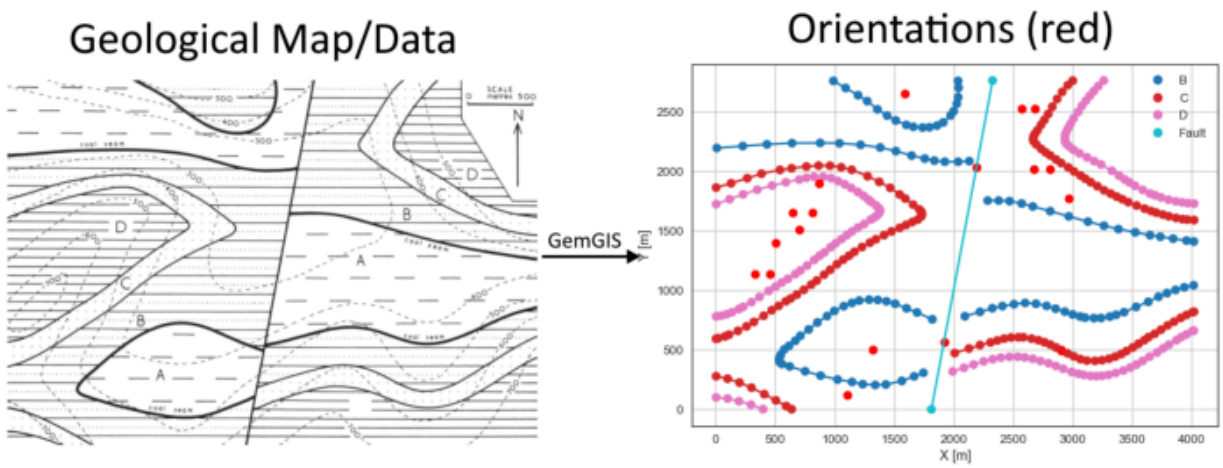
[13]:
```
(0.0, 2761.0)
```



7.18.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example18.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes18.shp')
strikes.head()
```

[15]:
```
   id formation    Z                                           geometry
0   1        C2  300  LINESTRING (2391.317 2132.396, 2809.062 2132.076)
1   2        C2  400  LINESTRING (2334.977 1895.513, 3168.227 1897.754)
2   3        C2  500  LINESTRING (3747.308 1638.464, 2608.992 1647.107)
3   1        D2  400  LINESTRING (2393.237 2137.518, 3072.514 2137.838)
4   2        D2  500  LINESTRING (2273.516 1895.513, 3486.738 1900.315)
```

Calculate Orientations for each formation

[16]:
```
orientations_d1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D1'].sort_values(by='Z', ascending=True).reset_index())
orientations_d1
```

[16]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 21.76   180.11 650.00  POINT (2901.894 492.786)      1.00        D1 2901.89

       Y
0 492.79
```

[17]:
```
orientations_d2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D2'].sort_values(by='Z', ascending=True).reset_index())
orientations_d2
```

[17]:
```
    dip  azimuth      Z                  geometry  polarity formation  \
0 22.69   359.82 450.00  POINT (2806.501 2017.796)      1.00        D2
```

(continues on next page)

(continued from previous page)

```
        X       Y
0 2806.50 2017.80
```

[18]:
```
orientations_d3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D3'].sort_values(by='Z', ascending=True).reset_index())
orientations_d3
```

[18]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0  0.00     0.00 400.00  POINT (645.350 1647.827)      1.00        D3 645.35
1 21.14     0.41 450.00  POINT (507.462 1396.780)      1.00        D3 507.46
2 21.96   359.46 550.00  POINT (333.881 1138.690)      1.00        D3 333.88

        Y
0 1647.83
1 1396.78
2 1138.69
```

[19]:
```
orientations_d4 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'D4'].sort_values(by='Z', ascending=True).reset_index())
orientations_d4
```

[19]:
```
    dip  azimuth      Z                  geometry  polarity formation  \
0 21.86   180.50 450.00  POINT (2683.258 2522.931)      1.00        D4

        X       Y
0 2683.26 2522.93
```

[20]:
```
orientations_c1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C1'].sort_values(by='Z', ascending=True).reset_index())
orientations_c1
```

[20]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 22.06   180.54 350.00  POINT (865.026 1893.753)      1.00        C1 865.03

        Y
0 1893.75
```

[21]:
```
orientations_c2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C2'].sort_values(by='Z', ascending=True).reset_index())
orientations_c2
```

[21]:
```
    dip  azimuth      Z                  geometry  polarity formation  \
0 23.03   359.89 350.00  POINT (2675.896 2014.435)      1.00        C2
1 21.87     0.23 450.00  POINT (2964.876 1769.710)      1.00        C2

        X       Y
0 2675.90 2014.43
1 2964.88 1769.71
```

[22]:
```
orientations_c3 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C3'].sort_values(by='Z', ascending=True).reset_index())
orientations_c3
```

[22]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0  0.00     0.32 300.00  POINT (812.528 1648.067)      1.00        C3 812.53
1 11.14   359.66 350.00  POINT (699.208 1512.340)      1.00        C3 699.21
2 23.45   359.84 450.00  POINT (459.445 1139.411)      1.00        C3 459.45
3 20.58   180.00 550.00   POINT (278.262 889.083)      1.00        C3 278.26

        Y
0 1648.07
1 1512.34
2 1139.41
3  889.08
```

[23]:
```
orientations_c4 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C4'].sort_values(by='Z', ascending=True).reset_index())
orientations_c4
```

[23]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 21.54   180.38 350.00  POINT (2575.061 2523.892)      1.00        C4

        X       Y
0 2575.06 2523.89
```

[24]:
```
orientations_c5 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'C5'].sort_values(by='Z', ascending=True).reset_index())
orientations_c5
```

[24]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 23.45   180.08 650.00  POINT (1102.869 125.938)      1.00        C5 1102.87

       Y
0 125.94
```

[25]:
```
orientations_b1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B1'].sort_values(by='Z', ascending=True).reset_index())
orientations_b1
```

[25]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 24.55   180.05 450.00  POINT (1590.398 2647.615)      1.00        B1

        X       Y
0 1590.40 2647.61
```

[26]:
```
orientations_b2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B2'].sort_values(by='Z', ascending=True).reset_index())
orientations_b2
```

[26]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 11.29     0.26 450.00  POINT (1321.745 502.309)      1.00        B2 1321.74

       Y
0 502.31
```

[27]:
```
orientations_fault = gpd.read_file(file_path + 'orientations18_fault.shp')
orientations_fault = gg.vector.extract_xyz(gdf=orientations_fault, dem=topo_raster)
orientations_fault
```

[27]:
```
  formation   dip  azimuth  polarity                   geometry       X  \
0     Fault 90.00   280.00      1.00   POINT (1918.192 566.731) 1918.19
1     Fault 90.00   280.00      1.00  POINT (2190.927 2031.561) 2190.93

        Y      Z
0  566.73 490.30
1 2031.56 255.41
```

Merging Orientations

[28]:
```
import pandas as pd
orientations = pd.concat([orientations_fault, orientations_d1, orientations_d2,␣
↪orientations_d3, orientations_d4, orientations_c1, orientations_c2, orientations_c3[:-
↪1], orientations_c4, orientations_c5, orientations_b1, orientations_b2]).reset_index()
orientations['formation'] = ['Fault', 'Fault', 'D', 'D', 'D', 'D', 'D', 'D', 'C', 'C', 'C
↪', 'C', 'C', 'C', 'C', 'C', 'B', 'B']
orientations = orientations[orientations['formation'].isin(['Fault', 'D', 'C', 'B'])]
orientations
```

[28]:
```
    index formation   dip  azimuth  polarity                   geometry  \
0       0     Fault 90.00   280.00      1.00   POINT (1918.192 566.731)
1       1     Fault 90.00   280.00      1.00  POINT (2190.927 2031.561)
2       0         D 21.76   180.11      1.00   POINT (2901.894 492.786)
3       0         D 22.69   359.82      1.00  POINT (2806.501 2017.796)
4       0         D  0.00     0.00      1.00   POINT (645.350 1647.827)
5       1         D 21.14     0.41      1.00   POINT (507.462 1396.780)
6       2         D 21.96   359.46      1.00   POINT (333.881 1138.690)
7       0         D 21.86   180.50      1.00  POINT (2683.258 2522.931)
8       0         C 22.06   180.54      1.00   POINT (865.026 1893.753)
9       0         C 23.03   359.89      1.00  POINT (2675.896 2014.435)
10      1         C 21.87     0.23      1.00  POINT (2964.876 1769.710)
11      0         C  0.00     0.32      1.00   POINT (812.528 1648.067)
12      1         C 11.14   359.66      1.00   POINT (699.208 1512.340)
13      2         C 23.45   359.84      1.00   POINT (459.445 1139.411)
14      0         C 21.54   180.38      1.00  POINT (2575.061 2523.892)
15      0         C 23.45   180.08      1.00   POINT (1102.869 125.938)
16      0         B 24.55   180.05      1.00  POINT (1590.398 2647.615)
17      0         B 11.29     0.26      1.00   POINT (1321.745 502.309)

         X       Y      Z
0  1918.19  566.73 490.30
1  2190.93 2031.56 255.41
2  2901.89  492.79 650.00
3  2806.50 2017.80 450.00
4   645.35 1647.83 400.00
5   507.46 1396.78 450.00
6   333.88 1138.69 550.00
7  2683.26 2522.93 450.00
```

(continues on next page)

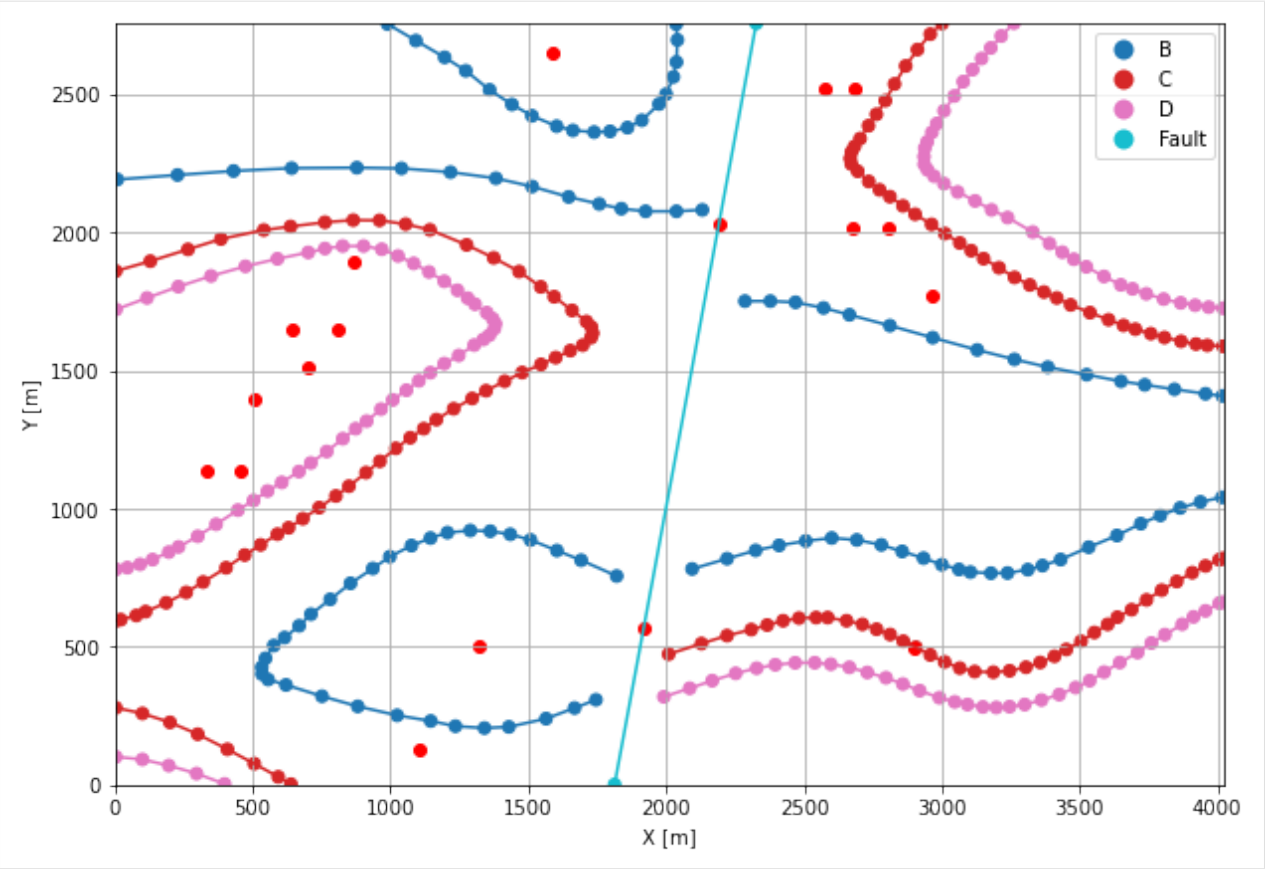
(continued from previous page)

```
8   865.03 1893.75 350.00
9  2675.90 2014.43 350.00
10 2964.88 1769.71 450.00
11  812.53 1648.07 300.00
12  699.21 1512.34 350.00
13  459.45 1139.41 450.00
14 2575.06 2523.89 350.00
15 1102.87  125.94 650.00
16 1590.40 2647.61 450.00
17 1321.74  502.31 450.00
```

Plotting the Orientations

[29]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 4022)
ax.set_ylim(0, 2761)
```

[29]:
```
(0.0, 2761.0)
```



7.18.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[30]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[31]:
```
geo_model = gp.create_model('Model18')
geo_model
```

[31]:
```
Model18  2022-04-05 11:12
```

Initiate Data

[32]:
```
gp.init_data(geo_model, [0, 4022, 0, 2761, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[32]:
```
Model18  2022-04-05 11:12
```

Model Surfaces

[33]:
```
geo_model.surfaces
```

[33]:

	surface	series	order_surfaces	color	id
0	Fault	Default series	1	#015482	1
1	D	Default series	2	#9f0052	2
2	C	Default series	3	#ffbe00	3
3	B	Default series	4	#728f02	4

Mapping the Stack to Surfaces

[34]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('Fault'),
                          'Strata1': ('D', 'C', 'B'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('A')
geo_model.set_is_fault(['Fault1'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[34]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault1	1	Fault	True	True	False
Strata1	2	Erosion	True	False	False

Showing the Number of Data Points

[35]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[35]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Fault	Fault1	1	#527682	1	2	2
1	D	Strata1	1	#9f0052	2	125	6
2	C	Strata1	2	#ffbe00	3	137	8
3	B	Strata1	3	#728f02	4	108	2
4	A	Strata1	4	#443988	5	0	0

Loading Digital Elevation Model

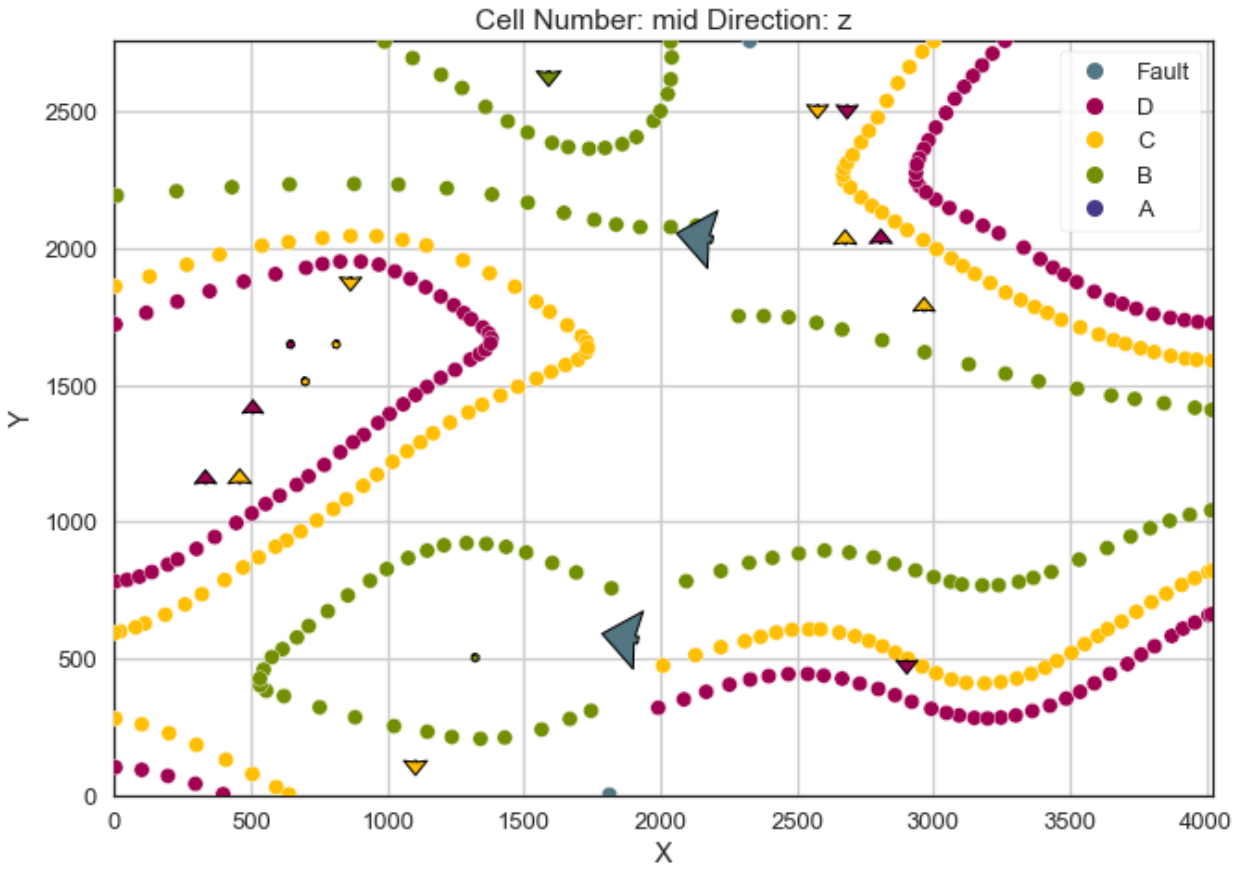
[36]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster18.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

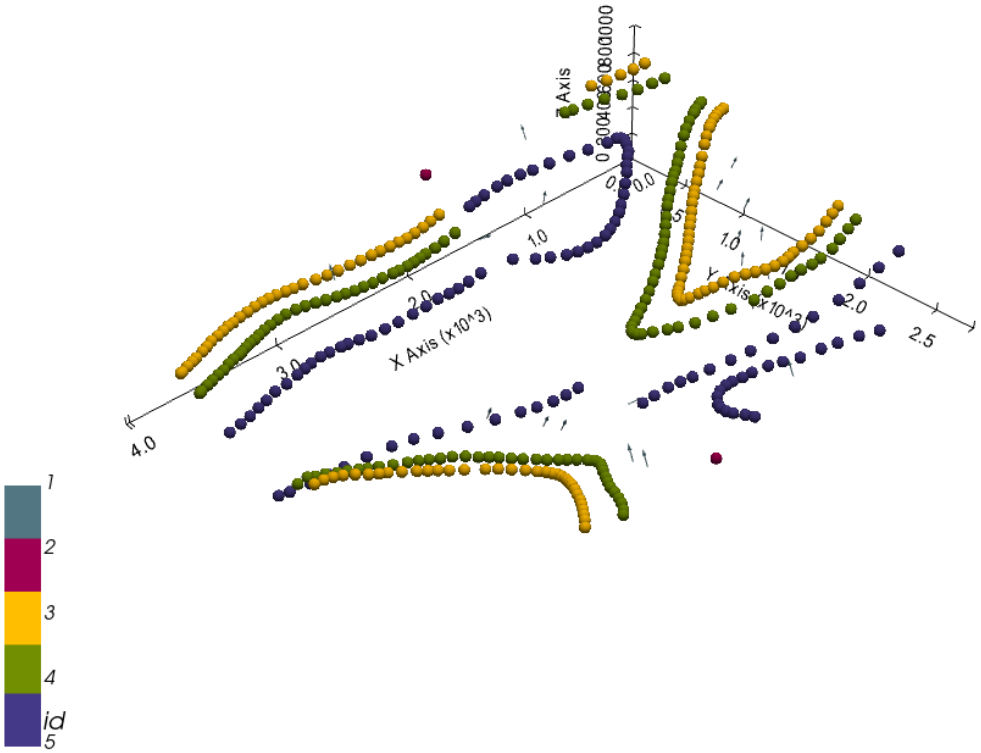
[36]:
```
Grid Object. Values:
array([[  20.11      ,   13.805     ,    5.        ],
       [  20.11      ,   13.805     ,   15.        ],
       [  20.11      ,   13.805     ,   25.        ],
       ...,
       [4016.99751244, 2735.99094203,  715.64483643],
       [4016.99751244, 2745.99456522,  716.50982666],
       [4016.99751244, 2755.99818841,  717.38183594]])
```

Plotting Input Data

[37]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[38]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[38]:
```
<gempy.plot.vista.GemPyToVista at 0x1e57957d130>
```

Setting the Interpolator

[39]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                    values
range             4979.92
```

(continues on next page)

(continued from previous page)

```
$C_o$           590466.79
drift equations    [3, 3]
```

[39]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1e578a9eb20>
```

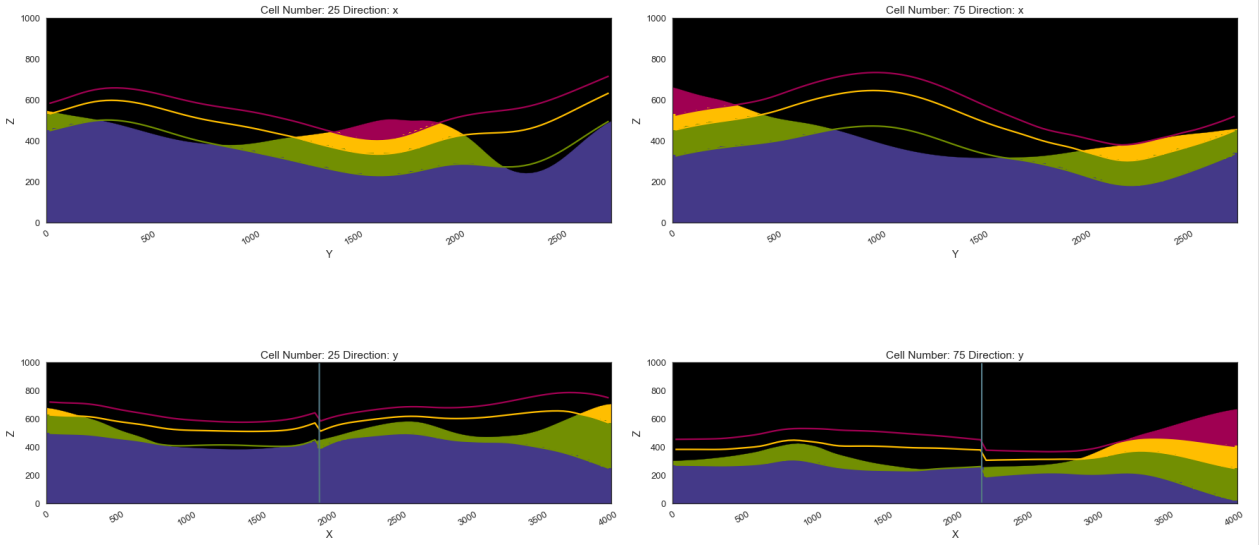
Computing Model

[40]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

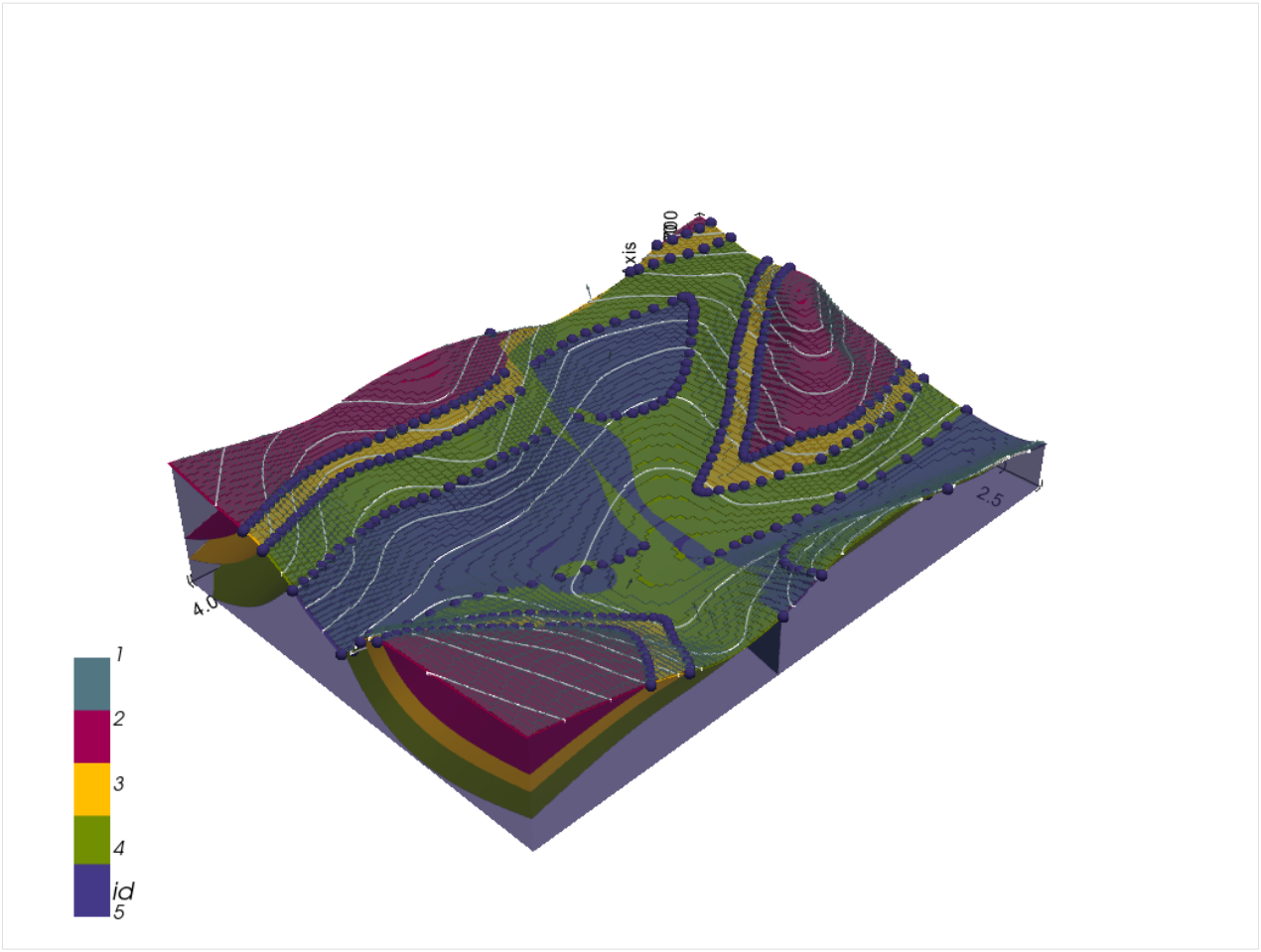
[41]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[41]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1e57c2d2d90>
```



Plotting 3D Model

[42]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.19 Example 19 - Faulted Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3990 m wide (W-E extent) and 2736 m high (N-S extent). The vertical model extent varies from 0 m to 1000 m. The model represents folded layers with a NW-SE striking fault running through the area and a Limestone unit unconformably overlaying the coal measures.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

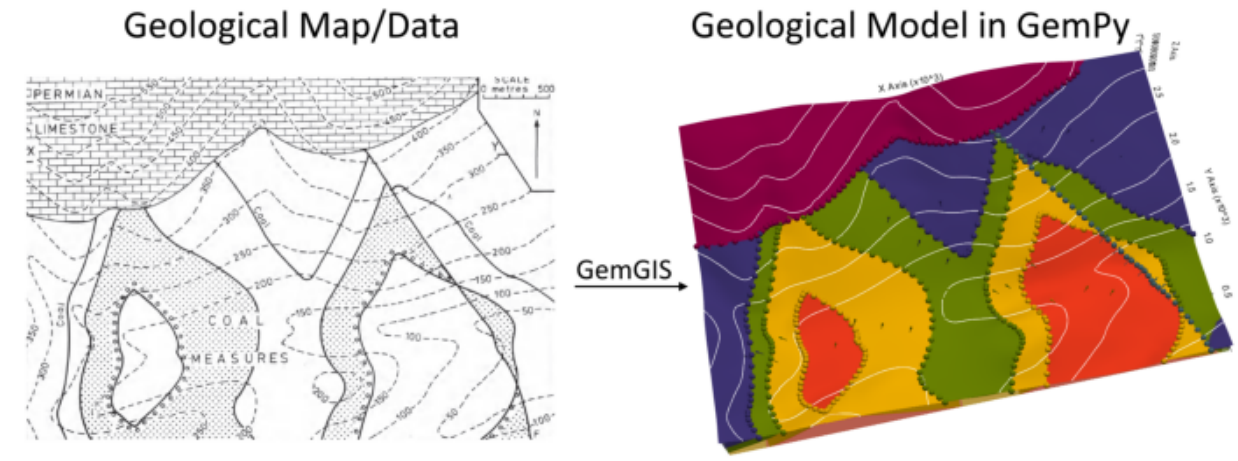
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example19.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.19.1 Licensing

7.19.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.19.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

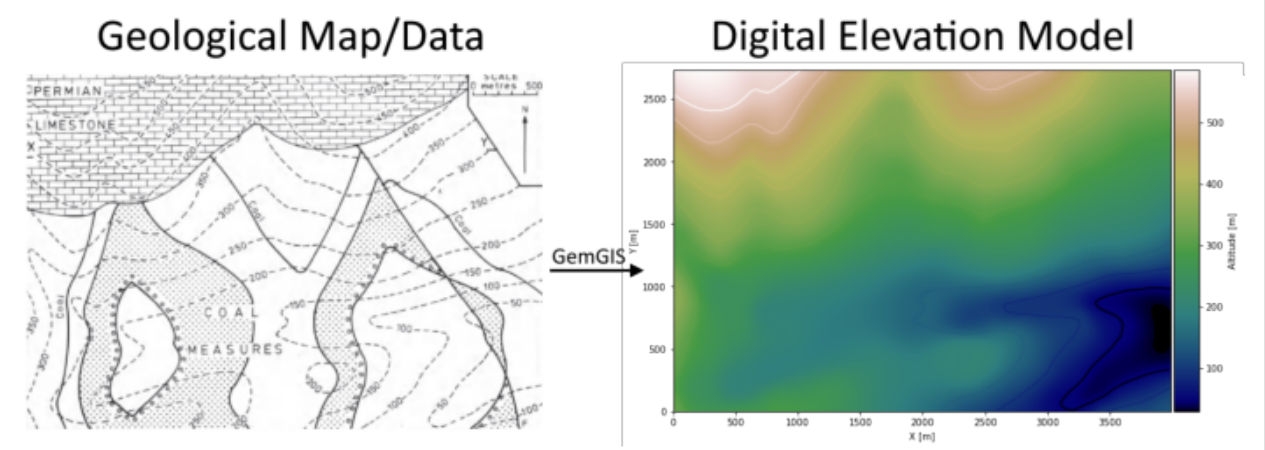
[4]:
```
file_path = 'data/example19/'
gg.download_gemgis_data.download_tutorial_data(filename="example19_faulted_folded_layers.
↪zip", dirpath=file_path)
```

```
Downloading file 'example19_faulted_folded_layers.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2Fexample19_faulted_folded_layers.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\example\data\example19'.
```

7.19.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example19.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo19.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	300	LINESTRING (8.174 150.549, 50.466 70.194, 80.9...
1	None	300	LINESTRING (3.099 239.363, 31.011 304.493, 55...
2	None	550	LINESTRING (3.944 2537.526, 66.114 2492.273, 1...
3	None	500	LINESTRING (3.099 2351.862, 90.221 2275.736, 1...
4	None	450	LINESTRING (3.099 2193.266, 109.252 2093.034, ...

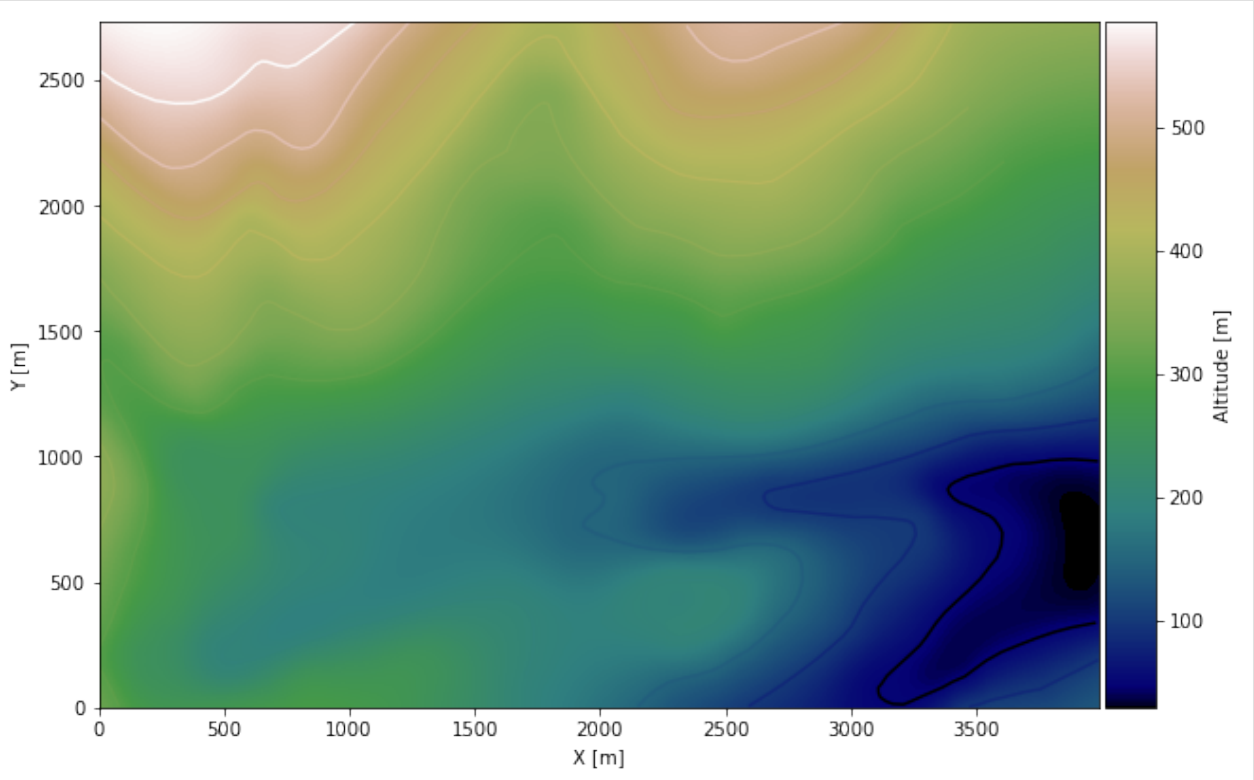
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0, 3990, 0, 2736], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3990)
ax.set_ylim(0, 2736)
```

[8]:
```
(0.0, 2736.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster19.tif', extent = [0, 3990, 0, 2736], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

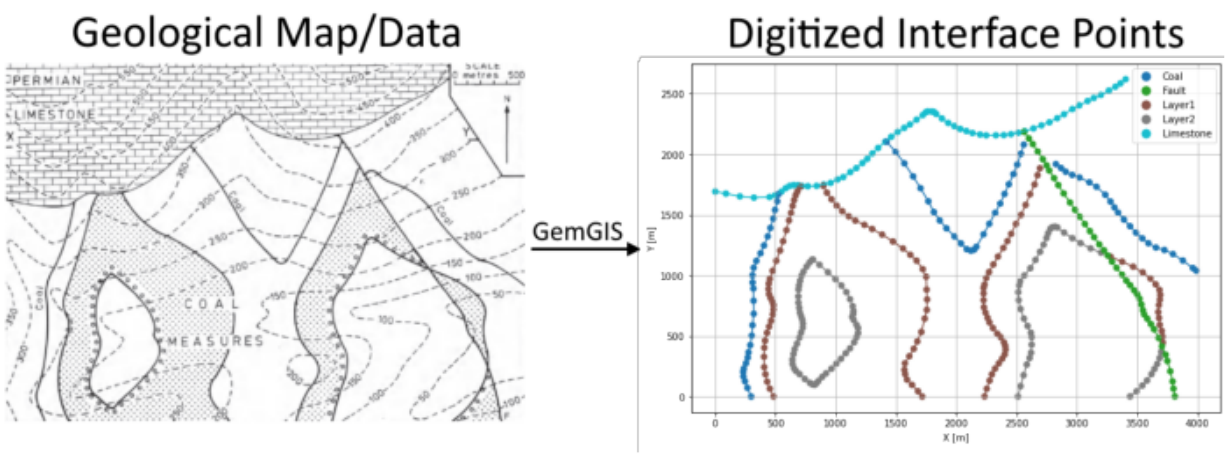
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster19.tif')
```

7.19.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example19.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces19.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Fault	LINESTRING (3811.835 3.901, 3800.205 67.868, 3...
1	None	Coal	LINESTRING (299.990 2.579, 269.064 74.740, 248...
2	None	Coal	LINESTRING (1422.850 2103.184, 1497.390 2019.1...

(continues on next page)

(continued from previous page)

```
3  None       Coal  LINESTRING (2825.632 1919.212, 2868.453 1886.7...
4  None  Limestone  LINESTRING (2.623 1692.420, 90.644 1675.767, 2...
```

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Limestone',
↪'Fault', 'Layer2', 'Layer1', 'Coal'])]
interfaces_coords.head()
```

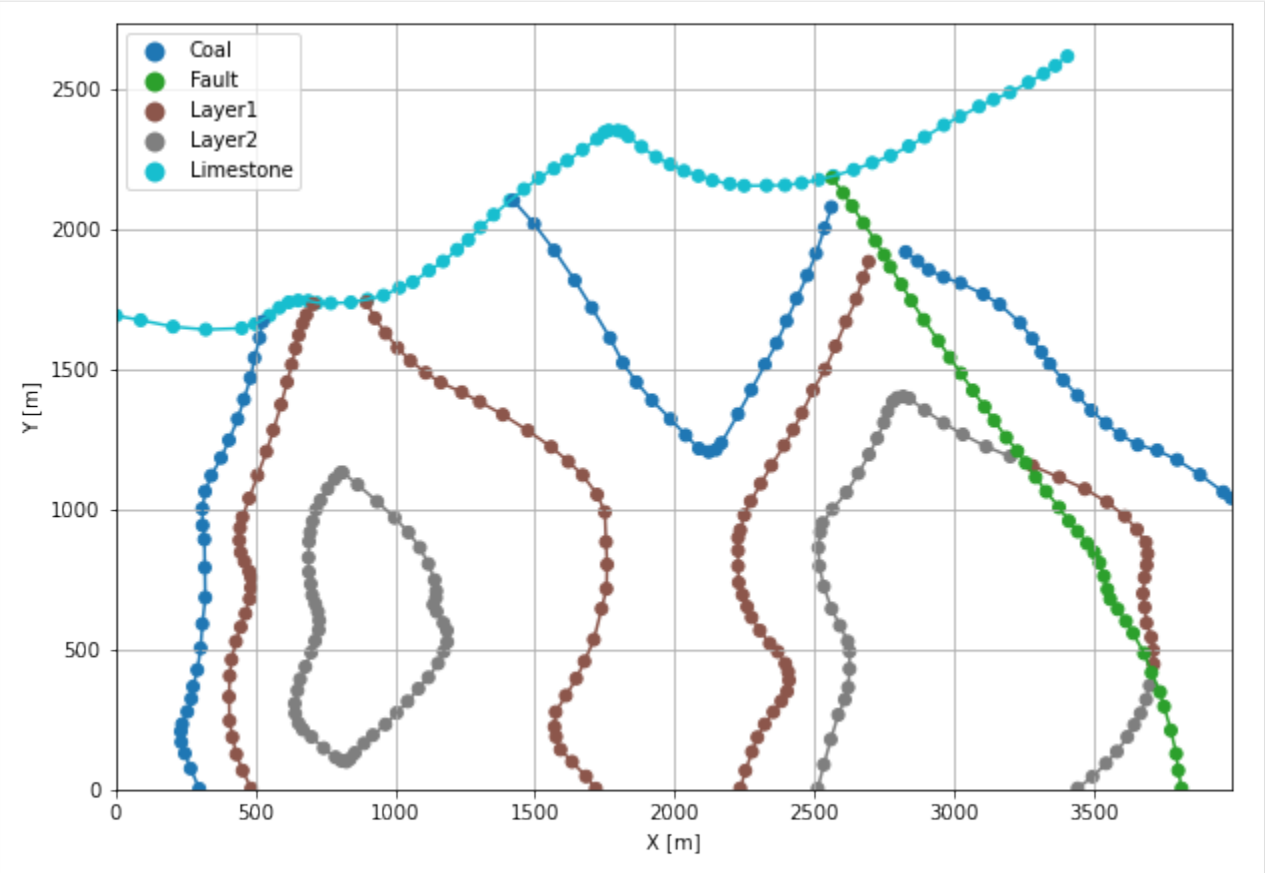
[12]:

	formation	geometry	X	Y	Z
161	Limestone	POINT (2453.526 2165.433)	2453.53	2165.43	411.30
171	Limestone	POINT (3088.307 2439.011)	3088.31	2439.01	416.81
165	Limestone	POINT (2706.091 2237.990)	2706.09	2237.99	420.49
166	Limestone	POINT (2771.908 2263.366)	2771.91	2263.37	418.46
167	Limestone	POINT (2836.932 2297.464)	2836.93	2297.46	419.85

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3990)
ax.set_ylim(0, 2736)
```

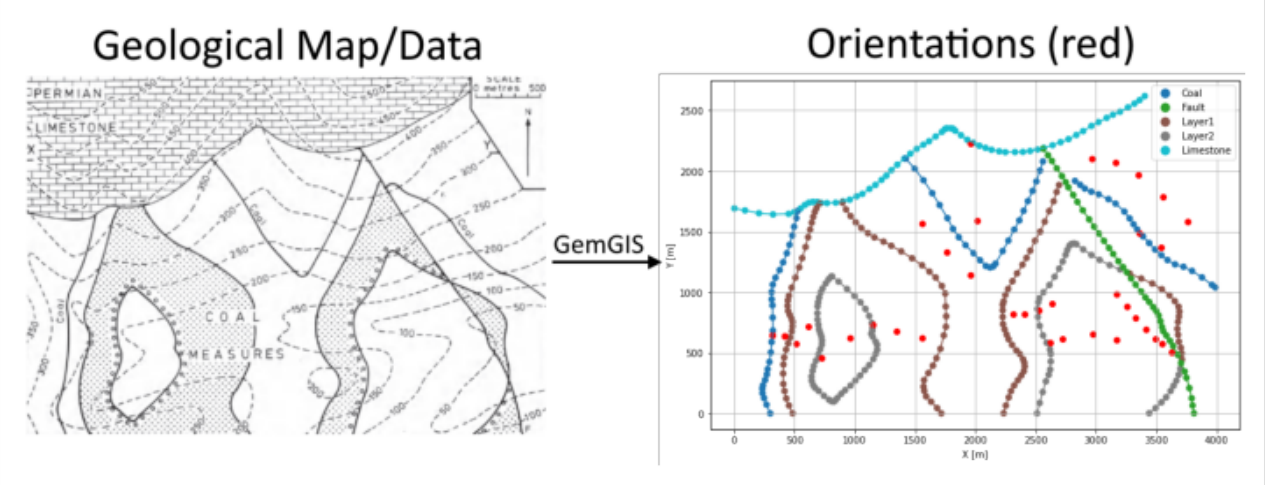
[13]:
```
(0.0, 2736.0)
```



7.19.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example19.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes19.shp')
strikes.head()
```

[15]:

	id	formation	Z	geometry
0	1	Fault	50	LINESTRING (3757.979 278.933, 3593.039 618.328)
1	2	Fault	100	LINESTRING (3800.800 93.375, 3345.629 1038.608)
2	3	Fault	150	LINESTRING (3248.886 1171.829, 3803.972 1.390)
3	4	Fault	200	LINESTRING (3147.384 1309.807, 3770.667 -6.540)
4	5	Fault	250	LINESTRING (3010.991 1501.708, 3731.017 -9.712)

Calculate Orientations for each formation

[16]:
```
orientations_limestone = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Limestone'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_limestone
```

[16]:

	dip	azimuth	Z	geometry	polarity	formation	X	Y
0	8.97	339.97	375.00	POINT (1957.914 2225.303)	1.00	Limestone	1957.91	2225.30

[17]:
```
orientations_fault = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Fault'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_fault
```

[17]:

	dip	azimuth	Z	geometry	polarity	formation	X
0	50.95	64.26	75.00	POINT (3624.362 507.311)	1.00	Fault	3624.36
1	58.76	64.49	125.00	POINT (3549.822 576.300)	1.00	Fault	3549.82
2	56.82	64.65	175.00	POINT (3492.727 619.121)	1.00	Fault	3492.73
3	53.36	64.59	225.00	POINT (3415.015 698.816)	1.00	Fault	3415.01
4	55.58	64.56	275.00	POINT (3329.373 792.784)	1.00	Fault	3329.37
5	65.78	64.73	325.00	POINT (3253.247 881.201)	1.00	Fault	3253.25
6	59.48	64.76	375.00	POINT (3172.363 982.703)	1.00	Fault	3172.36

(continues on next page)

(continued from previous page)

```
       Y
0 507.31
1 576.30
2 619.12
3 698.82
4 792.78
5 881.20
6 982.70
```

[18]:
```
orientations_layer2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer2'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer2
```

[18]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 28.13   269.43 175.00  POINT (2619.061 582.446)      1.00    Layer2 2619.06
1 24.29   269.17 225.00  POINT (2722.941 617.734)      1.00    Layer2 2722.94

       Y
0 582.45
1 617.73
```

[19]:
```
orientations_layer2a = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer2a'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer2a
```

[19]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 13.90    88.28 175.00  POINT (3168.002 606.434)      1.00   Layer2a 3168.00
1 15.16    88.43 225.00  POINT (2971.541 659.960)      1.00   Layer2a 2971.54

       Y
0 606.43
1 659.96
```

[20]:
```
orientations_layer2b = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer2b'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer2b
```

[20]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 14.01    89.59 225.00  POINT (963.913 621.104)      1.00   Layer2b 963.91

       Y
0 621.10
```

[21]:
```
orientations_layer2c = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer2c'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer2c
```

[21]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 24.58   269.89 225.00  POINT (722.451 459.732)      1.00   Layer2c 722.45

       Y
0 459.73
```

[22]:
```
orientations_layer1a = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer1a'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer1a
```

[22]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 29.74   269.93 175.00  POINT (2311.583 820.142)      1.00   Layer1a 2311.58
1 25.64   269.79 225.00  POINT (2410.309 818.159)      1.00   Layer1a 2410.31
2 21.62   269.51 275.00  POINT (2527.670 849.086)      1.00   Layer1a 2527.67
3 28.17   269.31 325.00  POINT (2639.084 904.594)      1.00   Layer1a 2639.08

       Y
0 820.14
1 818.16
2 849.09
3 904.59
```

[23]:
```
orientations_layer1b = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer1b'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer1b
```

[23]:
```
    dip  azimuth      Z                  geometry  polarity formation       X  \
0 13.52    89.97 225.00  POINT (1561.027 623.284)      1.00   Layer1b 1561.03
1 13.31    90.14 275.00  POINT (1349.698 678.991)      1.00   Layer1b 1349.70
2 15.20    90.57 325.00  POINT (1147.488 736.879)      1.00   Layer1b 1147.49

       Y
0 623.28
1 678.99
2 736.88
```

[24]:
```
orientations_layer1c = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer1c'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer1c
```

[24]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 26.25   269.71 275.00  POINT (514.888 575.111)      1.00   Layer1c 514.89
1 28.43   269.71 325.00  POINT (612.623 721.416)      1.00   Layer1c 612.62

       Y
0 575.11
1 721.42
```

[25]:
```
orientations_layer1d = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Layer1d'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_layer1d
```

[25]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 17.60    89.66  75.00  POINT (3539.315 1371.263)      1.00   Layer1d
1 13.66    89.75 125.00  POINT (3356.334 1492.986)      1.00   Layer1d

        X       Y
0 3539.31 1371.26
1 3356.33 1492.99
```

[26]:
```
orientations_coal = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal
```

[26]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 14.02    89.91 225.00  POINT (1955.842 1142.621)      1.00      Coal
1 14.99    89.72 275.00  POINT (1761.056 1331.614)      1.00      Coal
2 13.15    89.66 325.00  POINT (1558.846 1571.094)      1.00      Coal
3  2.56   269.64 375.00  POINT (2010.954 1589.465)      1.00      Coal

        X       Y
0 1955.84 1142.62
1 1761.06 1331.61
2 1558.85 1571.09
3 2010.95 1589.47
```

[27]:
```
orientations_coal1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal1'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal1
```

[27]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 24.80   270.11 275.00  POINT (316.048 650.841)      1.00     Coal1 316.05
1 28.64   269.67 325.00  POINT (421.118 644.100)      1.00     Coal1 421.12

       Y
0 650.84
1 644.10
```

[28]:
```
orientations_coal2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal1'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal2
```

[28]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 24.80   270.11 275.00  POINT (316.048 650.841)      1.00     Coal1 316.05
1 28.64   269.67 325.00  POINT (421.118 644.100)      1.00     Coal1 421.12

       Y
0 650.84
```

(continues on next page)

(continued from previous page)

```
1 644.10
```

[29]:
```
orientations_coal3 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal3'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal3
```

[29]:
```
    dip  azimuth      Z                   geometry  polarity formation  \
0 13.05    89.95 125.00  POINT (3760.576 1586.625)      1.00     Coal3
1 14.01    90.25 175.00  POINT (3551.229 1787.645)      1.00     Coal3
2 14.47    90.32 225.00  POINT (3352.588 1968.841)      1.00     Coal3
3 27.49    90.07 300.00  POINT (3158.704 2070.343)      1.00     Coal3
4 14.49    89.80 375.00  POINT (2964.424 2098.493)      1.00     Coal3

        X       Y
0 3760.58 1586.62
1 3551.23 1787.65
2 3352.59 1968.84
3 3158.70 2070.34
4 2964.42 2098.49
```

Merging Orientations

[30]:
```
import pandas as pd
orientations = pd.concat([orientations_limestone, orientations_fault, orientations_
↪layer2, orientations_layer2a, orientations_layer2b, orientations_layer2c, orientations_
↪layer1a, orientations_layer1b, orientations_layer1c,  orientations_layer1d, ␣
↪orientations_coal,  orientations_coal1, orientations_coal2,  orientations_coal3]).
↪reset_index()
orientations['formation'] = ['Limestone', 'Fault', 'Fault', 'Fault', 'Fault', 'Fault',
↪'Fault', 'Fault', 'Layer2', 'Layer2', 'Layer2', 'Layer2', 'Layer2', 'Layer2', 'Layer1',
↪ 'Layer1', 'Layer1', 'Layer1', 'Layer1', 'Layer1', 'Layer1', 'Layer1',
↪'Layer1', 'Layer1', 'Coal', 'Coal', 'Coal', 'Coal', 'Coal', 'Coal', 'Coal', 'Coal',
↪'Coal', 'Coal', 'Coal', 'Coal', 'Coal']
orientations = orientations[orientations['formation'].isin(['Limestone', 'Fault', 'Layer2
↪', 'Layer1', 'Coal'])]
orientations
```

[30]:
```
    index   dip  azimuth      Z                   geometry  polarity  \
0       0  8.97   339.97 375.00  POINT (1957.914 2225.303)      1.00
1       0 50.95    64.26  75.00   POINT (3624.362 507.311)      1.00
2       1 58.76    64.49 125.00   POINT (3549.822 576.300)      1.00
3       2 56.82    64.65 175.00   POINT (3492.727 619.121)      1.00
4       3 53.36    64.59 225.00   POINT (3415.015 698.816)      1.00
5       4 55.58    64.56 275.00   POINT (3329.373 792.784)      1.00
6       5 65.78    64.73 325.00   POINT (3253.247 881.201)      1.00
7       6 59.48    64.76 375.00   POINT (3172.363 982.703)      1.00
8       0 28.13   269.43 175.00   POINT (2619.061 582.446)      1.00
9       1 24.29   269.17 225.00   POINT (2722.941 617.734)      1.00
10      0 13.90    88.28 175.00   POINT (3168.002 606.434)      1.00
11      1 15.16    88.43 225.00   POINT (2971.541 659.960)      1.00
```

(continues on next page)

(continued from previous page)

```
12      0 14.01    89.59 225.00    POINT (963.913 621.104)      1.00
13      0 24.58   269.89 225.00    POINT (722.451 459.732)      1.00
14      0 29.74   269.93 175.00   POINT (2311.583 820.142)      1.00
15      1 25.64   269.79 225.00   POINT (2410.309 818.159)      1.00
16      2 21.62   269.51 275.00   POINT (2527.670 849.086)      1.00
17      3 28.17   269.31 325.00   POINT (2639.084 904.594)      1.00
18      0 13.52    89.97 225.00   POINT (1561.027 623.284)      1.00
19      1 13.31    90.14 275.00   POINT (1349.698 678.991)      1.00
20      2 15.20    90.57 325.00   POINT (1147.488 736.879)      1.00
21      0 26.25   269.71 275.00    POINT (514.888 575.111)      1.00
22      1 28.43   269.71 325.00    POINT (612.623 721.416)      1.00
23      0 17.60    89.66  75.00  POINT (3539.315 1371.263)      1.00
24      1 13.66    89.75 125.00  POINT (3356.334 1492.986)      1.00
25      0 14.02    89.91 225.00  POINT (1955.842 1142.621)      1.00
26      1 14.99    89.72 275.00  POINT (1761.056 1331.614)      1.00
27      2 13.15    89.66 325.00  POINT (1558.846 1571.094)      1.00
28      3  2.56   269.64 375.00  POINT (2010.954 1589.465)      1.00
29      0 24.80   270.11 275.00    POINT (316.048 650.841)      1.00
30      1 28.64   269.67 325.00    POINT (421.118 644.100)      1.00
31      0 24.80   270.11 275.00    POINT (316.048 650.841)      1.00
32      1 28.64   269.67 325.00    POINT (421.118 644.100)      1.00
33      0 13.05    89.95 125.00  POINT (3760.576 1586.625)      1.00
34      1 14.01    90.25 175.00  POINT (3551.229 1787.645)      1.00
35      2 14.47    90.32 225.00  POINT (3352.588 1968.841)      1.00
36      3 27.49    90.07 300.00  POINT (3158.704 2070.343)      1.00
37      4 14.49    89.80 375.00  POINT (2964.424 2098.493)      1.00

    formation       X       Y
0   Limestone 1957.91 2225.30
1       Fault 3624.36  507.31
2       Fault 3549.82  576.30
3       Fault 3492.73  619.12
4       Fault 3415.01  698.82
5       Fault 3329.37  792.78
6       Fault 3253.25  881.20
7       Fault 3172.36  982.70
8      Layer2 2619.06  582.45
9      Layer2 2722.94  617.73
10     Layer2 3168.00  606.43
11     Layer2 2971.54  659.96
12     Layer2  963.91  621.10
13     Layer2  722.45  459.73
14     Layer1 2311.58  820.14
15     Layer1 2410.31  818.16
16     Layer1 2527.67  849.09
17     Layer1 2639.08  904.59
18     Layer1 1561.03  623.28
19     Layer1 1349.70  678.99
20     Layer1 1147.49  736.88
21     Layer1  514.89  575.11
22     Layer1  612.62  721.42
23     Layer1 3539.31 1371.26
```

(continues on next page)

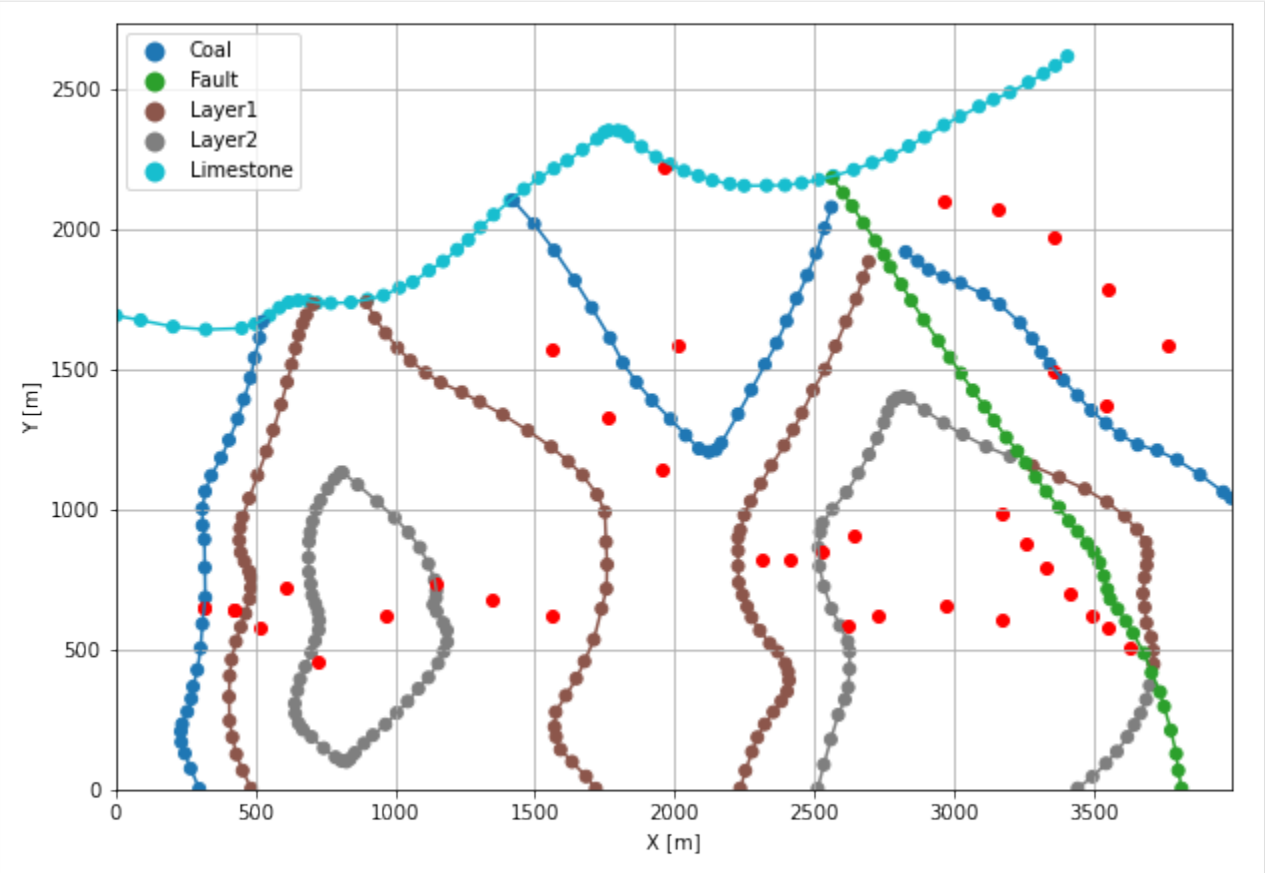
(continued from previous page)

```
24     Layer1 3356.33 1492.99
25       Coal 1955.84 1142.62
26       Coal 1761.06 1331.61
27       Coal 1558.85 1571.09
28       Coal 2010.95 1589.47
29       Coal  316.05  650.84
30       Coal  421.12  644.10
31       Coal  316.05  650.84
32       Coal  421.12  644.10
33       Coal 3760.58 1586.62
34       Coal 3551.23 1787.65
35       Coal 3352.59 1968.84
36       Coal 3158.70 2070.34
37       Coal 2964.42 2098.49
```

Plotting the Orientations

[31]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3990)
ax.set_ylim(0, 2736)
```

[31]:
```
(0.0, 2736.0)
```



7.19.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[32]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[33]:
```
geo_model = gp.create_model('Model18')
geo_model
```

[33]:
```
Model18  2022-04-05 11:16
```

Initiate Data

[34]:
```
gp.init_data(geo_model, [0, 3990, 0, 2736, 0, 1000], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[34]:
```
Model18  2022-04-05 11:16
```

Model Surfaces

[35]:
```
geo_model.surfaces
```

[35]:
```
     surface          series  order_surfaces    color  id
0  Limestone  Default series               1  #015482   1
1     Layer2  Default series               2  #9f0052   2
2     Layer1  Default series               3  #ffbe00   3
3      Fault  Default series               4  #728f02   4
4       Coal  Default series               5  #443988   5
```

Mapping the Stack to Surfaces

[36]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('Fault'),
                          'Strata1': ('Limestone'),
                          'Strata2': ('Coal', 'Layer1', 'Layer2'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_is_fault(['Fault1'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[36]:
```
         order_series BottomRelation  isActive  isFault  isFinite
Fault1              1          Fault      True     True     False
Strata1             2        Erosion      True    False     False
Strata2             3        Erosion      True    False     False
```

Showing the Number of Data Points

[37]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[37]:
```
     surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
3      Fault   Fault1               1  #527682   1                 41
↪7
0  Limestone  Strata1               1  #9f0052   2                 63
↪1
```

(continues on next page)

(continued from previous page)

```
1     Layer2  Strata2               1  #ffbe00   3                 97                 ␣
↪6
2     Layer1  Strata2               2  #728f02   4                115                 ␣
↪11
4       Coal  Strata2               3  #443988   5                 74                 ␣
↪13
5   Basement  Strata2               4  #ff3f20   6                  0                 ␣
↪0
```

Loading Digital Elevation Model

[38]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster19.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[38]:
```
Grid Object. Values:
array([[  19.95      ,   13.68      ,    5.        ],
       [  19.95      ,   13.68      ,   15.        ],
       [  19.95      ,   13.68      ,   25.        ],
       ...,
       [3985.        , 2711.03649635,  363.65274048],
       [3985.        , 2721.02189781,  365.24542236],
       [3985.        , 2731.00729927,  366.84155273]])
```

Defining Custom Section

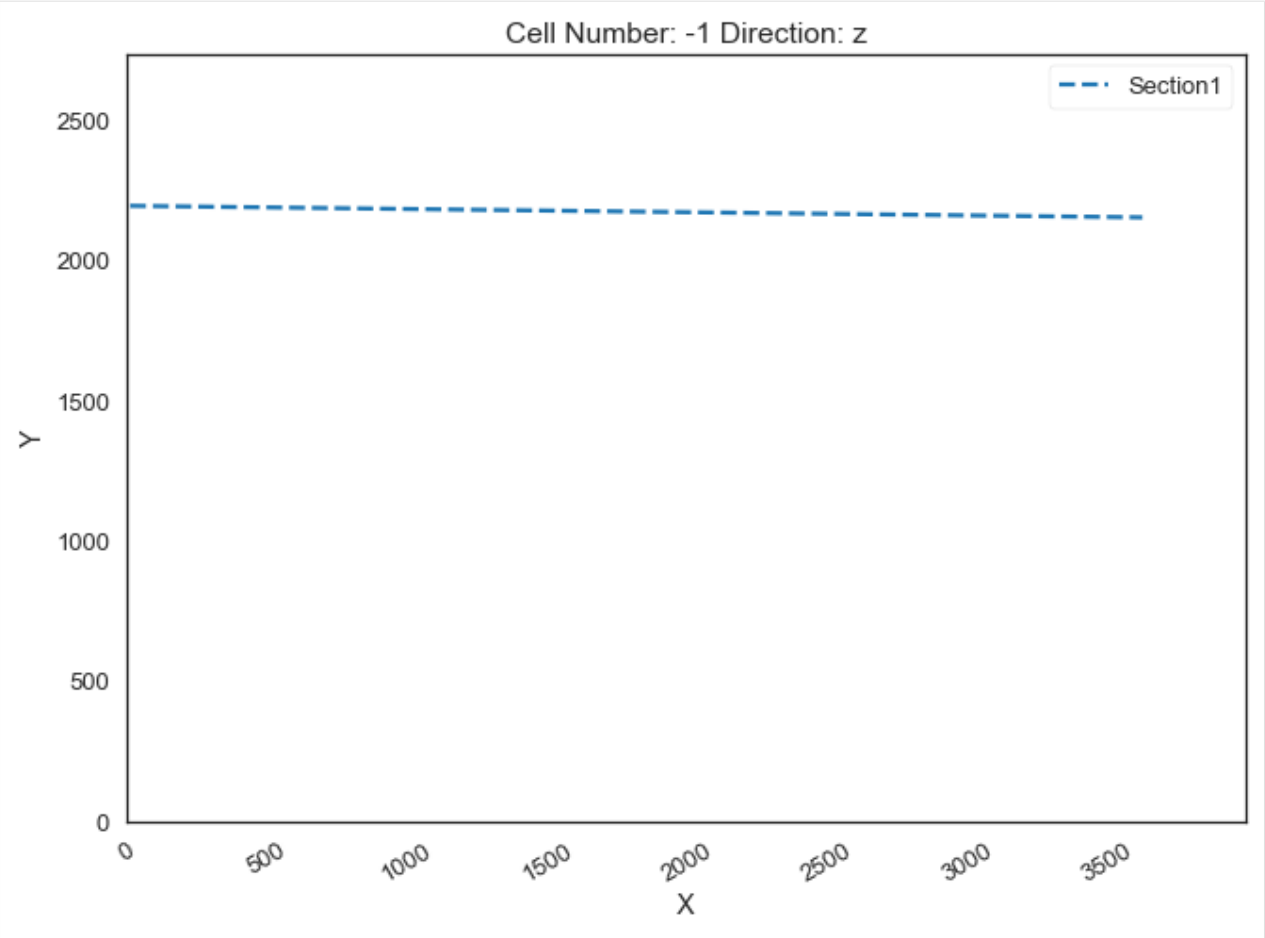
[39]:
```
custom_section = gpd.read_file(file_path + 'customsection19.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='name')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[39]:
```
                                          start                                    ␣
↪stop resolution    dist
Section1  [10.354283343821066, 2194.773034267066]  [3623.1721887321155, 2153.
↪538062563776]  [100, 80] 3613.05
```

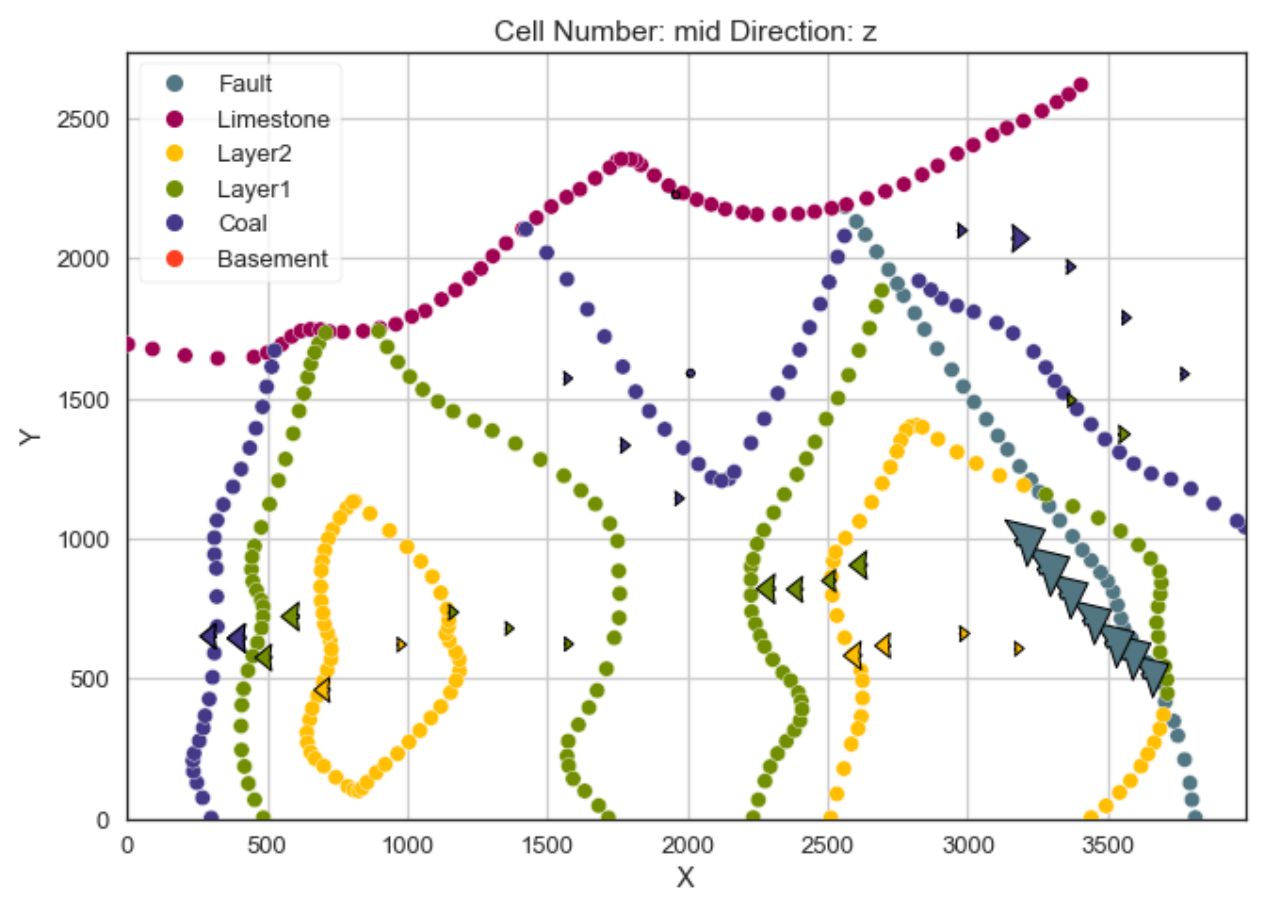
[40]:
```
gp.plot.plot_section_traces(geo_model)
```

[40]:
```
<gempy.plot.visualization_2d.Plot2D at 0x2666448b100>
```

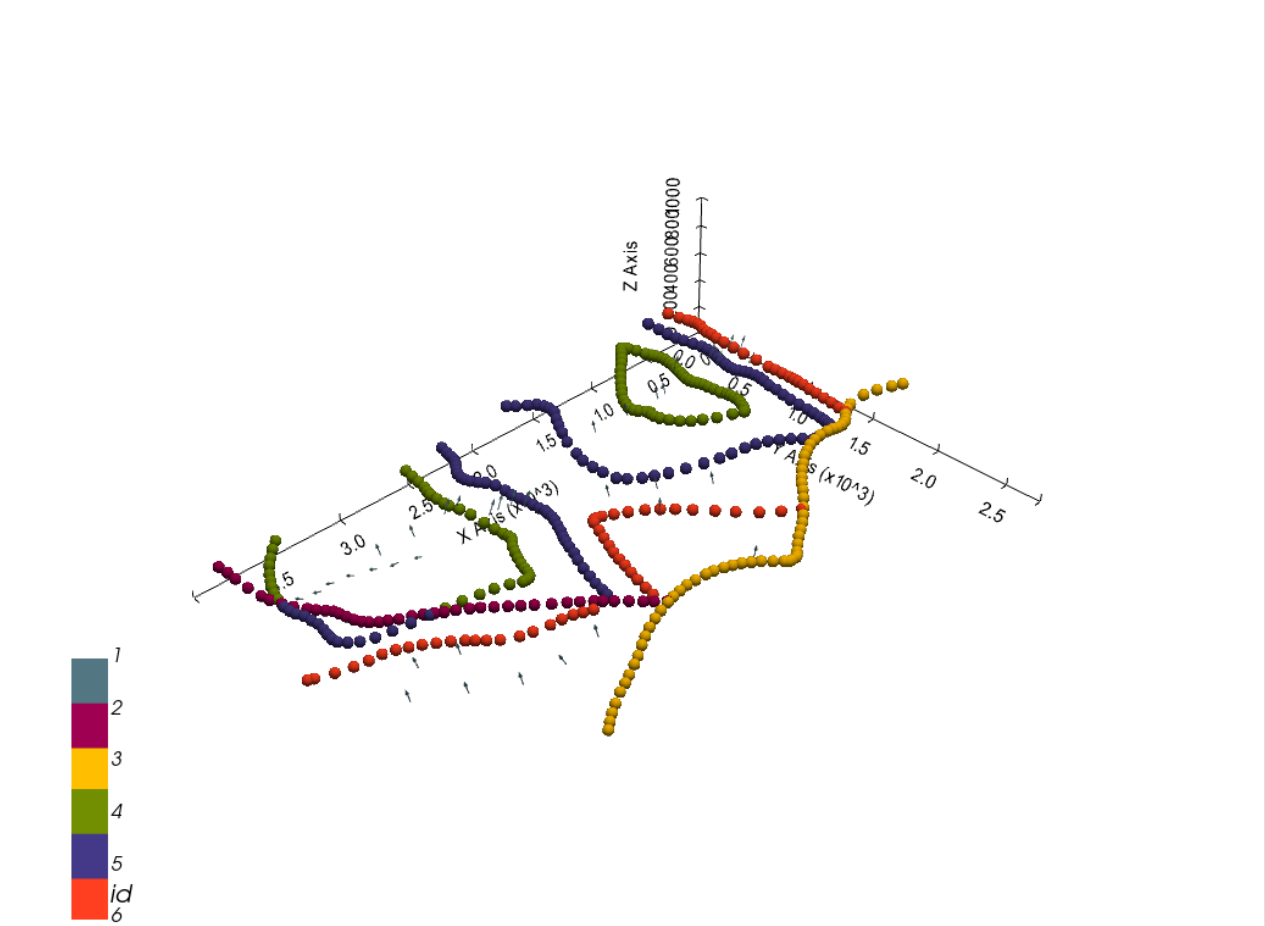


Plotting Input Data

[41]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[42]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[42]:
```
<gempy.plot.vista.GemPyToVista at 0x26662cf3490>
```

Setting the Interpolator

[43]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                      values
range               4940.22
$C_o$             581090.38
drift equations  [3, 3, 3]
```

[43]:
```
<gempy.core.interpolator.InterpolatorModel at 0x26662201100>
```

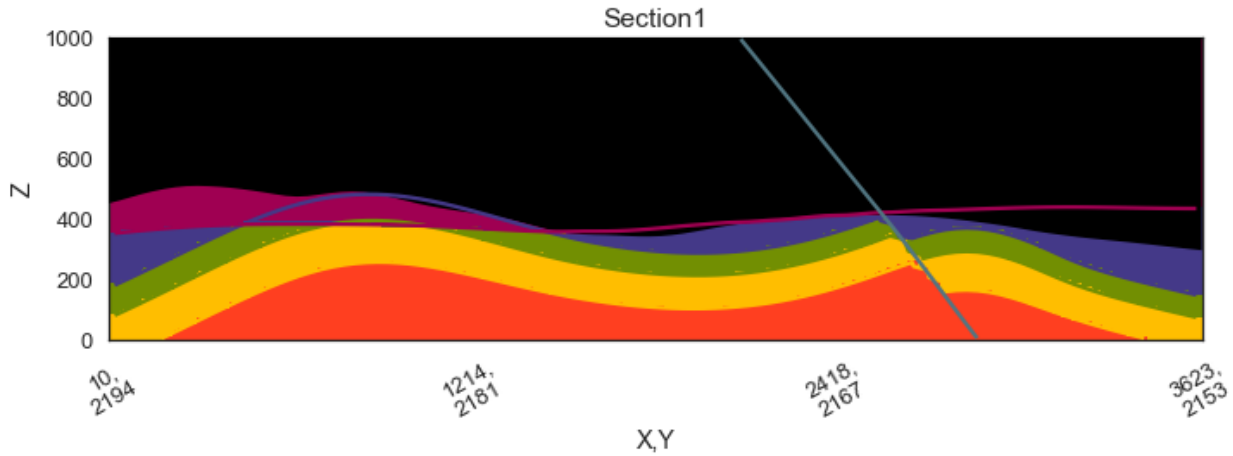
Computing Model

[44]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

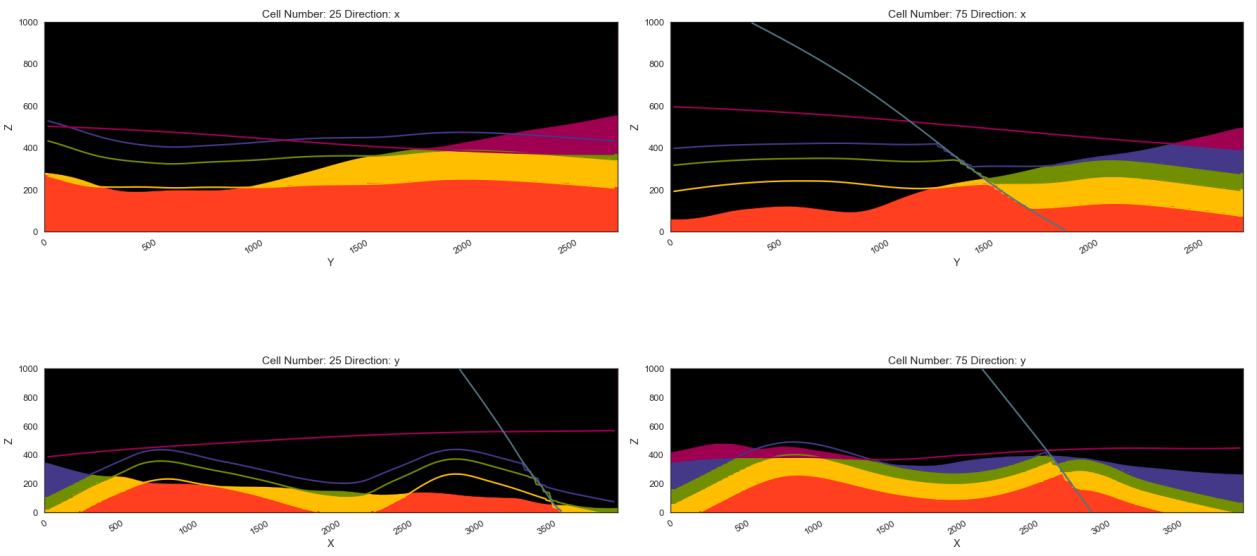
[45]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[45]:
```
<gempy.plot.visualization_2d.Plot2D at 0x26664a9a0d0>
```



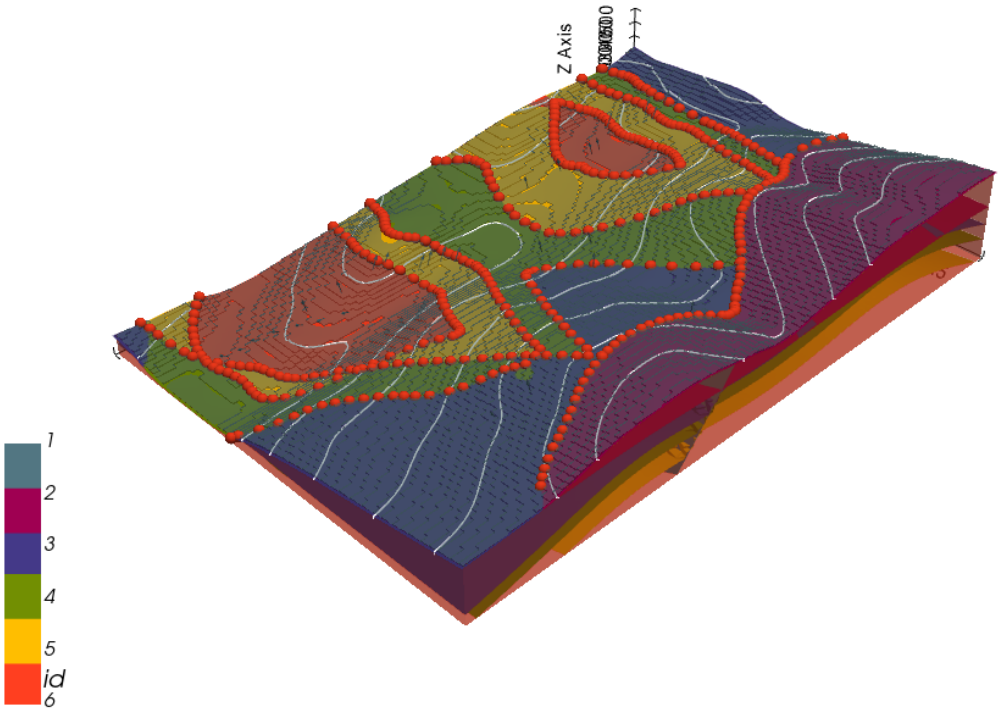
[46]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[46]:
```
<gempy.plot.visualization_2d.Plot2D at 0x26664aa18b0>
```



Plotting 3D Model

[47]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:
```

```

7.20 Example 20 - Sill

This example will show how to convert the geological map below using GemGIS to a GemPy model. This example is based on digitized data. The area is 1381 m wide (W-E extent) and 1768 m high (N-S extent). The vertical model extent varies between -500 m and 250 m. The model represents a wedge shaped sill that was encountered in boreholes.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the GemPy model. The contour lines were also digitized and will be interpolated with GemGIS to create a topography for the model.

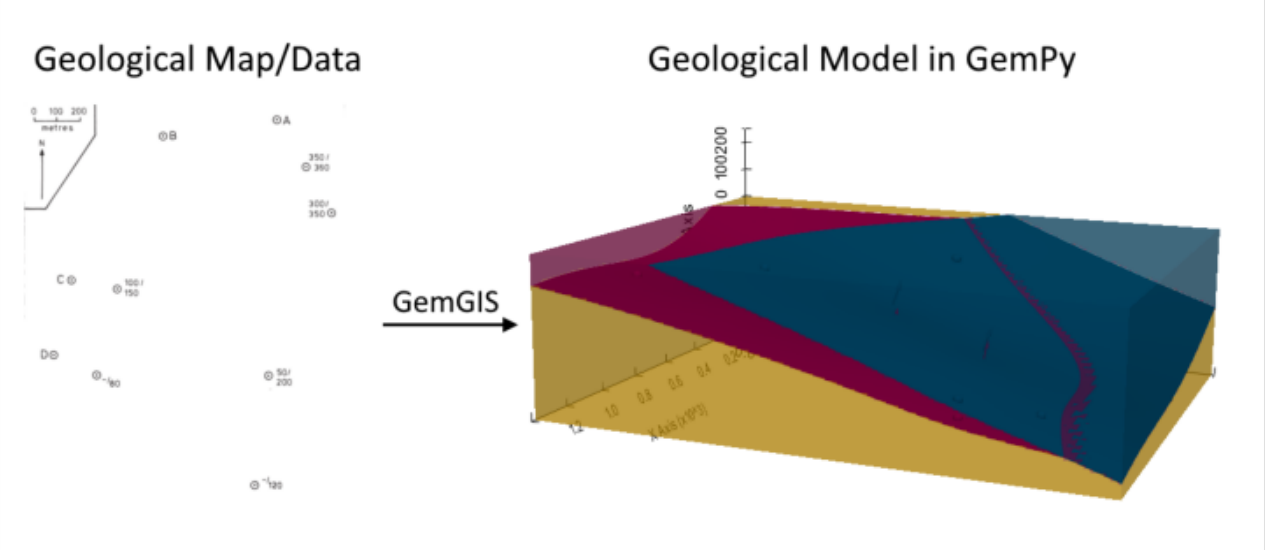
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example20.png')
```

(continues on next page)

(continued from previous page)

```
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.20.1 Licensing

7.20.2 Import GemGIS

If you have installed `GemGIS` via pip and conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.20.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example20/'
gg.download_gemgis_data.download_tutorial_data(filename="example20_sill.zip",␣
↪dirpath=file_path)
```

```
Downloading file 'example20_sill.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample20_sill.zip' to 'C:\Users\ale93371\Documents\
↪gemgis\docs\getting_started\example\data\example20'.
```

7.20.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by creating a NumPy array containing the height values. For this example, the height is 0, meaning that a flat topography at sea level is present.

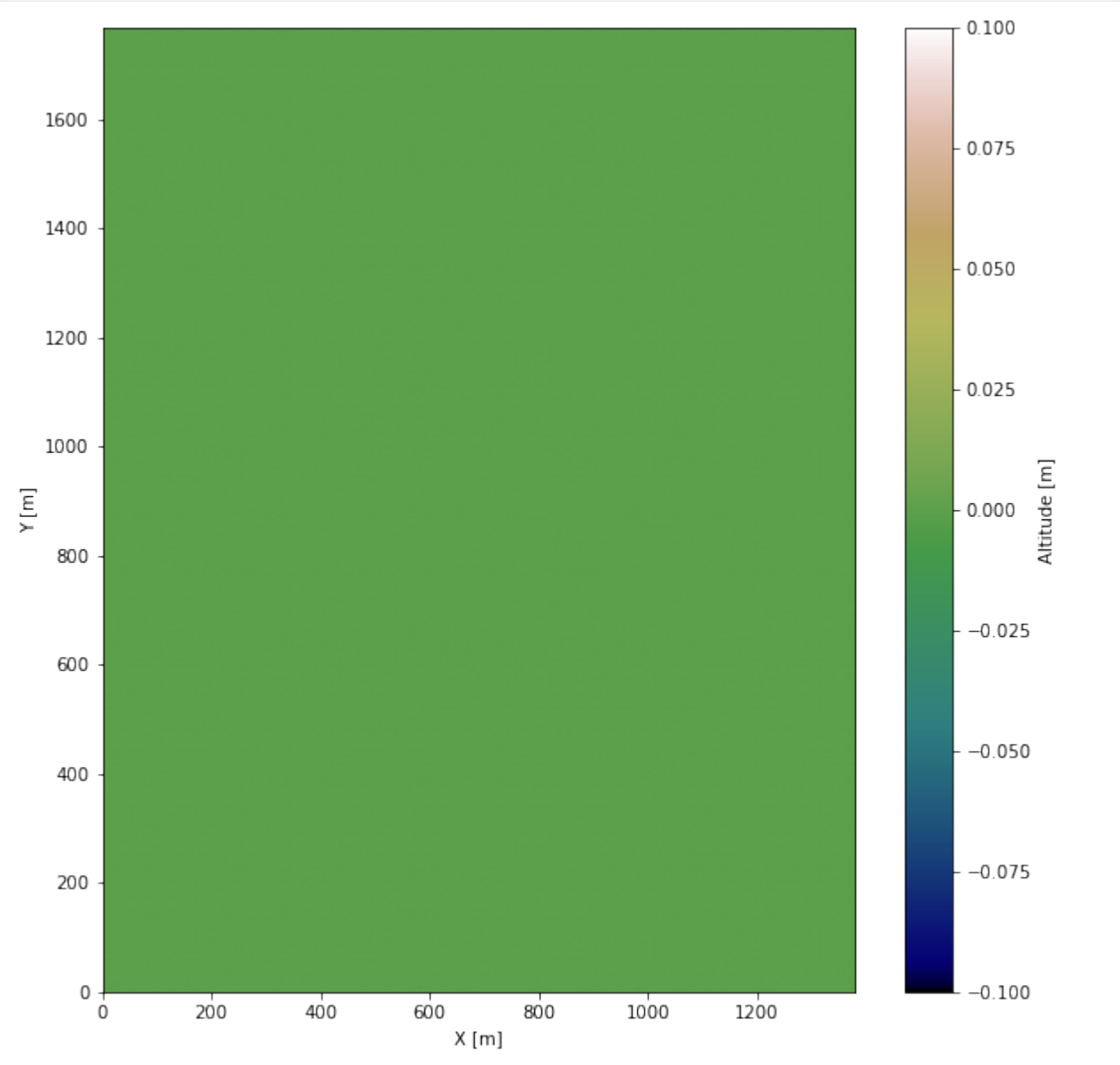
Creating the raster

[5]:
```
import numpy as np
topo_raster = np.zeros((138, 176))
```

Plotting the raster

[6]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
im = plt.imshow(topo_raster, origin='lower', extent=[0, 1381, 0, 1768], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 1381)
ax.set_ylim(0, 1768)
```

[6]:
```
(0.0, 1768.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster20.tif', extent = [0, 1381, 0, 1768], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

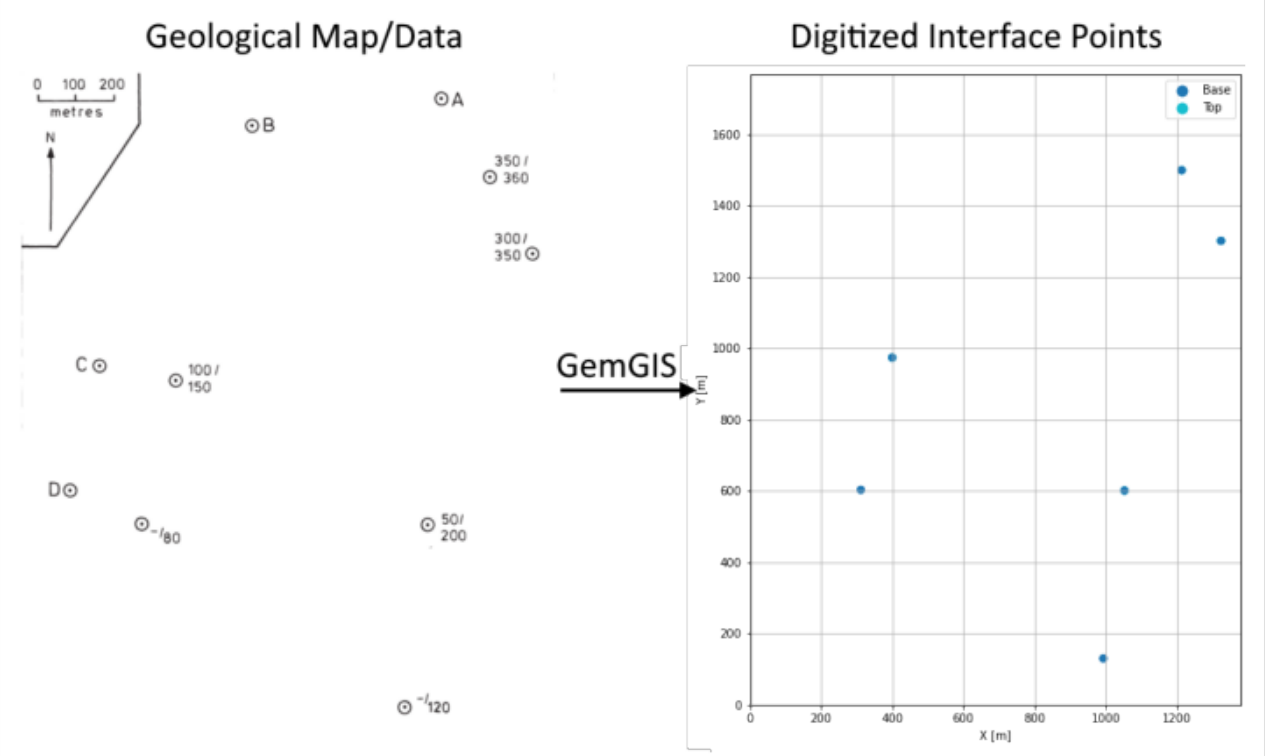
The previously computed and saved raster can now be opened using rasterio.

[7]:
```
topo_raster = rasterio.open(file_path + 'raster20.tif')
```

7.20.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[8]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example20.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[9]:
```
interfaces = gpd.read_file(file_path + 'interfaces20.shp')
interfaces.head()
```

[9]:

	id	formation	Z	geometry
0	None	Base	-80	POINT (311.718 603.282)
1	None	Base	-120	POINT (993.233 130.266)
2	None	Top	-50	POINT (1052.804 601.233)

(continues on next page)

(continued from previous page)

```
3  None      Base -200  POINT (1052.804 601.233)
4  None       Top -100   POINT (399.981 974.235)
```

Extracting Z coordinate from Digital Elevation Model

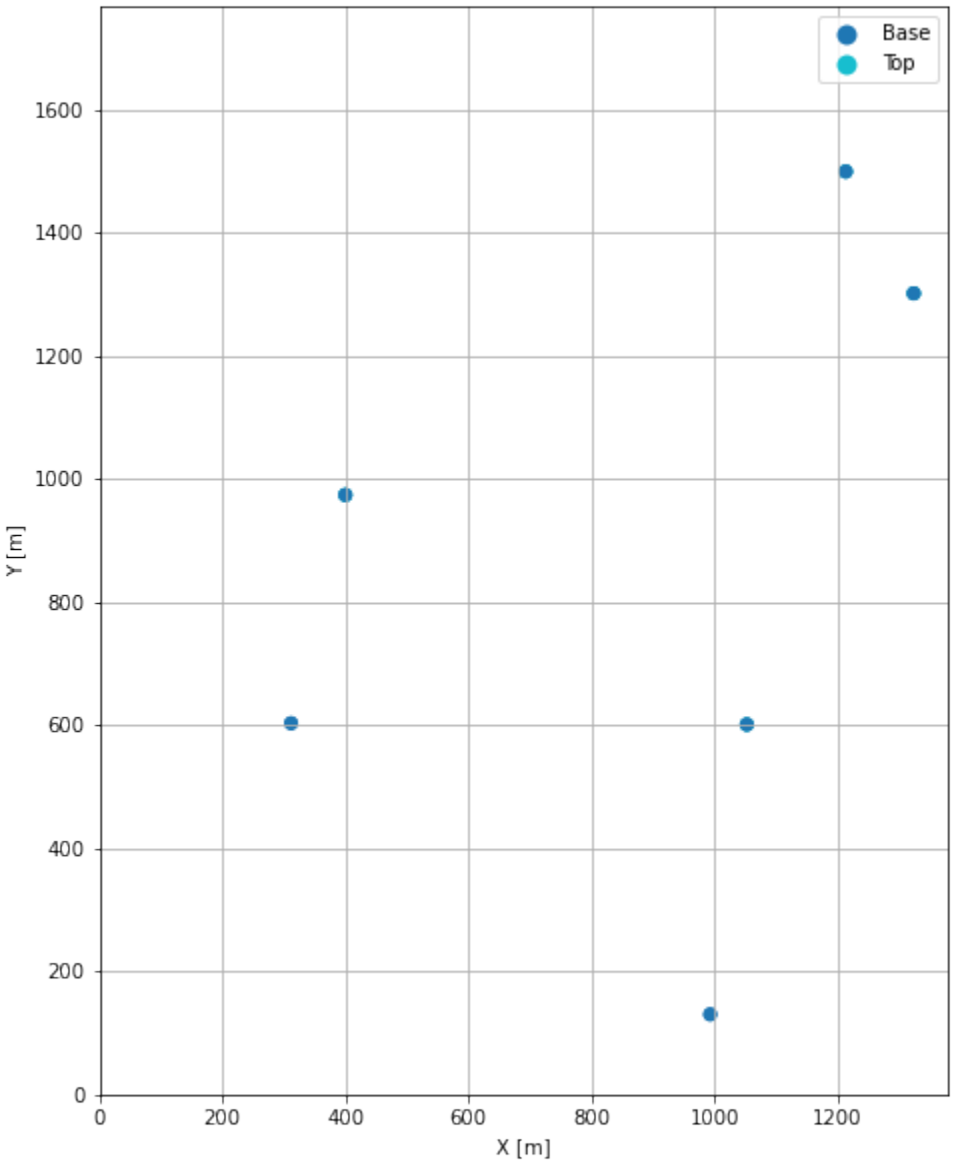
[10]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=None)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Base', 'Top
↪'])]
interfaces_coords.head()
```

[10]:
```
  formation       Z                   geometry       X       Y
2       Top  -50.00   POINT (1052.804 601.233) 1052.80  601.23
4       Top -100.00    POINT (399.981 974.235)  399.98  974.24
6       Top -300.00  POINT (1324.290 1301.740) 1324.29 1301.74
8       Top -350.00  POINT (1213.755 1499.718) 1213.76 1499.72
0      Base  -80.00    POINT (311.718 603.282)  311.72  603.28
```

Plotting the Interface Points

[11]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 1381)
ax.set_ylim(0, 1768)
```

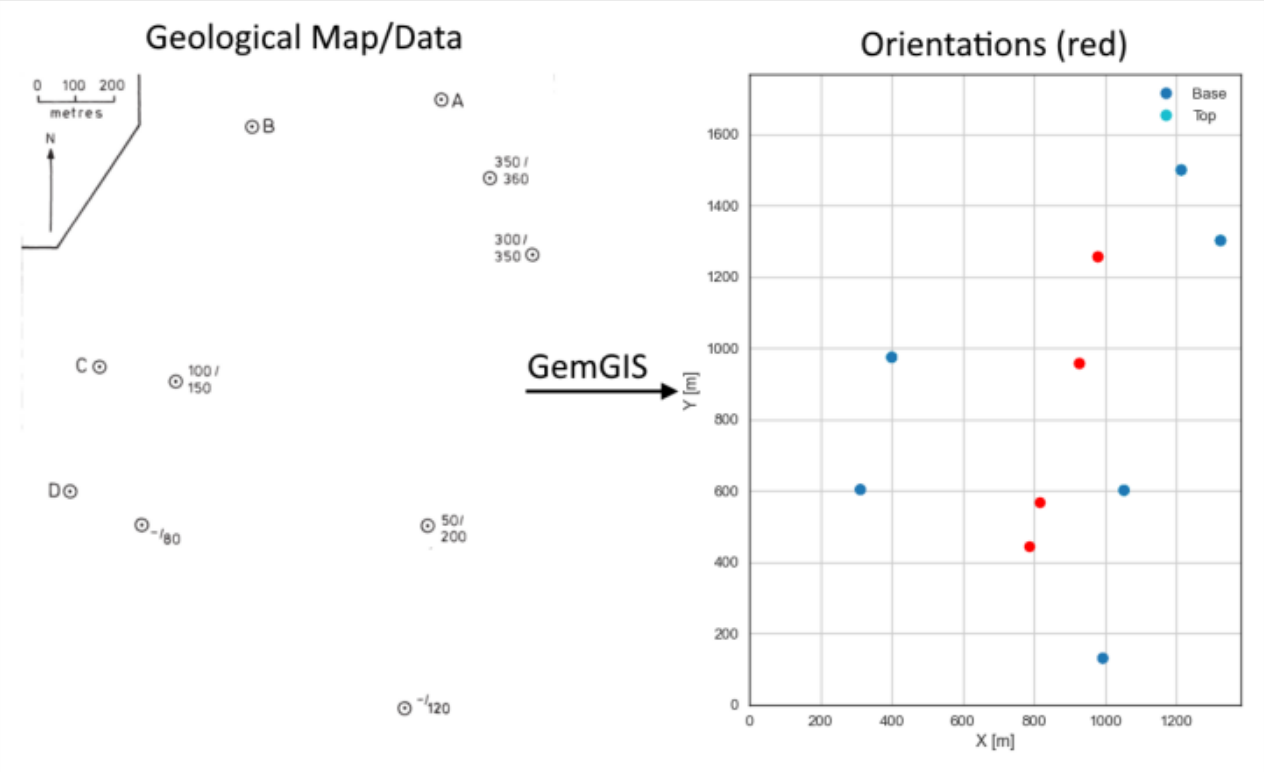
[11]:
```
(0.0, 1768.0)
```



7.20.6 Orientations from Borehole Observations

Orientations of the sill will be calculated as it was a three point problem with `calculate_orientation_for_three_point_problem()`.

[12]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example20.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[13]:
```
interfaces
```

[13]:
```
     id formation    Z                    geometry
0  None      Base  -80    POINT (311.718 603.282)
1  None      Base -120    POINT (993.233 130.266)
2  None       Top  -50   POINT (1052.804 601.233)
3  None      Base -200   POINT (1052.804 601.233)
4  None       Top -100    POINT (399.981 974.235)
5  None      Base -150    POINT (399.981 974.235)
6  None       Top -300  POINT (1324.290 1301.740)
7  None      Base -350  POINT (1324.290 1301.740)
8  None       Top -350  POINT (1213.755 1499.718)
9  None      Base -360  POINT (1213.755 1499.718)
```

Calculate Orientations for each formation

[14]:
```
interfaces_base = interfaces[interfaces['formation'] == 'Base'].reset_index()
interfaces_base
```

[14]:
```
   index    id formation    Z                    geometry
0      0  None      Base  -80    POINT (311.718 603.282)
1      1  None      Base -120    POINT (993.233 130.266)
2      3  None      Base -200   POINT (1052.804 601.233)
3      5  None      Base -150    POINT (399.981 974.235)
4      7  None      Base -350  POINT (1324.290 1301.740)
5      9  None      Base -360  POINT (1213.755 1499.718)
```

[15]:
```
orientations1 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪base.loc[:2])
orientations1['Z'] = orientations1['Z'].astype(float)
orientations1['azimuth'] = orientations1['azimuth'].astype(float)
orientations1['dip'] = orientations1['dip'].astype(float)
orientations1['dip'] = 180 - orientations1['dip']
orientations1['azimuth'] = 180 - orientations1['azimuth']
orientations1['polarity'] = orientations1['polarity'].astype(float)
orientations1['X'] = orientations1['X'].astype(float)
orientations1['Y'] = orientations1['Y'].astype(float)
orientations1
```

[15]:
```
        Z formation  azimuth   dip  polarity      X      Y  \
0 -133.33      Base   312.61 12.44      1.00 785.92 444.93

                  geometry
0  POINT (785.918 444.927)
```

[16]:
```
orientations2 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪base.loc[1:3])
orientations2['azimuth'] = 360-orientations2['azimuth']
orientations2
```

[16]:
```
        Z formation azimuth   dip polarity      X      Y  \
0 -156.67      Base  312.69 12.42        1 815.34 568.58

                  geometry
0  POINT (815.339 568.578)
```

[17]:
```
orientations3 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪base.loc[2:4])
orientations3['Z'] = orientations3['Z'].astype(float)
orientations3['azimuth'] = orientations3['azimuth'].astype(float)
orientations3['dip'] = orientations3['dip'].astype(float)
orientations3['dip'] = 180 - orientations3['dip']
orientations3['azimuth'] = 180 - orientations3['azimuth']
orientations3['polarity'] = orientations3['polarity'].astype(float)
orientations3['X'] = orientations3['X'].astype(float)
orientations3['Y'] = orientations3['Y'].astype(float)
orientations3
```

[17]:
```
        Z formation  azimuth   dip  polarity      X      Y  \
0 -233.33      Base   312.84 12.52      1.00 925.69 959.07

                  geometry
0  POINT (925.692 959.069)
```

[18]:
```
orientations4 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪base.loc[3:5])
orientations4['Z'] = orientations4['Z'].astype(float)
orientations4['azimuth'] = orientations4['azimuth'].astype(float)
orientations4['dip'] = orientations4['dip'].astype(float)
orientations4['dip'] = 180 - orientations4['dip']
```

(continues on next page)

(continued from previous page)

```
orientations4['azimuth'] = 180 - orientations4['azimuth']
orientations4['polarity'] = orientations4['polarity'].astype(float)
orientations4['X'] = orientations4['X'].astype(float)
orientations4['Y'] = orientations4['Y'].astype(float)
orientations4
```

[18]:
```
        Z formation  azimuth   dip  polarity      X       Y  \
0 -286.67      Base   310.80 12.35      1.00 979.34 1258.56

                    geometry
0  POINT (979.342 1258.564)
```

[19]:
```
interfaces_top = interfaces[interfaces['formation'] == 'Top'].reset_index()
interfaces_top
```

[19]:
```
   index    id formation    Z                    geometry
0      2  None       Top  -50   POINT (1052.804 601.233)
1      4  None       Top -100    POINT (399.981 974.235)
2      6  None       Top -300  POINT (1324.290 1301.740)
3      8  None       Top -350  POINT (1213.755 1499.718)
```

[20]:
```
orientations5 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪top.loc[0:2])
orientations5['azimuth'] = 360 - orientations5['azimuth']
orientations5
```

[20]:
```
       Z formation azimuth   dip polarity      X      Y  \
0 -150.0       Top  341.77 18.43        1 925.69 959.07

                   geometry
0  POINT (925.692 959.069)
```

[21]:
```
orientations6 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪top.loc[1:3])
orientations6['azimuth'] = 180 - orientations6['azimuth']
orientations6['dip'] = 180 - orientations6['dip']
orientations6
```

[21]:
```
       Z formation azimuth   dip polarity      X       Y  \
0 -250.0       Top  341.23 18.22        1 979.34 1258.56

                    geometry
0  POINT (979.342 1258.564)
```

[22]:
```
orientations7 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪top.loc[[0, 2, 3]])
orientations7['azimuth'] = 180 - orientations7['azimuth']
orientations7['dip'] = 180 - orientations7['dip']
orientations7
```

[22]:
```
        Z formation azimuth   dip polarity       X       Y  \
0 -233.33       Top  340.65 18.41        1 1196.95 1134.23
```

(continues on next page)

(continued from previous page)

```
                    geometry
0  POINT (1196.950 1134.230)
```

[23]:
```
orientations8 = gg.vector.calculate_orientation_for_three_point_problem(gdf=interfaces_
↪top.loc[[0, 1, 3]])
orientations8['azimuth'] = 360 - orientations8['azimuth']
orientations8
```

[23]:
```
        Z formation azimuth   dip polarity      X       Y  \
0 -166.67       Top  341.82 18.36        1 888.85 1025.06

                   geometry
0  POINT (888.847 1025.062)
```

Merging Orientations

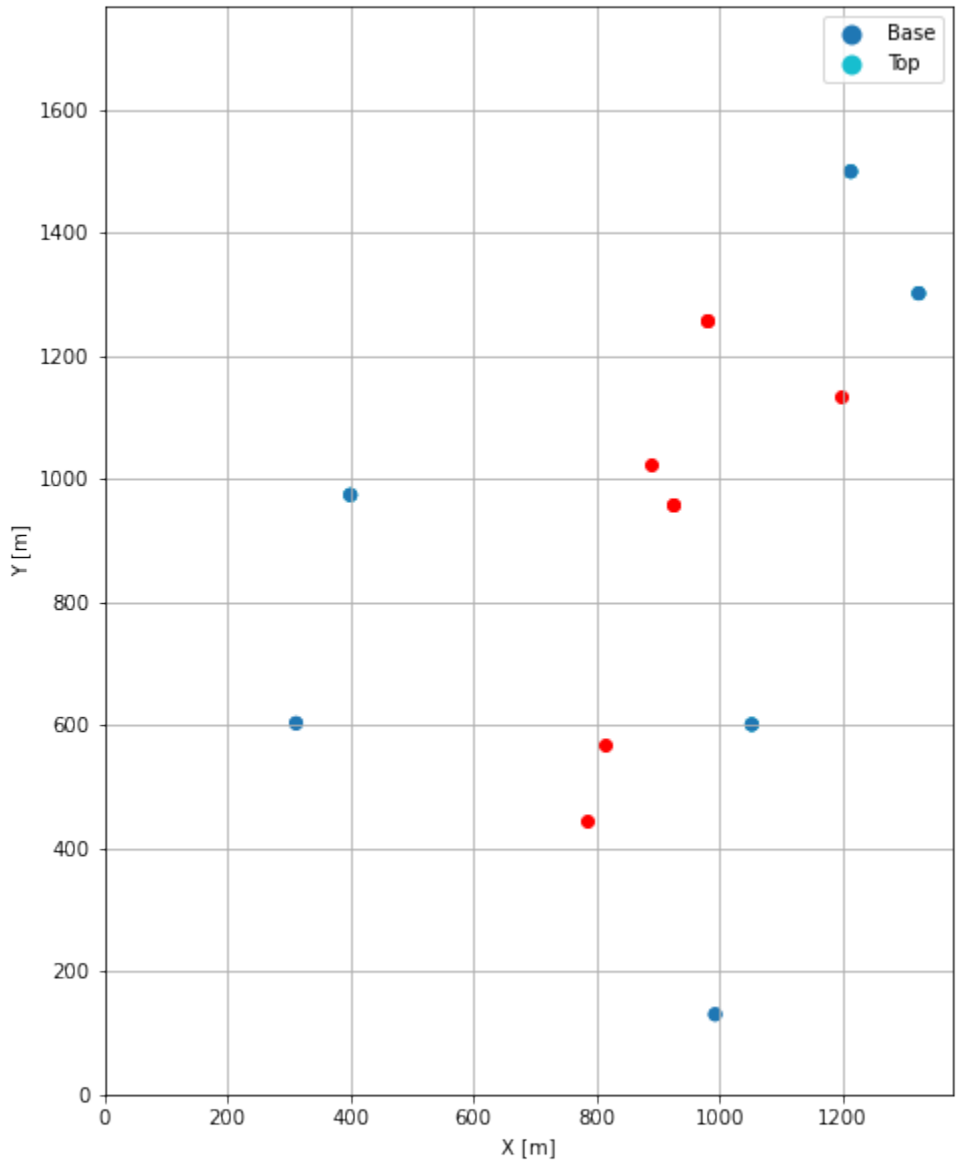
[24]:
```
import pandas as pd
orientations = pd.concat([orientations1, orientations2, orientations3, orientations4,␣
↪orientations5, orientations6, orientations7, orientations8]).reset_index()
orientations = orientations[orientations['formation'].isin(['Base', 'Top'])]
orientations['Z'] = orientations['Z'].astype(float)
orientations['azimuth'] = orientations['azimuth'].astype(float)
orientations['dip'] = orientations['dip'].astype(float)
orientations['polarity'] = orientations['polarity'].astype(float)
orientations['X'] = orientations['X'].astype(float)
orientations['Y'] = orientations['Y'].astype(float)
orientations
```

[24]:
```
   index       Z formation  azimuth   dip  polarity       X       Y  \
0      0 -133.33      Base   312.61 12.44      1.00  785.92  444.93
1      0 -156.67      Base   312.69 12.42      1.00  815.34  568.58
2      0 -233.33      Base   312.84 12.52      1.00  925.69  959.07
3      0 -286.67      Base   310.80 12.35      1.00  979.34 1258.56
4      0 -150.00       Top   341.77 18.43      1.00  925.69  959.07
5      0 -250.00       Top   341.23 18.22      1.00  979.34 1258.56
6      0 -233.33       Top   340.65 18.41      1.00 1196.95 1134.23
7      0 -166.67       Top   341.82 18.36      1.00  888.85 1025.06

                    geometry
0    POINT (785.918 444.927)
1    POINT (815.339 568.578)
2    POINT (925.692 959.069)
3   POINT (979.342 1258.564)
4    POINT (925.692 959.069)
5   POINT (979.342 1258.564)
6  POINT (1196.950 1134.230)
7   POINT (888.847 1025.062)
```

Plotting the Orientations

[25]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 1381)
ax.set_ylim(0, 1768)
```

[25]:
```
(0.0, 1768.0)
```



7.20.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[26]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[27]:
```
geo_model = gp.create_model('Model20')
geo_model
```

[27]:
```
Model20  2022-04-05 11:17
```

Initiate Data

[28]:
```
gp.init_data(geo_model, [0, 1381, 0, 1768, -500, 250], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[28]:
```
Model20  2022-04-05 11:17
```

Model Surfaces

[29]:
```
geo_model.surfaces
```

[29]:

	surface	series	order_surfaces	color	id
0	Top	Default series	1	#015482	1
1	Base	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[30]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Top'),
                          'Strata2': ('Base'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[30]:
```
    surface   series  order_surfaces    color  id
0       Top  Strata1               1  #015482   1
1      Base  Strata2               1  #9f0052   2
2  Basement  Strata2               2  #ffbe00   3
```

Showing the Number of Data Points

[31]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[31]:
```
    surface   series  order_surfaces    color  id  No. of Interfaces  No. of Orientations
0       Top  Strata1               1  #015482   1                  4                    4
1      Base  Strata2               1  #9f0052   2                  6                    4
2  Basement  Strata2               2  #ffbe00   3                  0                    0
```

Loading Digital Elevation Model

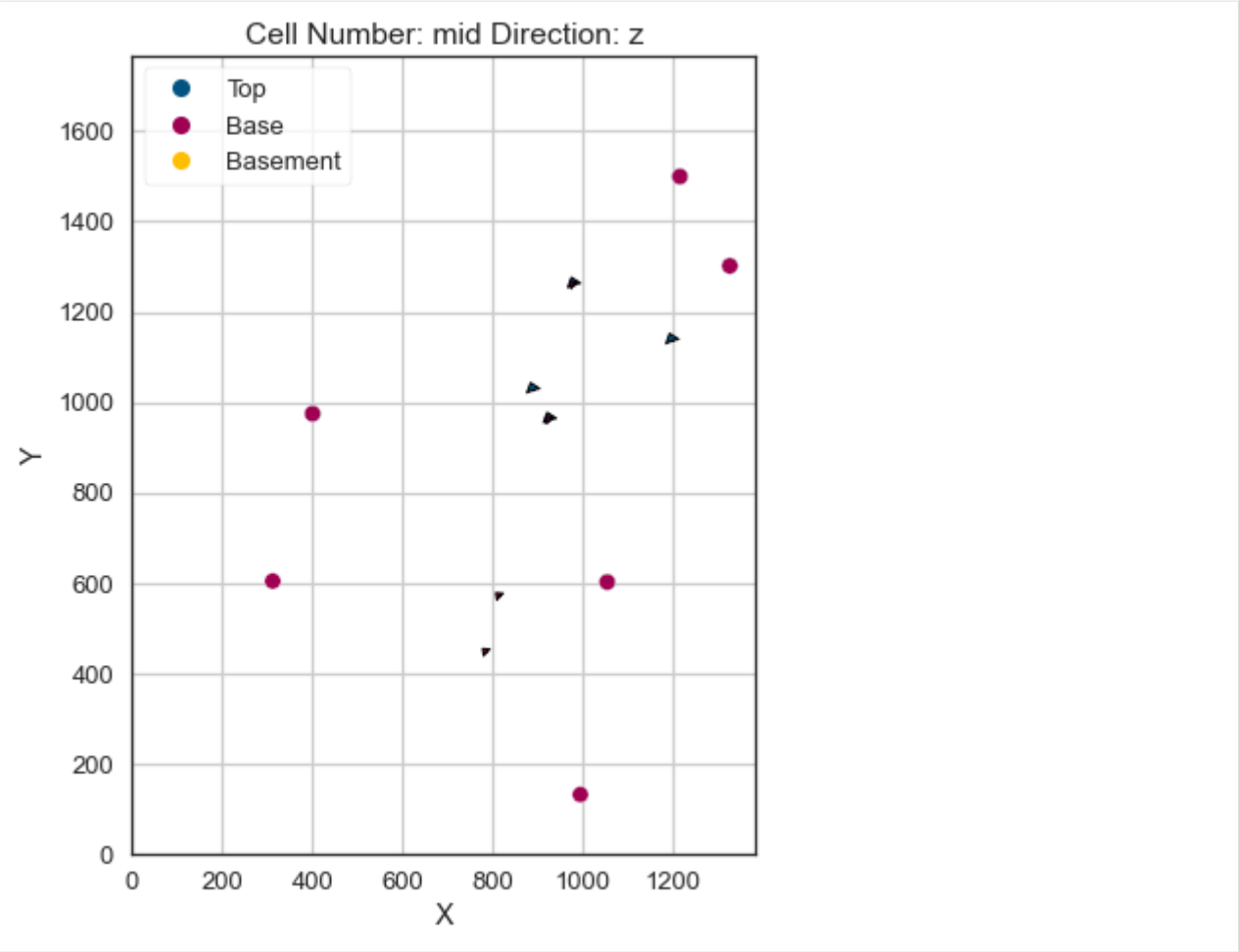
[32]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster20.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

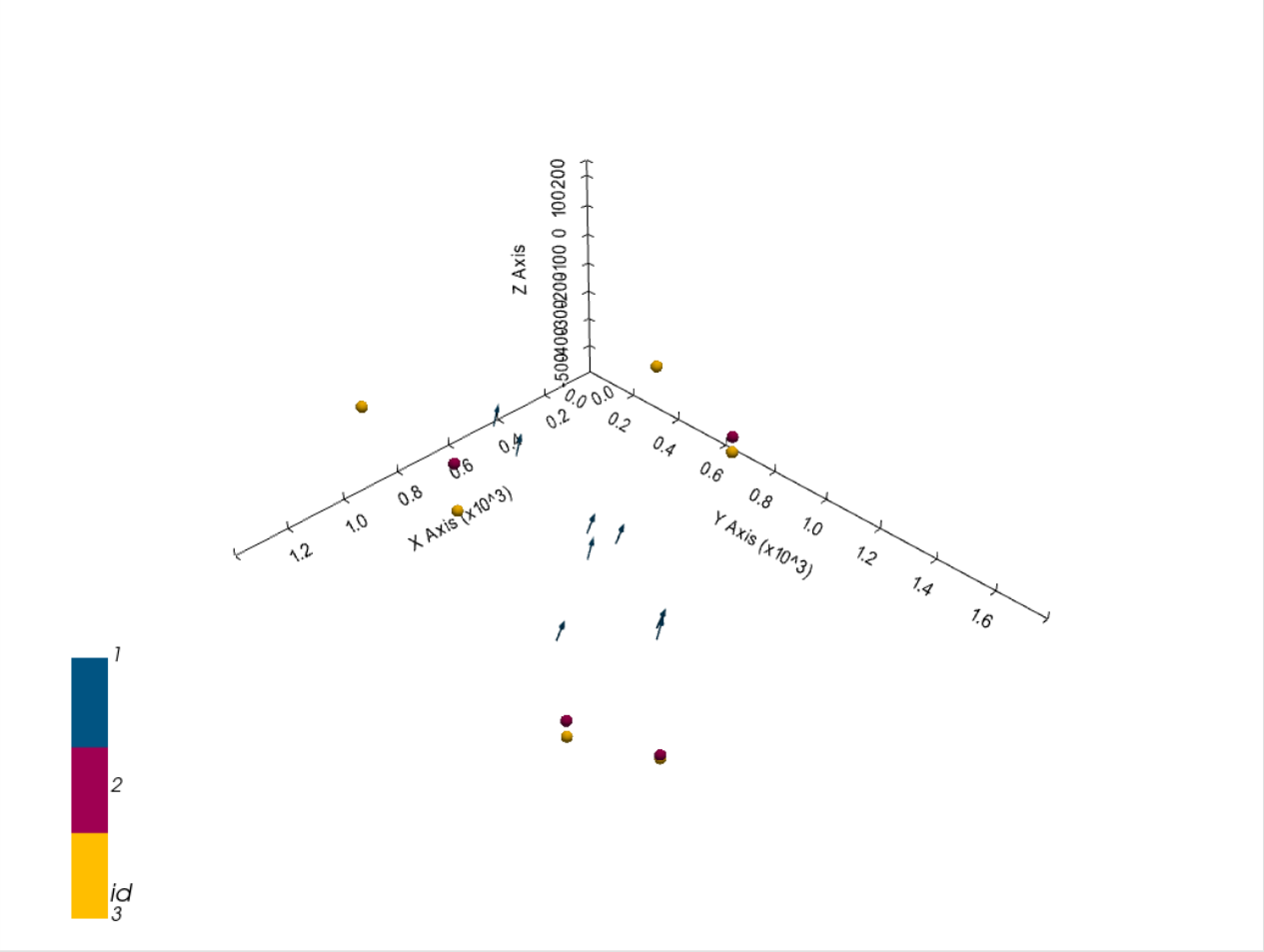
[32]:
```
Grid Object. Values:
array([[   6.905     ,    8.84      , -496.25      ],
       [   6.905     ,    8.84      , -488.75      ],
       [   6.905     ,    8.84      , -481.25      ],
       ...,
       [1377.07670455, 1735.97101449,    0.        ],
       [1377.07670455, 1748.7826087 ,    0.        ],
       [1377.07670455, 1761.5942029 ,    0.        ]])
```

Plotting Input Data

[33]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[34]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[34]:
```
<gempy.plot.vista.GemPyToVista at 0x25117b97730>
```

Setting the Interpolator

[35]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              2365.48
$C_o$            133225.83
drift equations     [3, 3]
```

[35]:
```
<gempy.core.interpolator.InterpolatorModel at 0x2511175ab20>
```

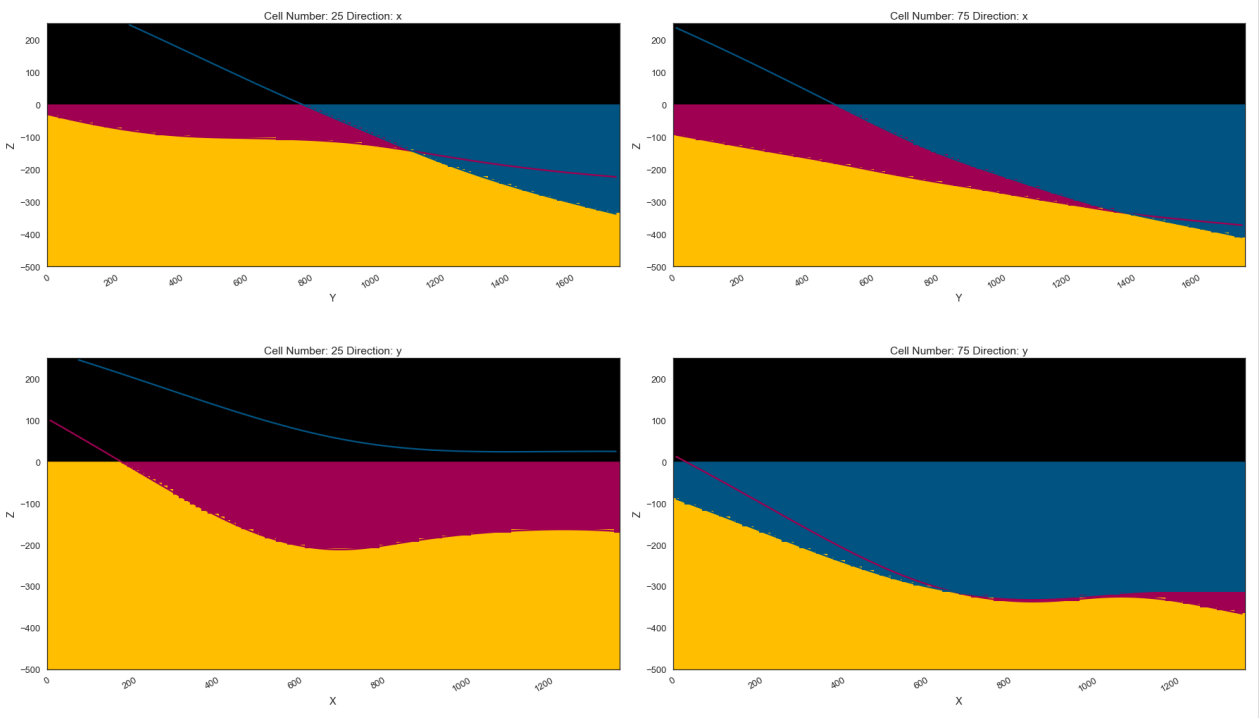
Computing Model

[36]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

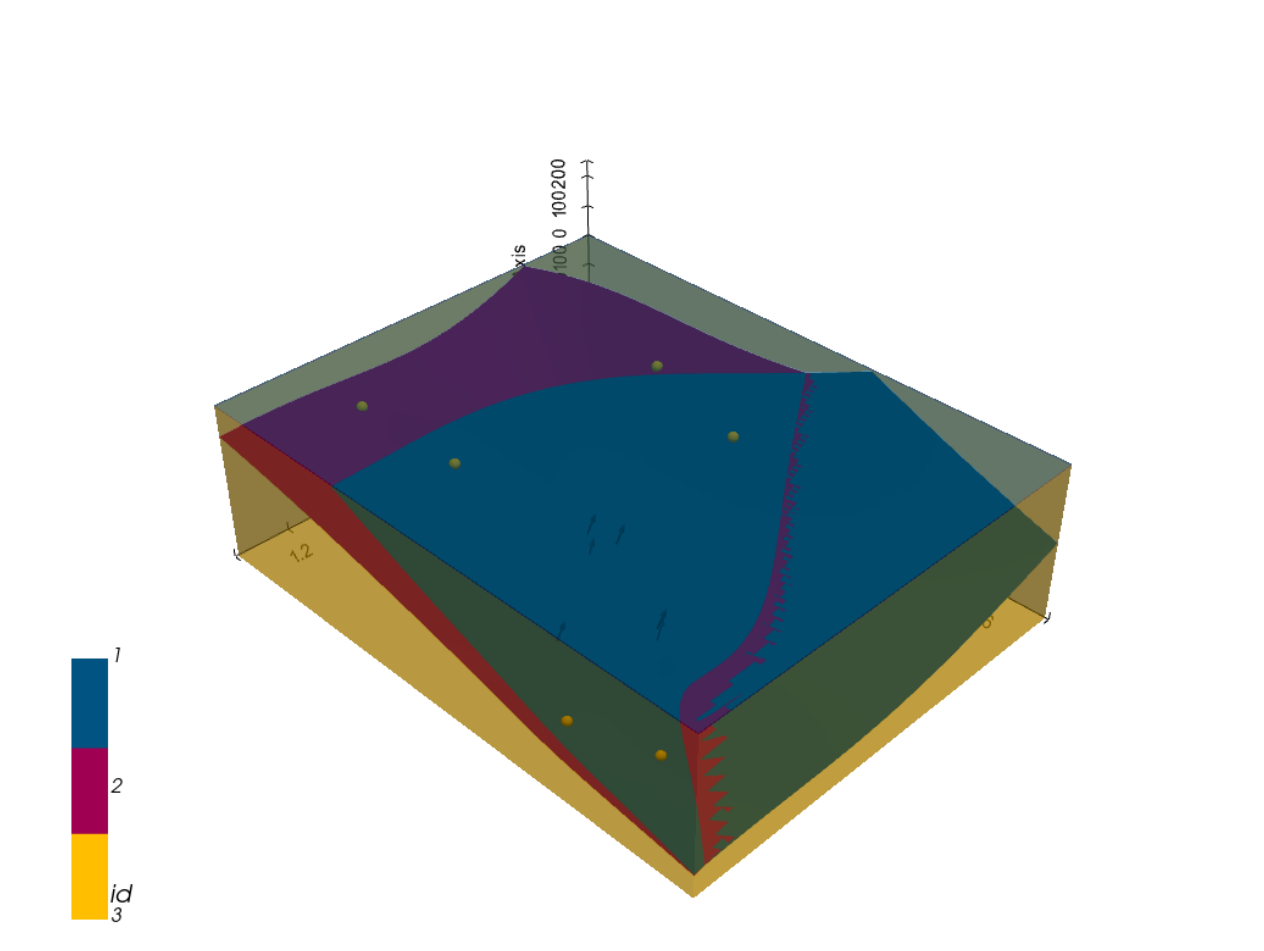
[37]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[37]:
```
<gempy.plot.visualization_2d.Plot2D at 0x25119b53f10>
```



Plotting 3D Model

[38]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=False,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.21 Example 21 - Coal Seam Mining

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3727 m wide (W-E extent) and 2596 m high (N-S extent). The vertical model extent varies between 2400 m and 2475 m. The model represents a coal seam that is outcropping at the surface and was encountered in boreholes at depth.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

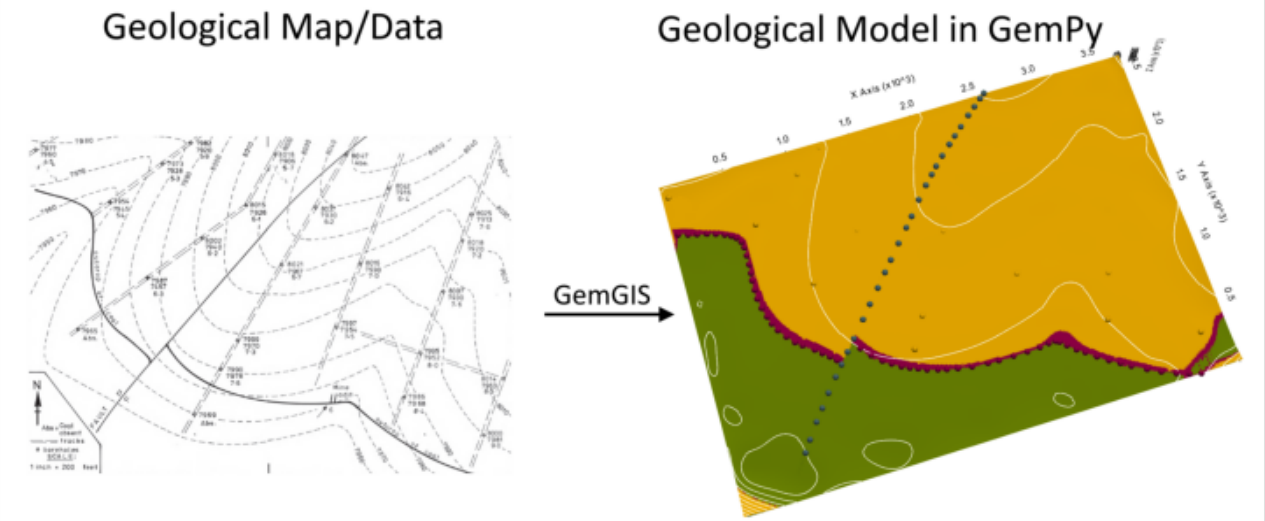
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example21.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.21.1 Licensing

7.21.2 Import GemGIS

If you have installed `GemGIS` via pip and conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.21.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

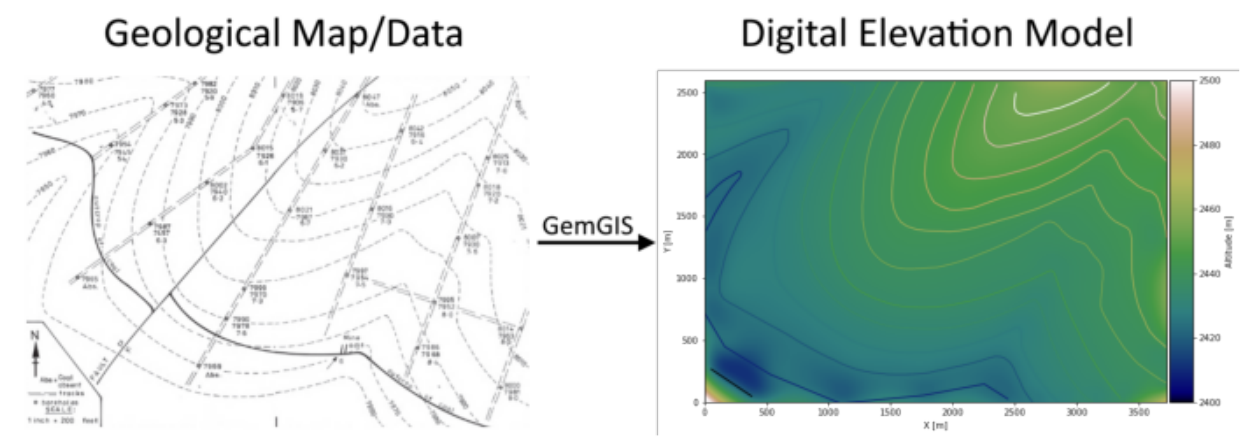
[4]:
```
file_path = 'data/example21/'
gg.download_gemgis_data.download_tutorial_data(filename="example21_coal_seam_mining.zip",
↪ dirpath=file_path)
```

```
Downloading file 'example21_coal_seam_mining.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample21_coal_seam_mining.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\example\data\example21'.
```

7.21.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example21.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo21.shp')
topo['Z'] = topo['Z']*0.3048
topo.head()
```

[6]:

	id	Z	geometry
0	None	2423.16	LINESTRING (8.426 1648.306, 35.717 1675.998, 6...
1	None	2426.21	LINESTRING (5.215 1950.921, 79.464 1983.831, 1...
2	None	2429.26	LINESTRING (16.854 2215.007, 64.213 2217.014, ...
3	None	2432.30	LINESTRING (24.179 2563.075, 70.835 2554.346, ...
4	None	2432.30	LINESTRING (957.912 2594.380, 1031.960 2594.98...

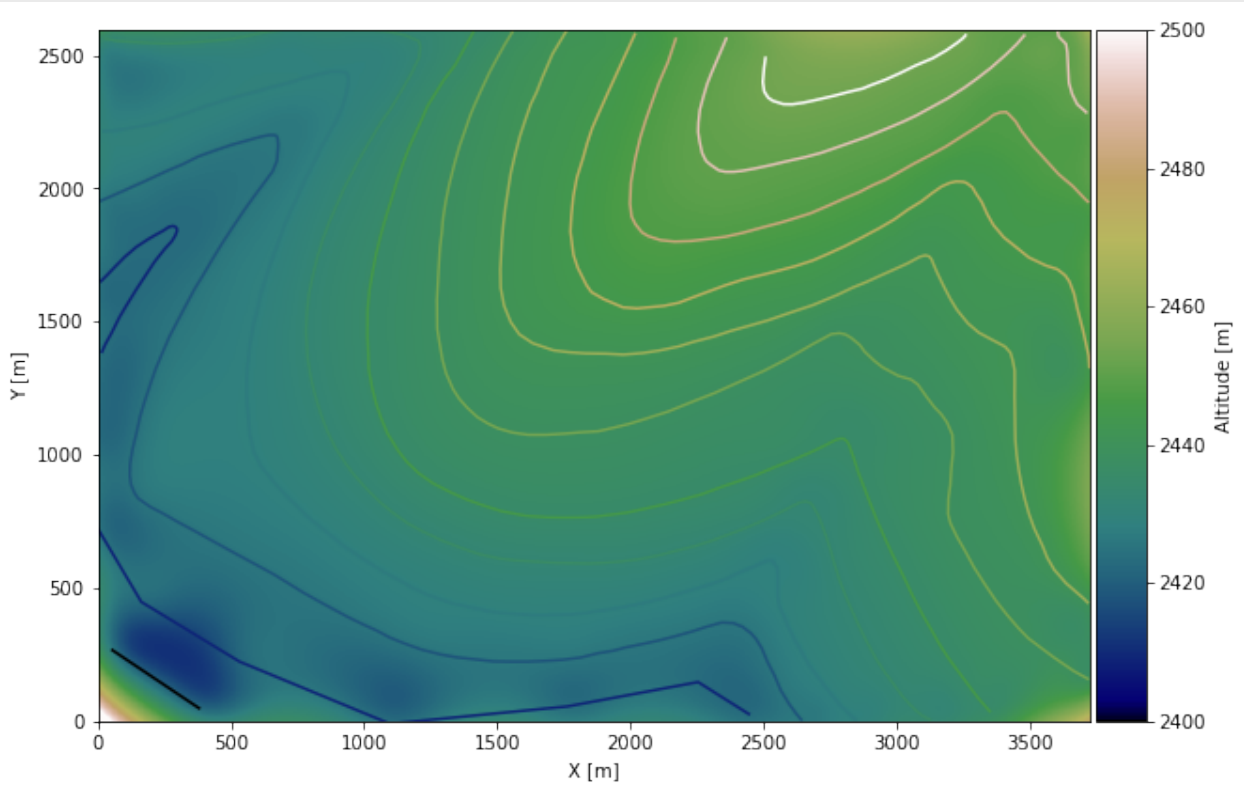
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0, 3727, 0, 2596], cmap='gist_earth',
↪ clim=[2400, 2500])
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3727)
ax.set_ylim(0, 2596)
```

[8]:
```
(0.0, 2596.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster21.tif', extent = [0, 3727, 0, 2596], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

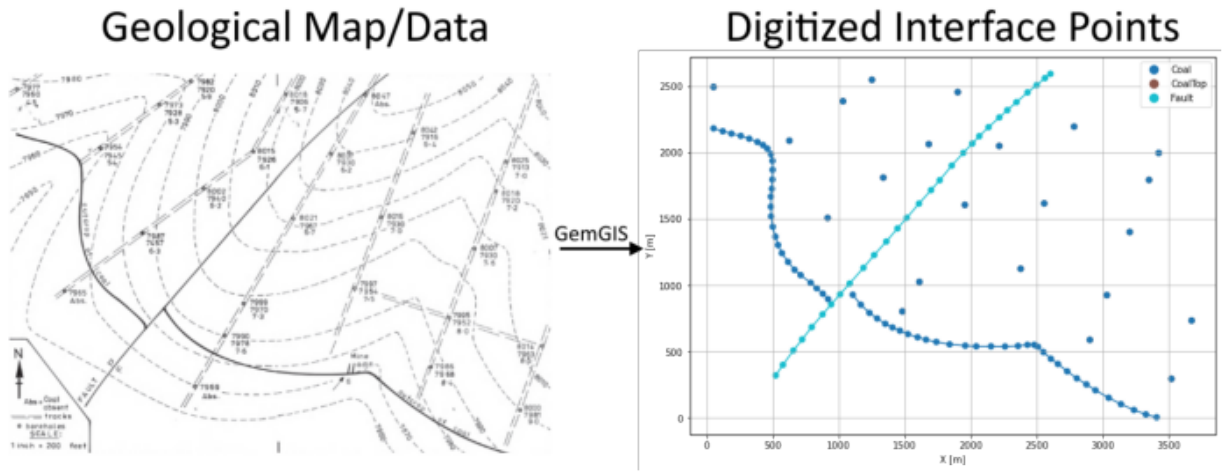
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster21.tif')
```

7.21.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example21.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces21.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Coal	LINESTRING (53.076 2181.695, 122.910 2160.023,...
1	None	Coal	LINESTRING (1107.213 927.687, 1168.017 852.435...
2	None	Fault	LINESTRING (523.855 319.647, 578.639 399.114, ...

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

[12]:
```
   formation                   geometry       X       Y       Z
87     Fault  POINT (2604.437 2592.875) 2604.44 2592.88 2458.99
73     Fault  POINT (1608.696 1614.592) 1608.70 1614.59 2442.26
59     Fault    POINT (523.855 319.647)  523.86  319.65 2422.09
60     Fault    POINT (578.639 399.114)  578.64  399.11 2424.08
61     Fault    POINT (657.504 507.477)  657.50  507.48 2425.55
```

[13]:
```
boreholes = gpd.read_file(file_path + 'points21.shp')
boreholes = gg.vector.extract_xy(gdf=boreholes)
boreholes['Z'] = boreholes['Z']*0.3048
boreholes.head()
```

[13]:
```
  formation       Z                   geometry       X       Y
0   CoalTop 2421.64   POINT (627.403 2088.984)  627.40 2088.98
1   CoalTop 2426.21    POINT (53.076 2493.542)   53.08 2493.54
2   CoalTop 2416.45  POINT (1033.164 2386.382) 1033.16 2386.38
3   CoalTop 2414.02  POINT (1252.300 2547.724) 1252.30 2547.72
4   CoalTop 2425.29   POINT (916.975 1506.229)  916.97 1506.23
```

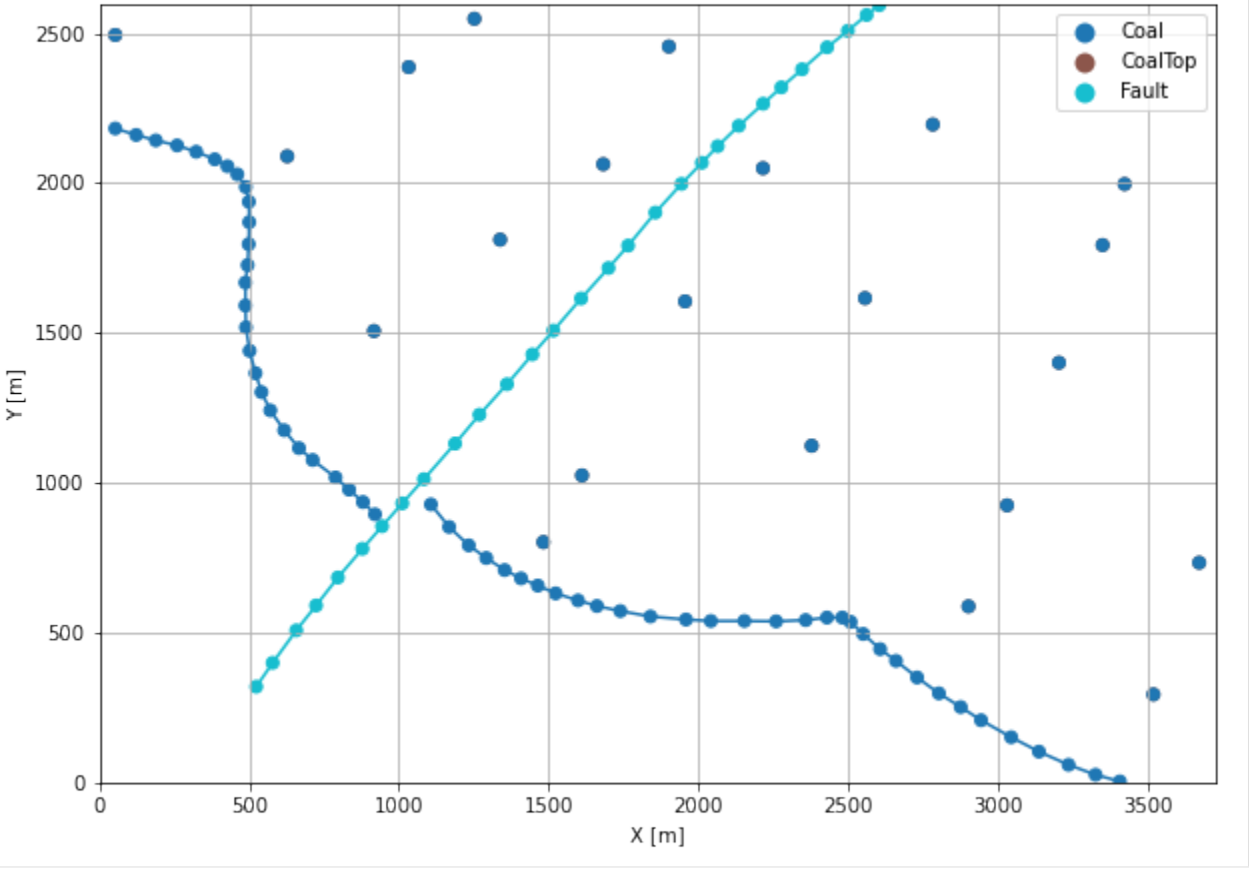
[14]:
```
import pandas as pd
interfaces_coords = pd.concat([interfaces_coords, boreholes])
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Fault',
↪'CoalTop', 'Coal'])].reset_index()
interfaces_coords
```

[14]:
```
     index formation                   geometry       X       Y       Z
0       87     Fault  POINT (2604.437 2592.875) 2604.44 2592.88 2458.99
1       73     Fault  POINT (1608.696 1614.592) 1608.70 1614.59 2442.26
2       59     Fault    POINT (523.855 319.647)  523.86  319.65 2422.09
3       60     Fault    POINT (578.639 399.114)  578.64  399.11 2424.08
4       61     Fault    POINT (657.504 507.477)  657.50  507.48 2425.55
..     ...       ...                       ...     ...     ...     ...
127     39      Coal  POINT (3204.651 1400.273) 3204.65 1400.27 2416.45
128     40      Coal  POINT (3350.340 1792.790) 3350.34 1792.79 2411.88
129     41      Coal  POINT (3423.787 1996.273) 3423.79 1996.27 2409.75
130     42      Coal   POINT (3520.110 293.760) 3520.11  293.76 2429.87
131     43      Coal   POINT (3673.023 733.235) 3673.02  733.23 2424.68

[132 rows x 6 columns]
```

Plotting the Interface Points

[15]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3727)
ax.set_ylim(0, 2596)
```

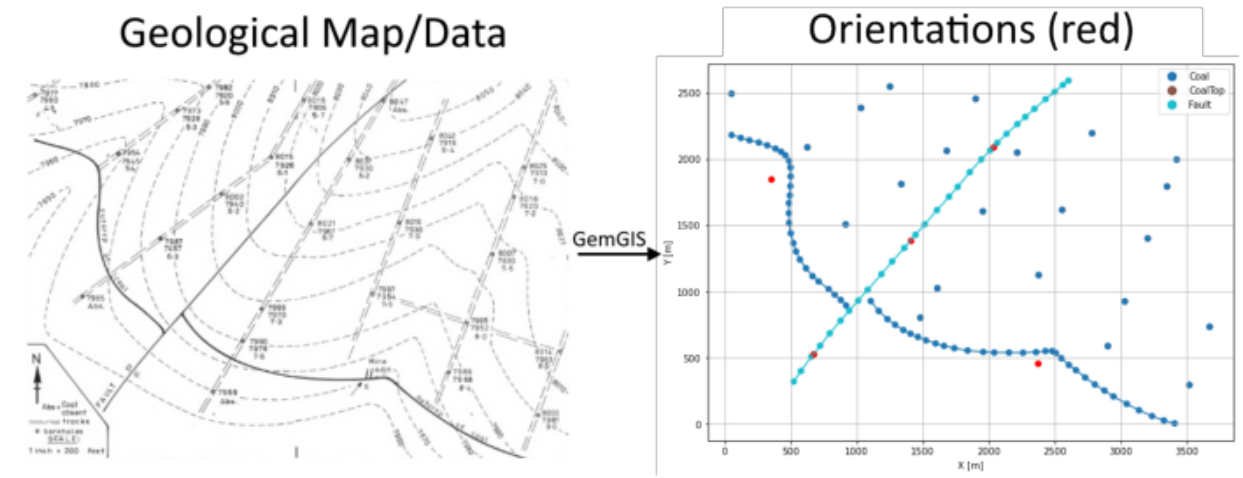
[15]:
```
(0.0, 2596.0)
```



7.21.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[16]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example21.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[17]:
```
strikes = gpd.read_file(file_path + 'strikes21.shp')
strikes['Z'] = strikes['Z']*0.3048
strikes.head()
```

[17]:

	id	formation	Z	geometry
0	1	Coal1	2426.21	LINESTRING (335.423 2099.821, 499.173 1840.952)
1	2	Coal1	2429.26	LINESTRING (564.191 1250.972, 3.108 2216.613)
2	2	Coal2	2432.30	LINESTRING (1619.532 600.790, 2945.181 207.069)
3	1	Coal2	2429.26	LINESTRING (2364.833 540.588, 2570.724 477.978)

[18]:
```
orientations_fault = gpd.read_file(file_path + 'orientations21_fault.shp')
orientations_fault = gg.vector.extract_xyz(gdf=orientations_fault, dem=topo_raster)
orientations_fault
```

[18]:

	formation	dip	azimuth	polarity	geometry	X	Y	Z
0	Fault	60.00	305.00	1.00	POINT (670.146 526.140)	670.15	526.14	2425.71
1	Fault	60.00	305.00	1.00	POINT (1410.631 1387.029)	1410.63	1387.03	2439.39
2	Fault	60.00	305.00	1.00	POINT (2034.324 2095.005)	2034.32	2095.00	2447.71

Calculate Orientations for each formation

[19]:
```
orientations_coal1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal1'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal1
```

[19]:
```
   dip  azimuth       Z                  geometry  polarity formation      X  \
0 0.76    59.69 2427.73  POINT (350.474 1852.089)      1.00     Coal1 350.47

        Y
0 1852.09
```

[20]:
```
orientations_coal2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Coal2'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal2
```

[20]:
```
   dip  azimuth       Z                  geometry  polarity formation       X  \
0 1.14    16.55 2430.78  POINT (2375.067 456.607)      1.00     Coal2 2375.07

       Y
0 456.61
```

Merging Orientations

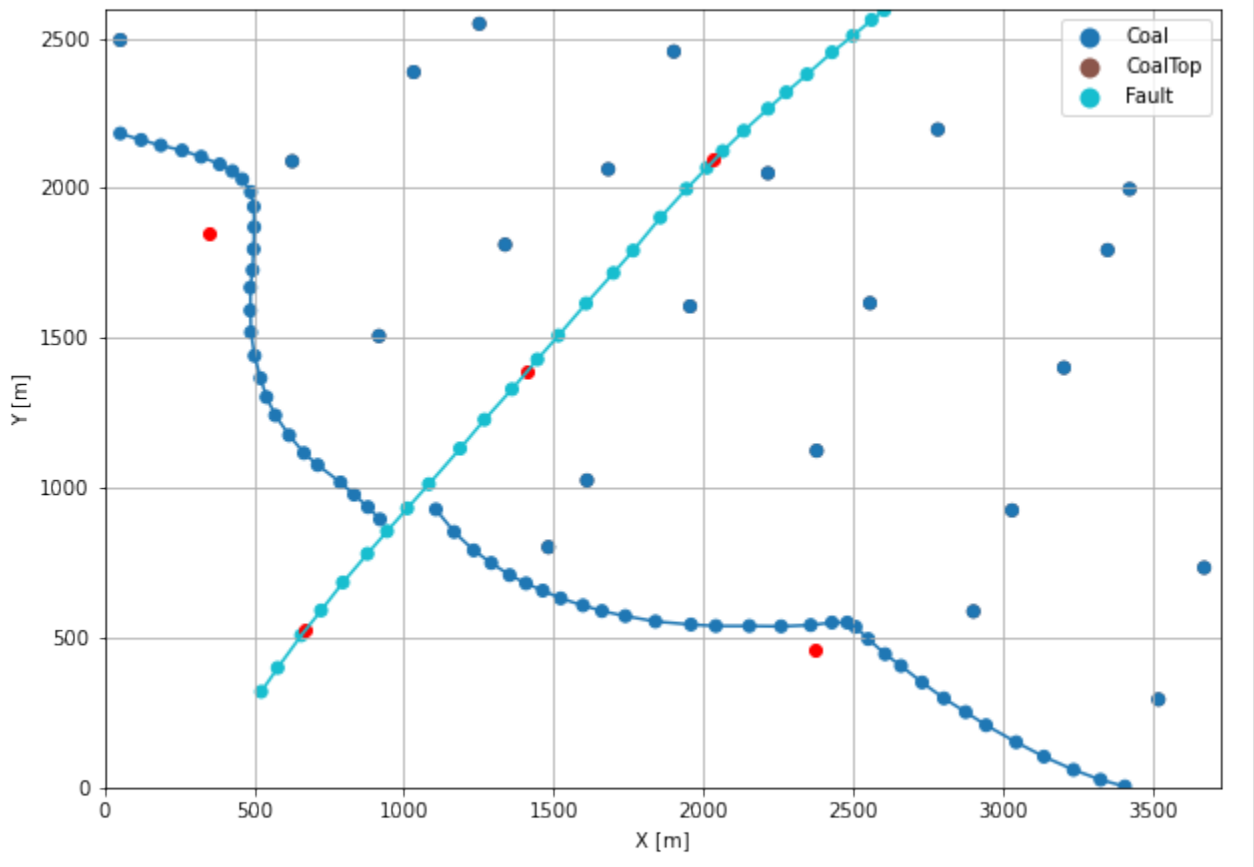
[21]:
```
import pandas as pd
orientations = pd.concat([orientations_coal1, orientations_coal2, orientations_fault]).
↪reset_index()
orientations['formation'] = ['Coal', 'Coal', 'Fault', 'Fault', 'Fault']
orientations = orientations[orientations['formation'].isin(['Coal', 'Fault'])]
orientations
```

[21]:
```
   index   dip  azimuth       Z                   geometry  polarity  \
0      0  0.76    59.69 2427.73   POINT (350.474 1852.089)      1.00
1      0  1.14    16.55 2430.78   POINT (2375.067 456.607)      1.00
2      0 60.00   305.00 2425.71    POINT (670.146 526.140)      1.00
3      1 60.00   305.00 2439.39  POINT (1410.631 1387.029)      1.00
4      2 60.00   305.00 2447.71  POINT (2034.324 2095.005)      1.00

  formation       X       Y
0      Coal  350.47 1852.09
1      Coal 2375.07  456.61
2     Fault  670.15  526.14
3     Fault 1410.63 1387.03
4     Fault 2034.32 2095.00
```

Plotting the Orientations

[22]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 3727)
ax.set_ylim(0, 2596)
```

[22]:
```
(0.0, 2596.0)
```



7.21.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[23]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
```

(continues on next page)

(continued from previous page)

```
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[24]:
```
geo_model = gp.create_model('Model21')
geo_model
```

[24]:
```
Model21  2022-04-05 11:19
```

Initiate Data

[25]:
```
gp.init_data(geo_model, [0, 3727, 0, 2596, 2400, 2475], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[25]:
```
Model21  2022-04-05 11:19
```

Model Surfaces

[26]:
```
geo_model.surfaces
```

[26]:

	surface	series	order_surfaces	color	id
0	Fault	Default series	1	#015482	1
1	Coal	Default series	2	#9f0052	2
2	CoalTop	Default series	3	#ffbe00	3

Mapping the Stack to Surfaces

[27]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Fault1': ('Fault'),
                          'Strata1': ('CoalTop', 'Coal'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
geo_model.set_is_fault(['Fault1'])
```

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

[27]:

	order_series	BottomRelation	isActive	isFault	isFinite
Fault1	1	Fault	True	True	False
Strata1	2	Erosion	True	False	False

Showing the Number of Data Points

[28]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[28]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Fault	Fault1	1	#527682	1	29	3
1	Coal	Strata1	1	#9f0052	2	81	2
2	CoalTop	Strata1	2	#ffbe00	3	22	0
3	Basement	Strata1	3	#728f02	4	0	0

Loading Digital Elevation Model

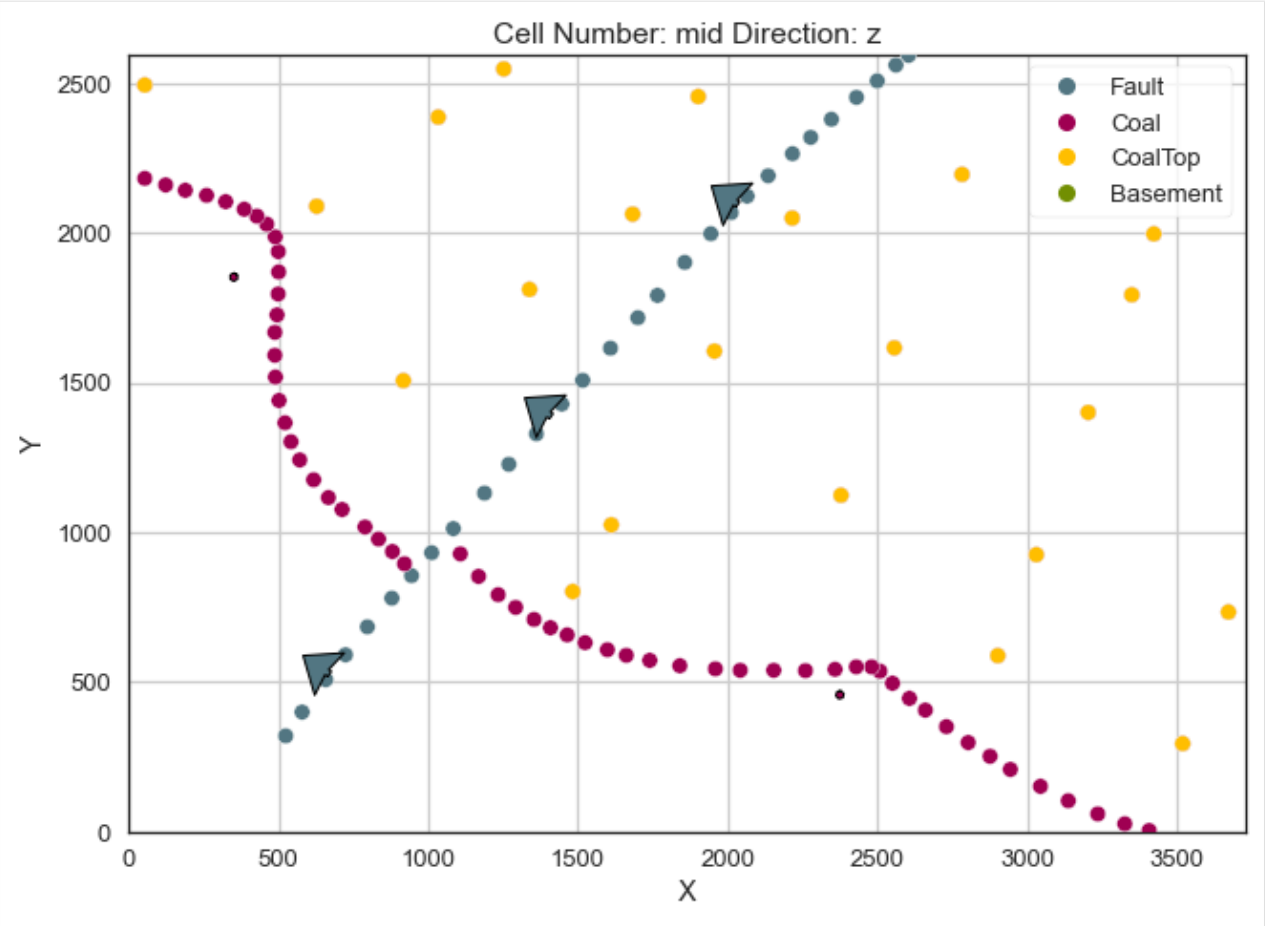
[29]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster21.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

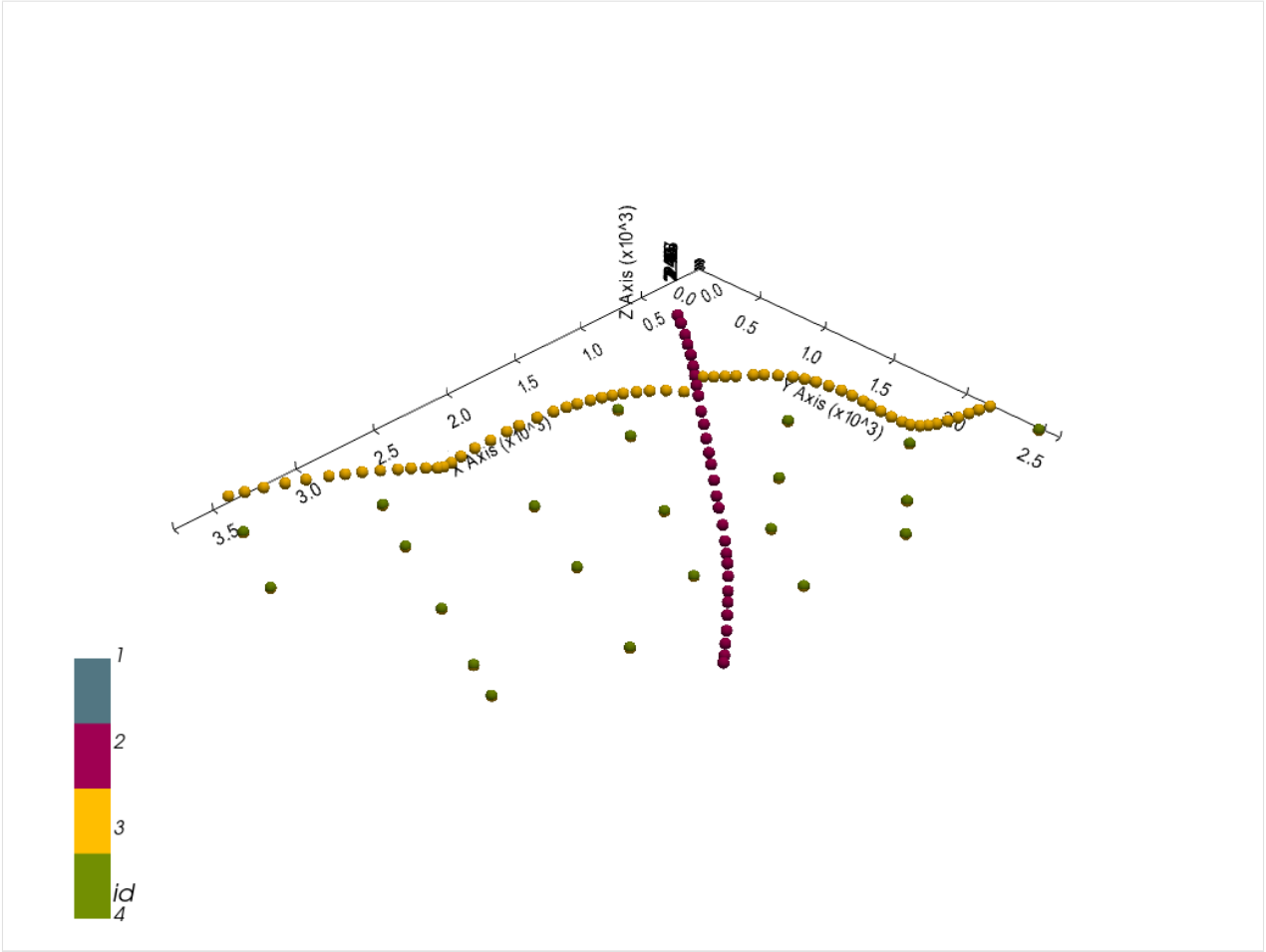
[29]:
```
Grid Object. Values:
array([[  18.635     ,   12.98      , 2400.375     ],
       [  18.635     ,   12.98      , 2401.125     ],
       [  18.635     ,   12.98      , 2401.875     ],
       ...,
       [3722.04388298, 2571.13409962, 2462.30371094],
       [3722.04388298, 2581.08045977, 2463.51391602],
       [3722.04388298, 2591.02681992, 2464.8894043 ]])
```

Plotting Input Data

[30]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[31]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[31]:
```
<gempy.plot.vista.GemPyToVista at 0x1799e93afa0>
```

Setting the Interpolator

[32]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                    values
range              4542.62
$C_o$            491318.33
drift equations     [3, 3]
```

[32]:
```
<gempy.core.interpolator.InterpolatorModel at 0x17997fa6520>
```

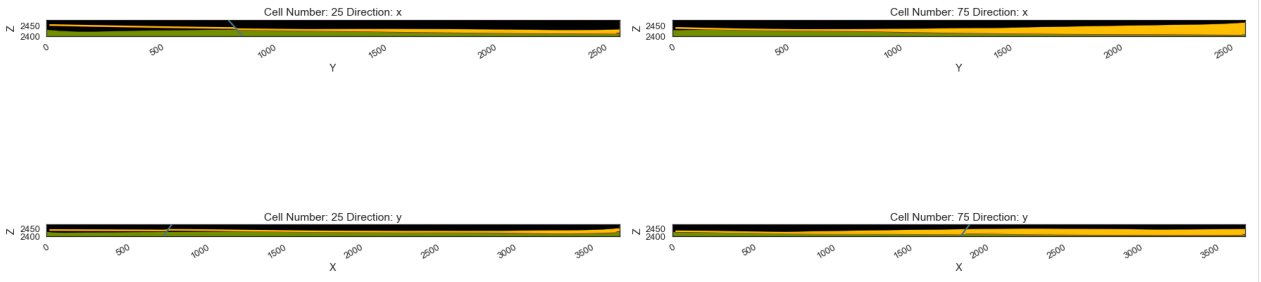
Computing Model

[33]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

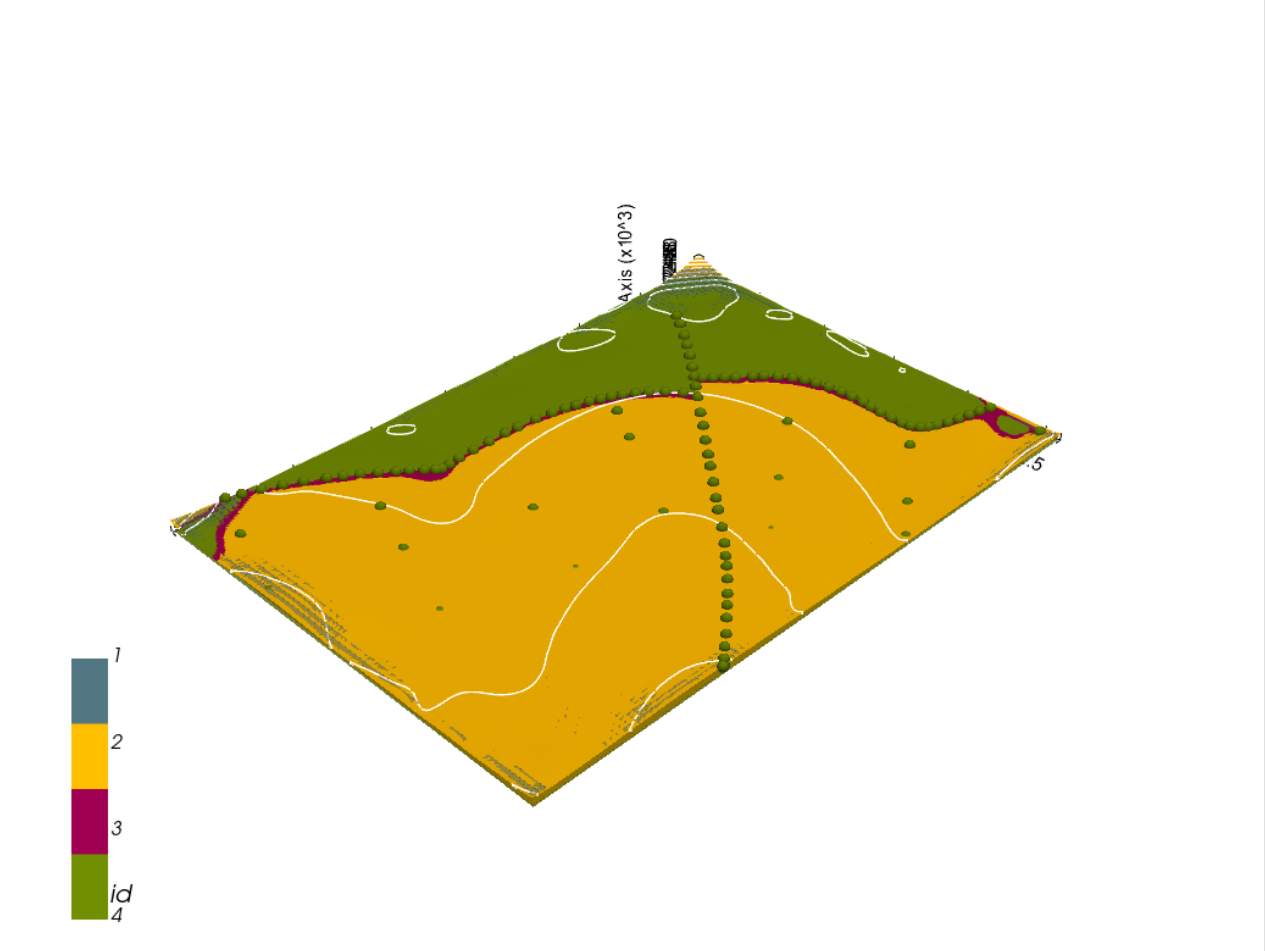
[34]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[34]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1799fb532b0>
```



Plotting 3D Model

[35]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

7.22 Example 22 - Coal Measures

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 6875 m wide (W-E extent) and 9954 m high (N-S extent). The vertical model extent varies 500 m and 1250 m. The model represents coal measures which were mapped on the surface and at depth using boreholes.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

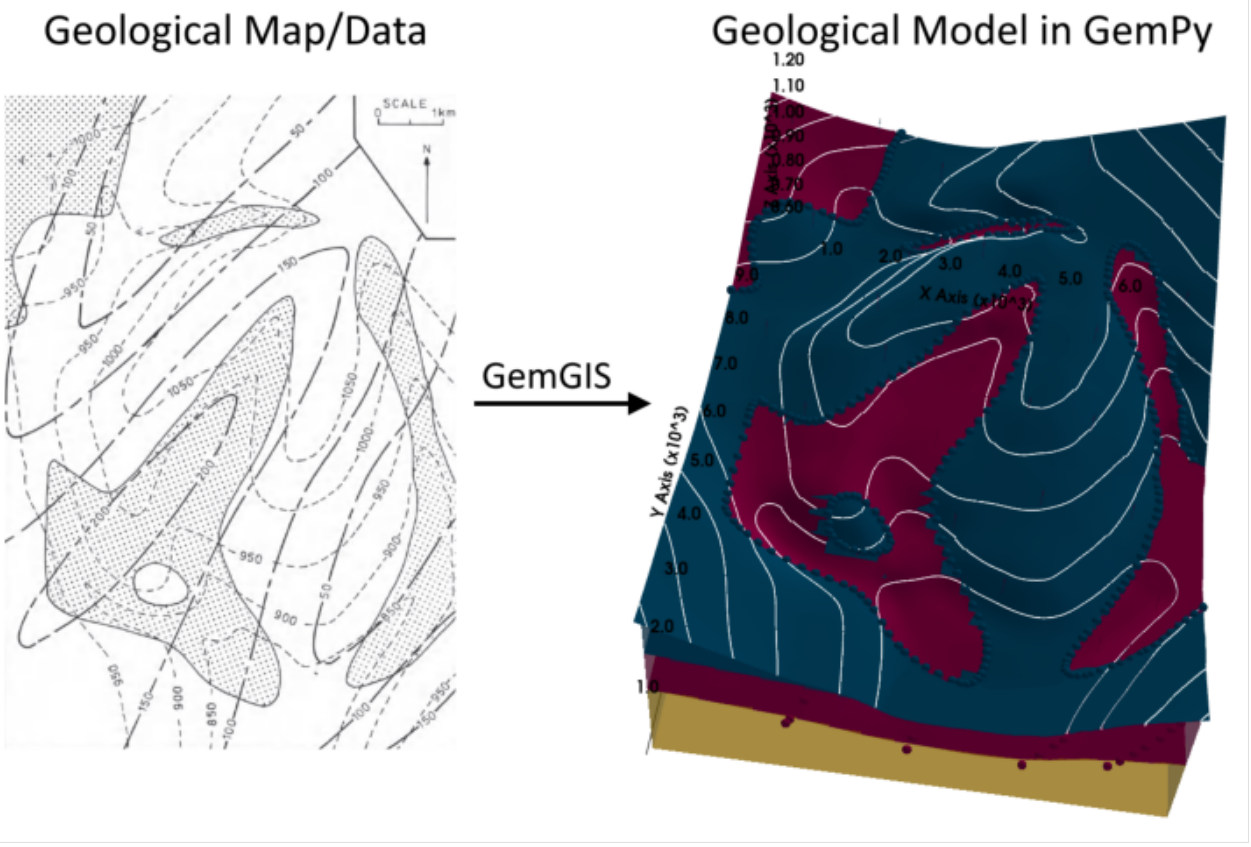
Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example22.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
```

(continues on next page)

(continued from previous page)

```
plt.axis('off')
plt.tight_layout()
```



7.22.1 Licensing

7.22.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.22.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using pooch. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

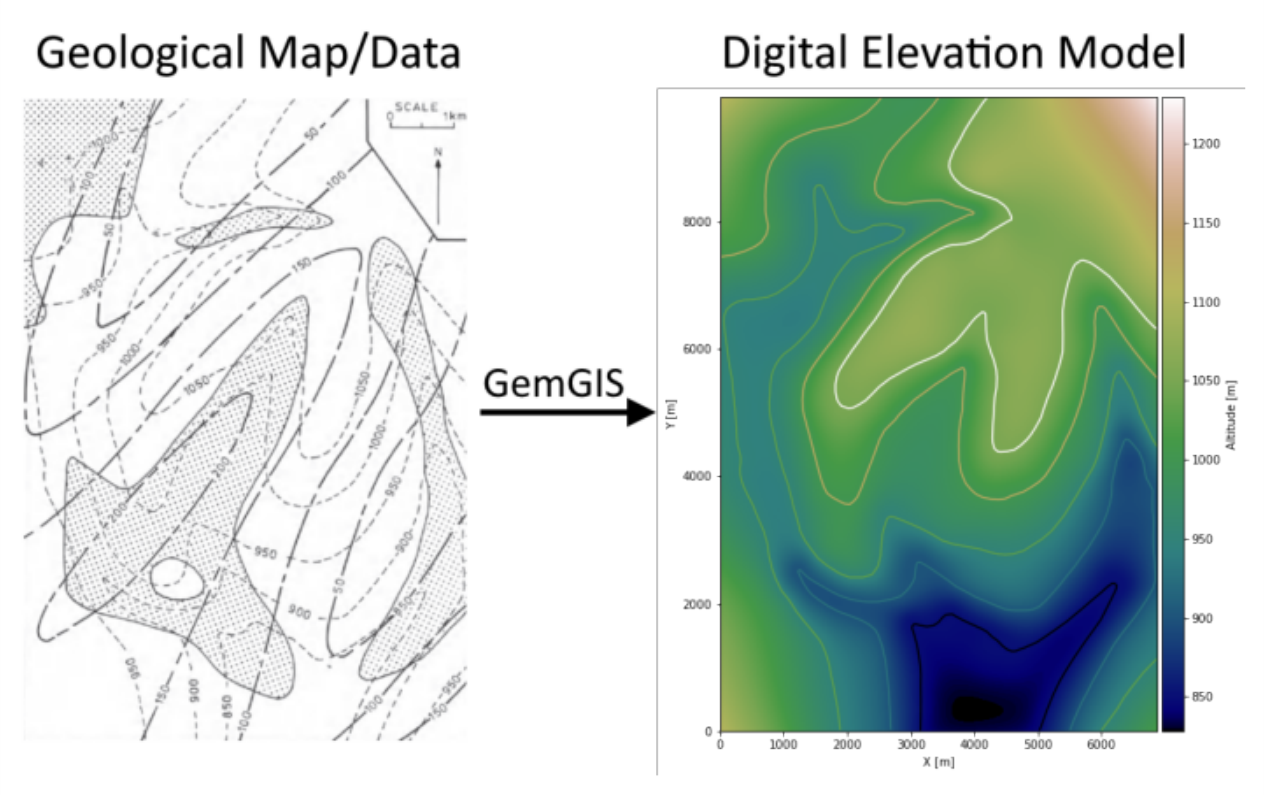
[4]:
```
file_path = 'data/example22/'
gg.download_gemgis_data.download_tutorial_data(filename="example22_coal_measures.zip",␣
↪dirpath=file_path)
```

```
Downloading file 'example22_coal_measures.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample22_coal_measures.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\example\data\example22'.
```

7.22.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the SciPy Radial Basis Function interpolation wrapped in GemGIS. The respective function used for that is gg.vector.interpolate_raster().

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example22.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo22.shp')
topo.head()
```

[6]:
```
     id     Z                                           geometry
0  None   950  LINESTRING (6001.096 12.580, 6136.515 195.703,...
1  None   900  LINESTRING (5508.664 11.041, 5542.518 117.222,...
2  None   850  LINESTRING (3151.144 4.886, 3160.377 221.864, ...
3  None   950  LINESTRING (1889.286 18.735, 1878.514 141.843,...
4  None  1000  LINESTRING (21.120 7446.770, 227.327 7482.163,...
```

Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

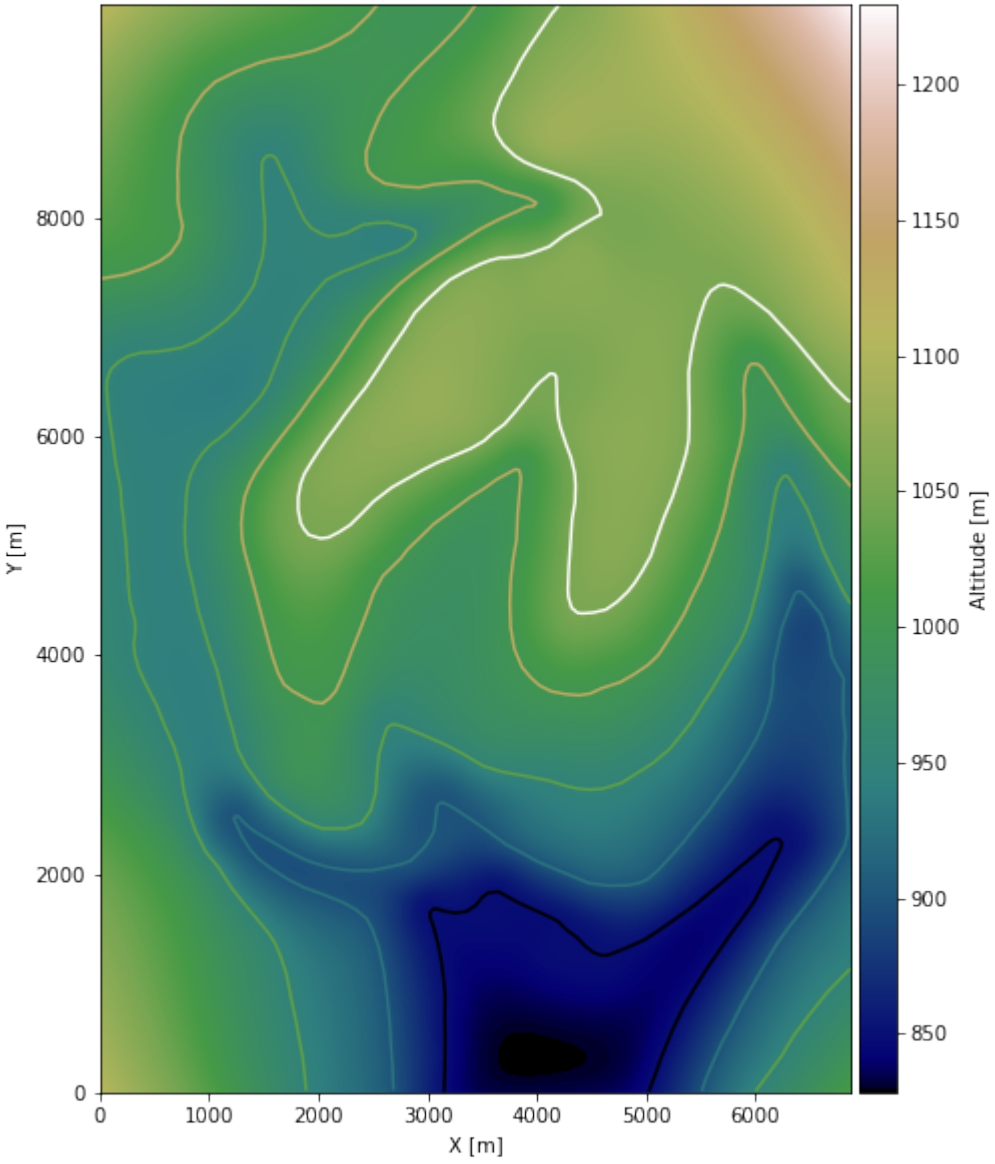
[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0, 6875, 0, 9954], cmap='gist_earth')
divider = make_axes_locatable(ax)
```

(continues on next page)

(continued from previous page)

```
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 6875)
ax.set_ylim(0, 9954)
```

[8]: (0.0, 9954.0)



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster22.tif', extent = [0, 6875, 0, 9954], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

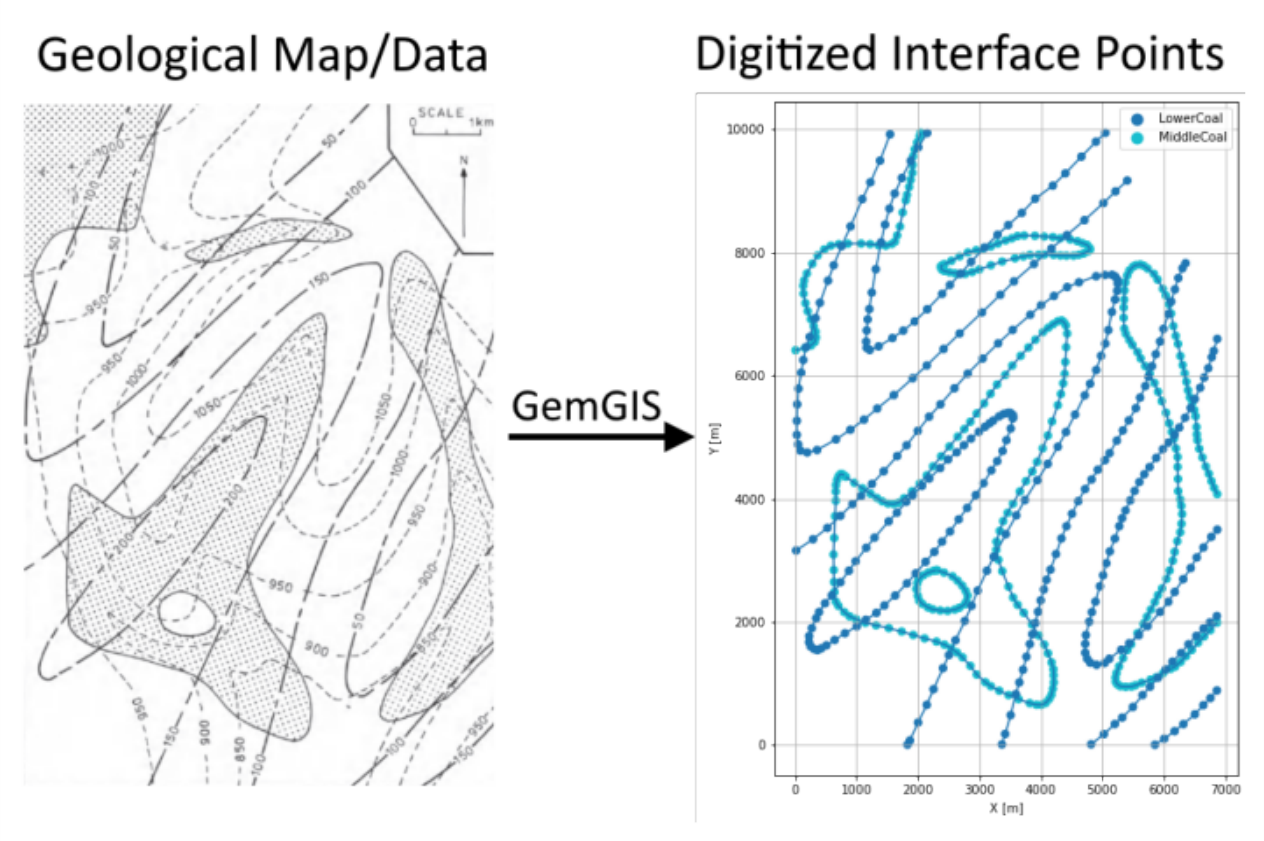
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster22.tif')
```

7.22.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example22.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces22.shp')
interfaces.head()
```

[11]:
```
     id   formation                                           geometry
0  None  MiddleCoal  LINESTRING (7.271 6412.662, 159.617 6449.594, ...
1  None  MiddleCoal  LINESTRING (2429.423 7811.477, 2543.298 7865.3...
2  None  MiddleCoal  LINESTRING (2314.009 2839.449, 2423.267 2800.9...
3  None  MiddleCoal  LINESTRING (6865.930 1990.004, 6756.672 1866.8...
4  None  MiddleCoal  LINESTRING (824.401 4356.757, 964.436 4259.809...
```

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

[12]:
```
     formation                    geometry       X       Y       Z
0   MiddleCoal     POINT (7.271 6412.662)    7.27 6412.66  951.72
205 MiddleCoal  POINT (6536.616 5044.623) 6536.62 5044.62  945.45
223 MiddleCoal  POINT (1772.333 4008.976) 1772.33 4008.98 1008.61
222 MiddleCoal  POINT (1675.385 3935.111) 1675.39 3935.11 1004.46
221 MiddleCoal  POINT (1603.059 3916.645) 1603.06 3916.65 1000.21
```

[13]:
```
interfaces_lowercoal = gpd.read_file(file_path + 'interfaces22a.shp')
interfaces_lowercoal['Z'] = interfaces_lowercoal['Z']+600
```

(continues on next page)

(continued from previous page)

```
interfaces_lowercoal.head()
```

[13]:
```
     id  formation    Z                                           geometry
0  None  LowerCoal  700  LINESTRING (1544.583 9919.704, 1381.465 9491.9...
1  None  LowerCoal  650  LINESTRING (2147.813 9941.247, 2003.161 9708.8...
2  None  LowerCoal  750  LINESTRING (16.504 3161.069, 290.419 3342.654,...
3  None  LowerCoal  800  LINESTRING (3540.473 5306.228, 3508.157 5141.5...
4  None  LowerCoal  700  LINESTRING (3365.044 11.041, 3417.365 180.315,...
```

[14]:
```
interfaces_coords_lowercoal = gg.vector.extract_xy(gdf=interfaces_lowercoal)
interfaces_coords_lowercoal = interfaces_coords_lowercoal.sort_values(by='formation',␣
↪ascending=False)
interfaces_coords_lowercoal.head()
```

[14]:
```
     formation      Z                   geometry       X       Y
0    LowerCoal 700.00  POINT (1544.583 9919.704) 1544.58 9919.70
206  LowerCoal 700.00  POINT (3951.346 2165.433) 3951.35 2165.43
213  LowerCoal 700.00  POINT (4414.541 3553.476) 4414.54 3553.48
212  LowerCoal 700.00  POINT (4332.981 3356.503) 4332.98 3356.50
211  LowerCoal 700.00  POINT (4245.267 3102.593) 4245.27 3102.59
```

[15]:
```
import pandas as pd
interfaces_coords = pd.concat([interfaces_coords, interfaces_coords_lowercoal])
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['MiddleCoal',
↪'LowerCoal'])].reset_index()
interfaces_coords
```

[15]:
```
     index   formation                   geometry       X       Y       Z
0        0  MiddleCoal     POINT (7.271 6412.662)    7.27 6412.66  951.72
1      205  MiddleCoal  POINT (6536.616 5044.623) 6536.62 5044.62  945.45
2      223  MiddleCoal  POINT (1772.333 4008.976) 1772.33 4008.98 1008.61
3      222  MiddleCoal  POINT (1675.385 3935.111) 1675.39 3935.11 1004.46
4      221  MiddleCoal  POINT (1603.059 3916.645) 1603.06 3916.65 1000.21
..     ...         ...                         ...     ...     ...     ...
635    104   LowerCoal  POINT (5250.137 7492.935) 5250.14 7492.94  750.00
636    103   LowerCoal  POINT (5247.059 7556.028) 5247.06 7556.03  750.00
637    102   LowerCoal  POINT (5205.510 7612.966) 5205.51 7612.97  750.00
638    101   LowerCoal  POINT (5156.267 7637.587) 5156.27 7637.59  750.00
639    312   LowerCoal   POINT (6862.853 886.647) 6862.85  886.65  750.00

[640 rows x 6 columns]
```

Plotting the Interface Points

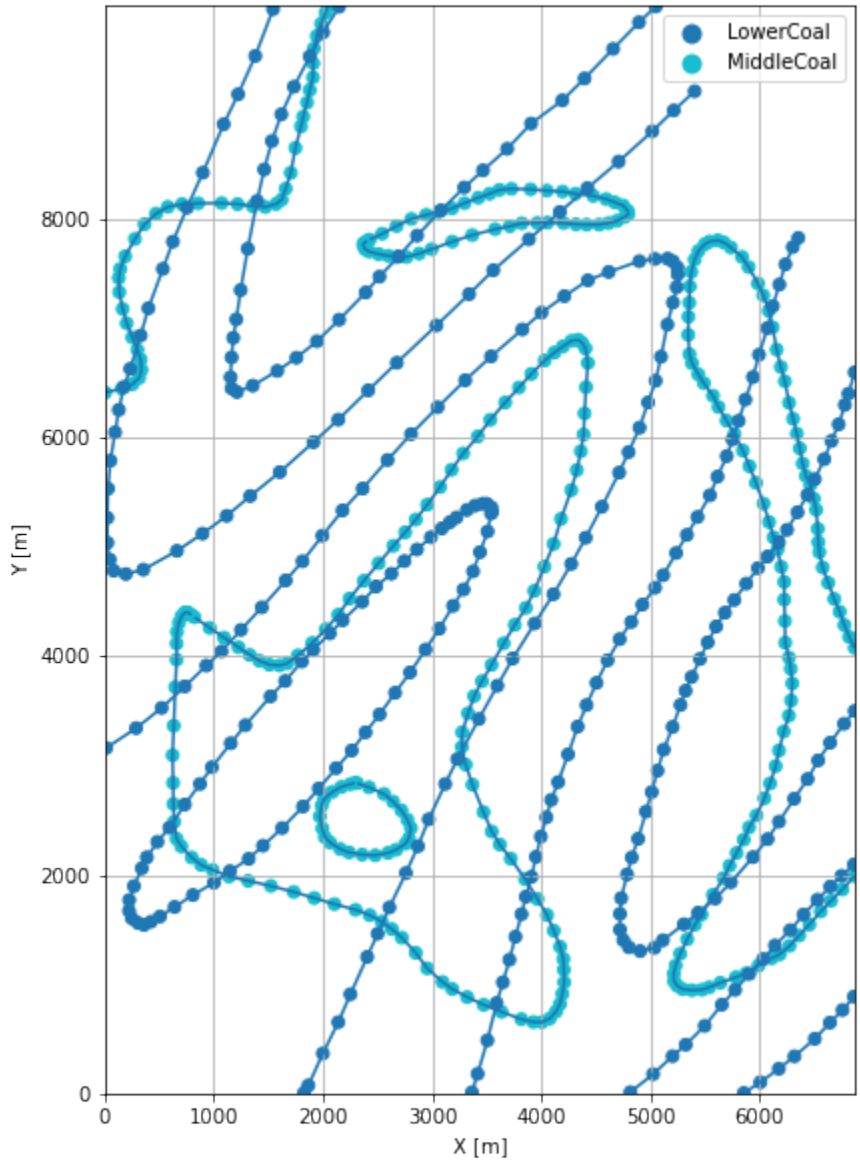
[16]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_lowercoal.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
```

(continues on next page)

(continued from previous page)

```
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 6875)
ax.set_ylim(0, 9954)
```

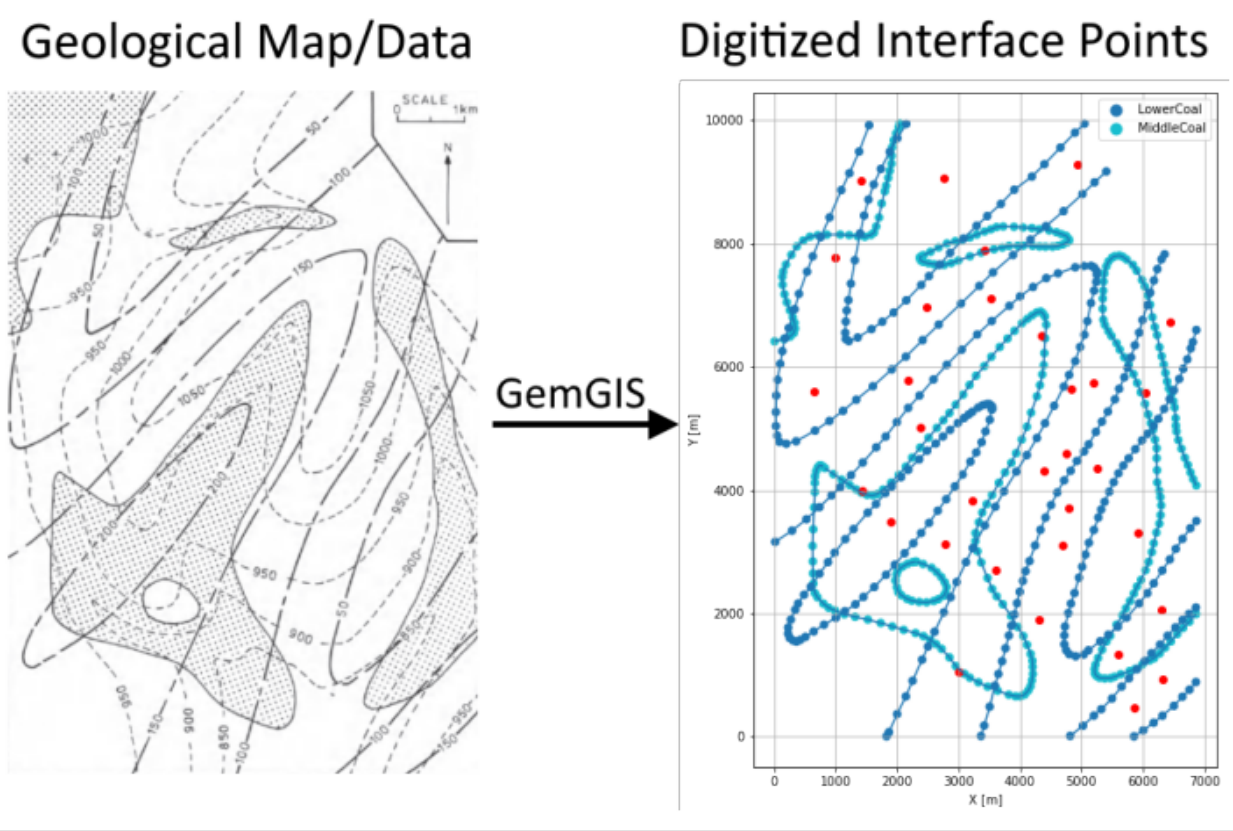
[16]: (0.0, 9954.0)



7.22.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for GemPy. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[17]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/orientations_example22.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[18]:
```
strikes = gpd.read_file(file_path + 'strikes22.shp')
strikes.head()
```

[18]:

	id	formation	Z	geometry
0	4	MiddleCoal1	1050	LINESTRING (5399.405 6655.800, 4311.437 5714.023)
1	3	MiddleCoal1	1000	LINESTRING (5844.133 5890.991, 3757.451 4279.814)
2	2	MiddleCoal1	950	LINESTRING (6116.510 5133.876, 3277.330 3110.287)
3	1	MiddleCoal1	900	LINESTRING (6231.924 4242.882, 3554.323 2383.949)

Calculate Orientations for each formation

[19]:
```
orientations_coal1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'MiddleCoal1'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_coal1
```

[19]:
```
   dip  azimuth       Z                   geometry  polarity    formation  \
0 3.79   144.85  925.00  POINT (4795.021 3717.749)      1.00  MiddleCoal1
1 4.24   143.74  975.00  POINT (4748.856 4603.742)      1.00  MiddleCoal1
2 3.59   141.62 1025.00  POINT (4828.107 5635.157)      1.00  MiddleCoal1

        X       Y
0 4795.02 3717.75
1 4748.86 4603.74
2 4828.11 5635.16
```

[20]:
```
gradients = gpd.read_file(file_path + 'gradients22.shp')
gradients.head()
```

[20]:
```
     id  formation  dZ                                              geometry
0  None  LowerCoal  50  LINESTRING (1199.880 9087.185, 1618.448 8957.922)
1  None  LowerCoal  50   LINESTRING (664.360 7856.104, 1307.600 7699.141)
2  None  LowerCoal  50   LINESTRING (119.607 4772.246, 1193.725 6431.128)
3  None  LowerCoal  50  LINESTRING (5207.049 8991.776, 4653.062 9558.074)
4  None  LowerCoal  50  LINESTRING (3668.198 7622.199, 3181.921 8173.107)
```

[21]:
```
orientations_lowercoal = gg.vector.extract_orientations_from_map(gdf=gradients)
orientations_lowercoal = gg.vector.extract_xyz(gdf=orientations_lowercoal, dem=topo_
↪raster)
orientations_lowercoal
```

[21]:
```
    formation                   geometry  azimuth  dip       X       Y  \
0   LowerCoal  POINT (1409.164 9022.553)   107.16 6.51 1409.16 9022.55
1   LowerCoal   POINT (985.980 7777.623)   103.71 4.32  985.98 7777.62
2   LowerCoal   POINT (656.666 5601.687)    32.92 1.45  656.67 5601.69
3   LowerCoal  POINT (4930.056 9274.925)   315.63 3.61 4930.06 9274.92
4   LowerCoal  POINT (3425.059 7897.653)   318.57 3.89 3425.06 7897.65
5   LowerCoal  POINT (2487.899 6960.493)   314.81 4.42 2487.90 6960.49
6   LowerCoal  POINT (3529.701 7108.222)   315.90 5.13 3529.70 7108.22
7   LowerCoal  POINT (2172.434 5789.427)   316.92 4.88 2172.43 5789.43
8   LowerCoal  POINT (2386.335 5018.463)   313.43 3.99 2386.33 5018.46
9   LowerCoal  POINT (1443.019 3995.127)   309.88 4.69 1443.02 3995.13
10  LowerCoal  POINT (1890.825 3479.611)    40.77 0.00 1890.82 3479.61
11  LowerCoal  POINT (4346.831 6503.454)    36.23 1.02 4346.83 6503.45
12  LowerCoal  POINT (5197.816 5730.951)   113.56 3.41 5197.82 5730.95
13  LowerCoal  POINT (4383.763 4315.208)   122.29 3.91 4383.76 4315.21
14  LowerCoal  POINT (3226.547 3827.392)   121.01 3.92 3226.55 3827.39
15  LowerCoal  POINT (2777.203 3124.137)   115.19 3.33 2777.20 3124.14
16  LowerCoal  POINT (3598.949 2710.186)   114.22 2.93 3598.95 2710.19
17  LowerCoal  POINT (3004.953 1058.999)   113.13 2.27 3004.95 1059.00
18  LowerCoal  POINT (4319.132 1906.906)   103.48 3.39 4319.13 1906.91
19  LowerCoal  POINT (4683.839 3113.365)   108.71 3.47 4683.84 3113.36
20  LowerCoal  POINT (5265.525 4350.601)   123.11 4.23 5265.53 4350.60
```

(continues on next page)

(continued from previous page)

```
21  LowerCoal  POINT (6033.412 5581.682)   118.78 3.50 6033.41 5581.68
22  LowerCoal  POINT (6448.902 6720.432)   115.24 3.20 6448.90 6720.43
23  LowerCoal  POINT (5911.843 3310.338)   116.91 0.00 5911.84 3310.34
24  LowerCoal  POINT (6308.866 2057.713)   316.05 4.03 6308.87 2057.71
25  LowerCoal  POINT (5600.995 1343.686)   327.04 4.28 5600.99 1343.69
26  LowerCoal   POINT (6331.949 934.352)   311.05 3.35 6331.95  934.35
27  LowerCoal   POINT (5867.216 477.313)   320.58 3.76 5867.22  477.31
28  LowerCoal  POINT (2760.276 9053.330)   116.16 0.00 2760.28 9053.33

    polarity       Z
0       1.00  973.93
1       1.00  981.51
2       1.00  945.90
3       1.00 1090.31
4       1.00  995.92
5       1.00 1019.35
6       1.00 1067.49
7       1.00 1062.76
8       1.00 1034.22
9       1.00  989.47
10      1.00  997.89
11      1.00 1051.44
12      1.00 1057.67
13      1.00 1047.62
14      1.00  976.97
15      1.00  939.69
16      1.00  924.46
17      1.00  859.77
18      1.00  890.47
19      1.00  965.82
20      1.00 1004.28
21      1.00  962.60
22      1.00 1044.93
23      1.00  908.53
24      1.00  858.20
25      1.00  844.53
26      1.00  931.46
27      1.00  915.96
28      1.00 1003.53
```

Merging Orientations

[22]:
```
import pandas as pd
orientations = pd.concat([orientations_coal1]).reset_index()
orientations['formation'] = ['MiddleCoal', 'MiddleCoal', 'MiddleCoal']
orientations = pd.concat([orientations, orientations_lowercoal]).reset_index()
orientations = orientations[orientations['formation'].isin(['LowerCoal', 'MiddleCoal'])]
orientations.head()
```

[22]:
```
   level_0  index  dip  azimuth       Z                   geometry  polarity  \
0        0   0.00 3.79   144.85  925.00  POINT (4795.021 3717.749)      1.00
```

(continues on next page)

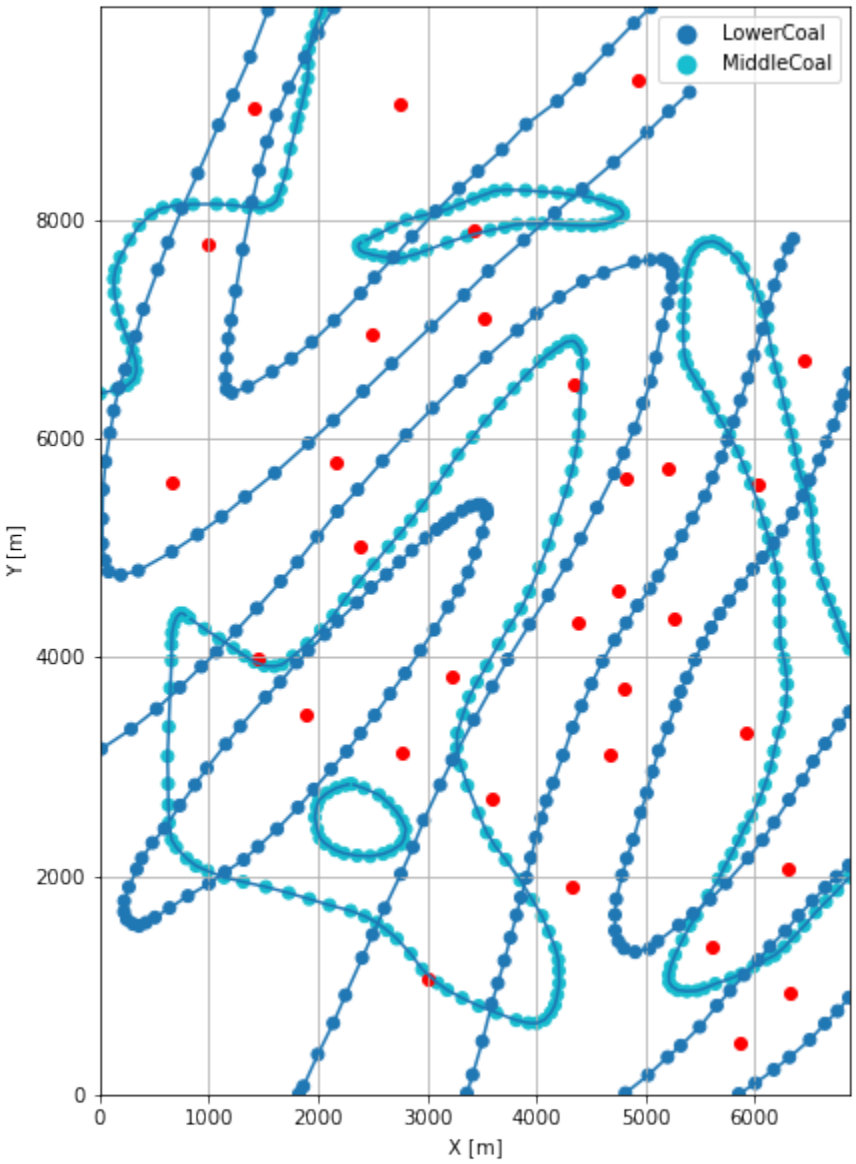
(continued from previous page)

```
1        1   1.00 4.24   143.74  975.00  POINT (4748.856 4603.742)      1.00
2        2   2.00 3.59   141.62 1025.00  POINT (4828.107 5635.157)      1.00
3        0    NaN 6.51   107.16  973.93  POINT (1409.164 9022.553)      1.00
4        1    NaN 4.32   103.71  981.51   POINT (985.980 7777.623)      1.00

    formation       X       Y
0  MiddleCoal 4795.02 3717.75
1  MiddleCoal 4748.86 4603.74
2  MiddleCoal 4828.11 5635.16
3   LowerCoal 1409.16 9022.55
4   LowerCoal  985.98 7777.62
```

Plotting the Orientations

[23]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_lowercoal.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 6875)
ax.set_ylim(0, 9954)
```

[23]:
```
(0.0, 9954.0)
```



7.22.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[24]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[25]:
```
geo_model = gp.create_model('Model22')
geo_model
```

[25]:
```
Model22  2022-04-05 11:20
```

Initiate Data

[26]:
```
gp.init_data(geo_model, [0, 6875, 0, 9954, 500, 1250], [100, 100, 100],
             surface_points_df=interfaces_coords[interfaces_coords['Z'] != 0],
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[26]:
```
Model22  2022-04-05 11:20
```

Model Surfaces

[27]:
```
geo_model.surfaces
```

[27]:

	surface	series	order_surfaces	color	id
0	MiddleCoal	Default series	1	#015482	1
1	LowerCoal	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[28]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('MiddleCoal', 'LowerCoal'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[28]:

	surface	series	order_surfaces	color	id
0	MiddleCoal	Strata1	1	#015482	1
1	LowerCoal	Strata1	2	#9f0052	2
2	Basement	Strata1	3	#ffbe00	3

Adding additional Orientations

[29]:
```
geo_model.add_orientations(X=380, Y=9550, Z=1025, surface='MiddleCoal', orientation=[107,
↪ 15, 1])
geo_model.add_orientations(X=1700, Y=9775, Z=1025, surface='MiddleCoal',␣
↪orientation=[107, 15, 1])
```

[29]:

	X	Y	Z	G_x	G_y	G_z	smooth	surface
0	4795.02	3717.75	925.00	0.04	-0.05	1.00	0.01	MiddleCoal
1	4748.86	4603.74	975.00	0.04	-0.06	1.00	0.01	MiddleCoal

(continues on next page)

(continued from previous page)

```
2  4828.11 5635.16 1025.00  0.04 -0.05 1.00    0.01  MiddleCoal
32  380.00 9550.00 1025.00  0.25 -0.08 0.97    0.01  MiddleCoal
33 1700.00 9775.00 1025.00  0.25 -0.08 0.97    0.01  MiddleCoal
3  1409.16 9022.55  973.93  0.11 -0.03 0.99    0.01   LowerCoal
4   985.98 7777.62  981.51  0.07 -0.02 1.00    0.01   LowerCoal
5   656.67 5601.69  945.90  0.01  0.02 1.00    0.01   LowerCoal
6  4930.06 9274.92 1090.31 -0.04  0.05 1.00    0.01   LowerCoal
7  3425.06 7897.65  995.92 -0.04  0.05 1.00    0.01   LowerCoal
8  2487.90 6960.49 1019.35 -0.05  0.05 1.00    0.01   LowerCoal
9  3529.70 7108.22 1067.49 -0.06  0.06 1.00    0.01   LowerCoal
10 2172.43 5789.43 1062.76 -0.06  0.06 1.00    0.01   LowerCoal
11 2386.33 5018.46 1034.22 -0.05  0.05 1.00    0.01   LowerCoal
12 1443.02 3995.13  989.47 -0.06  0.05 1.00    0.01   LowerCoal
13 1890.82 3479.61  997.89  0.00  0.00 1.00    0.01   LowerCoal
14 4346.83 6503.45 1051.44  0.01  0.01 1.00    0.01   LowerCoal
15 5197.82 5730.95 1057.67  0.05 -0.02 1.00    0.01   LowerCoal
16 4383.76 4315.21 1047.62  0.06 -0.04 1.00    0.01   LowerCoal
17 3226.55 3827.39  976.97  0.06 -0.04 1.00    0.01   LowerCoal
18 2777.20 3124.14  939.69  0.05 -0.02 1.00    0.01   LowerCoal
19 3598.95 2710.19  924.46  0.05 -0.02 1.00    0.01   LowerCoal
20 3004.95 1059.00  859.77  0.04 -0.02 1.00    0.01   LowerCoal
21 4319.13 1906.91  890.47  0.06 -0.01 1.00    0.01   LowerCoal
22 4683.84 3113.36  965.82  0.06 -0.02 1.00    0.01   LowerCoal
23 5265.53 4350.60 1004.28  0.06 -0.04 1.00    0.01   LowerCoal
24 6033.41 5581.68  962.60  0.05 -0.03 1.00    0.01   LowerCoal
25 6448.90 6720.43 1044.93  0.05 -0.02 1.00    0.01   LowerCoal
26 5911.84 3310.34  908.53  0.00  0.00 1.00    0.01   LowerCoal
27 6308.87 2057.71  858.20 -0.05  0.05 1.00    0.01   LowerCoal
28 5600.99 1343.69  844.53 -0.04  0.06 1.00    0.01   LowerCoal
29 6331.95  934.35  931.46 -0.04  0.04 1.00    0.01   LowerCoal
30 5867.22  477.31  915.96 -0.04  0.05 1.00    0.01   LowerCoal
31 2760.28 9053.33 1003.53  0.00  0.00 1.00    0.01   LowerCoal
```

Showing the Number of Data Points

[30]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[30]:
```
      surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0  MiddleCoal  Strata1               1  #015482   1                327          ␣
↪ 5
1   LowerCoal  Strata1               2  #9f0052   2                313          ␣
↪29
2    Basement  Strata1               3  #ffbe00   3                  0          ␣
↪ 0
```

Loading Digital Elevation Model

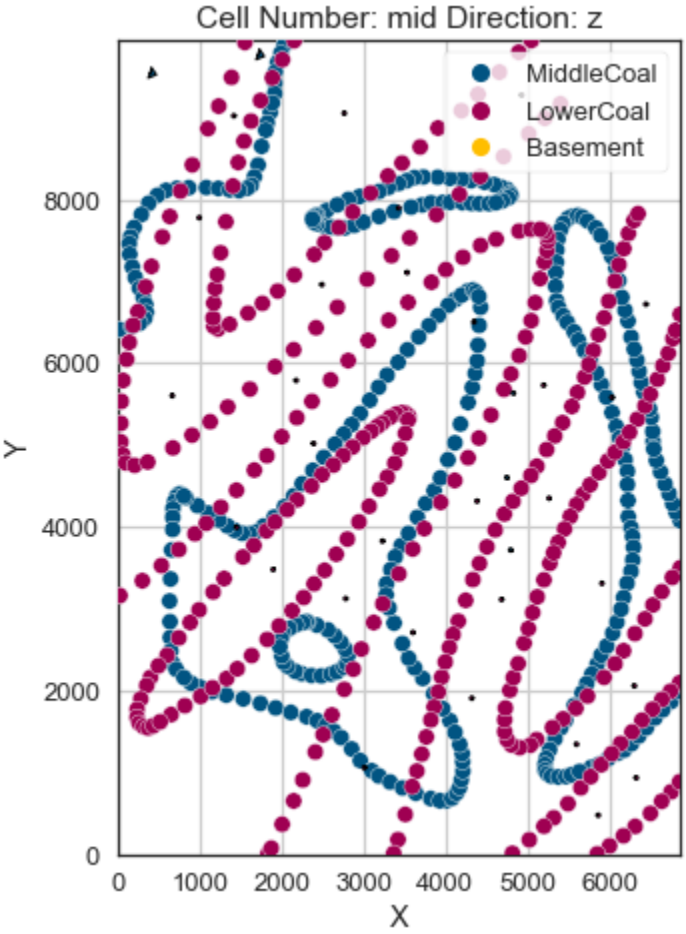
[31]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster22.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

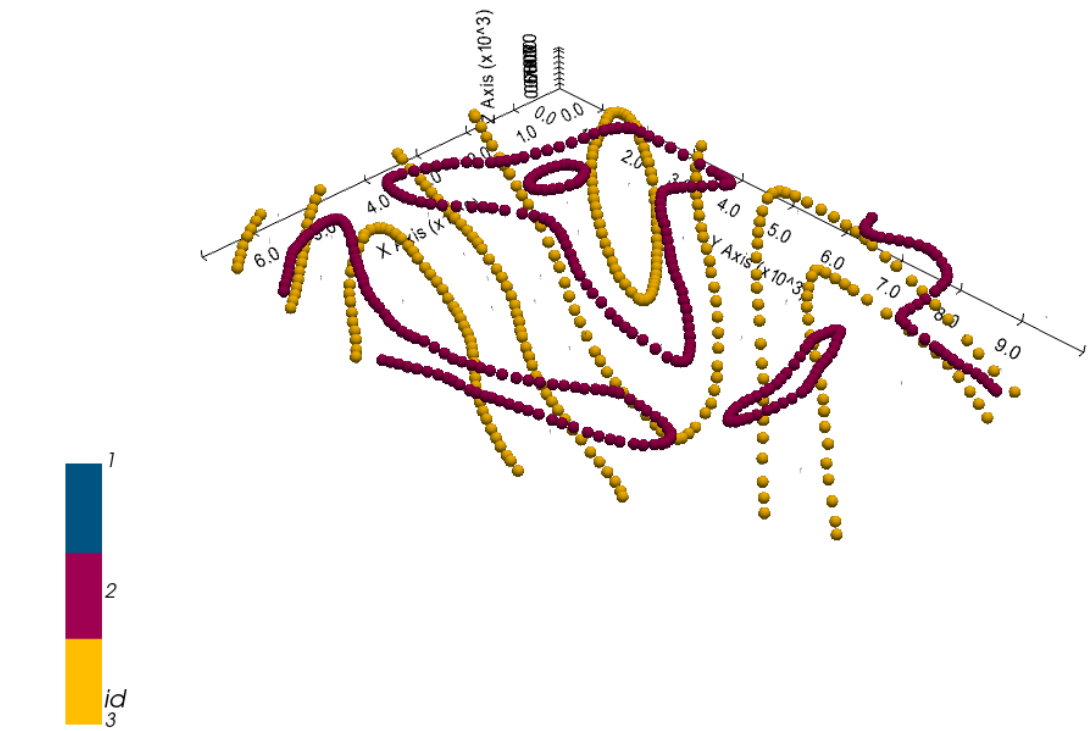
[31]:
```
Grid Object. Values:
array([[  34.375     ,   49.77      ,  503.75      ],
       [  34.375     ,   49.77      ,  511.25      ],
       [  34.375     ,   49.77      ,  518.75      ],
       ...,
       [6869.98906706, 9928.98994975, 1228.36560059],
       [6869.98906706, 9938.99396985, 1228.96081543],
       [6869.98906706, 9948.99798995, 1229.55761719]])
```

Plotting Input Data

[32]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[33]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[33]:
```
<gempy.plot.vista.GemPyToVista at 0x151a458fca0>
```

Setting the Interpolator

[34]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                     values
range             12120.65
$C_o$           3497862.88
drift equations        [3]
```

[34]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1519d051df0>
```

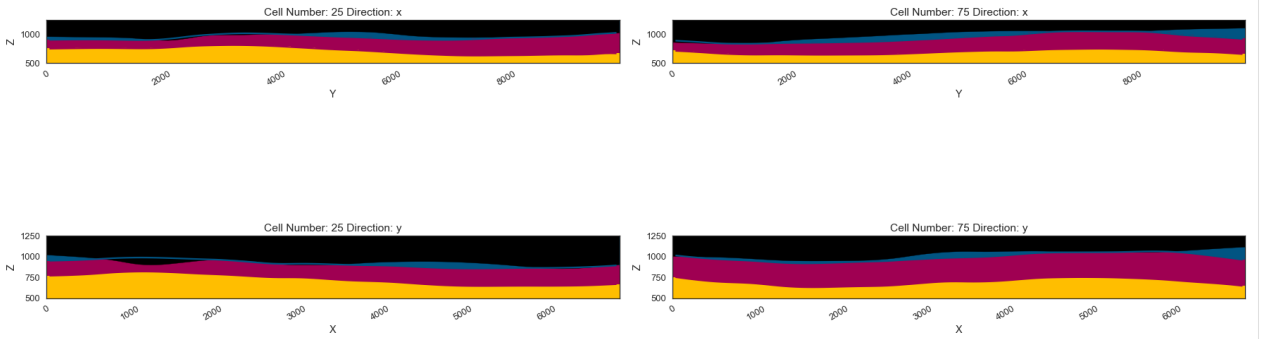
Computing Model

[35]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

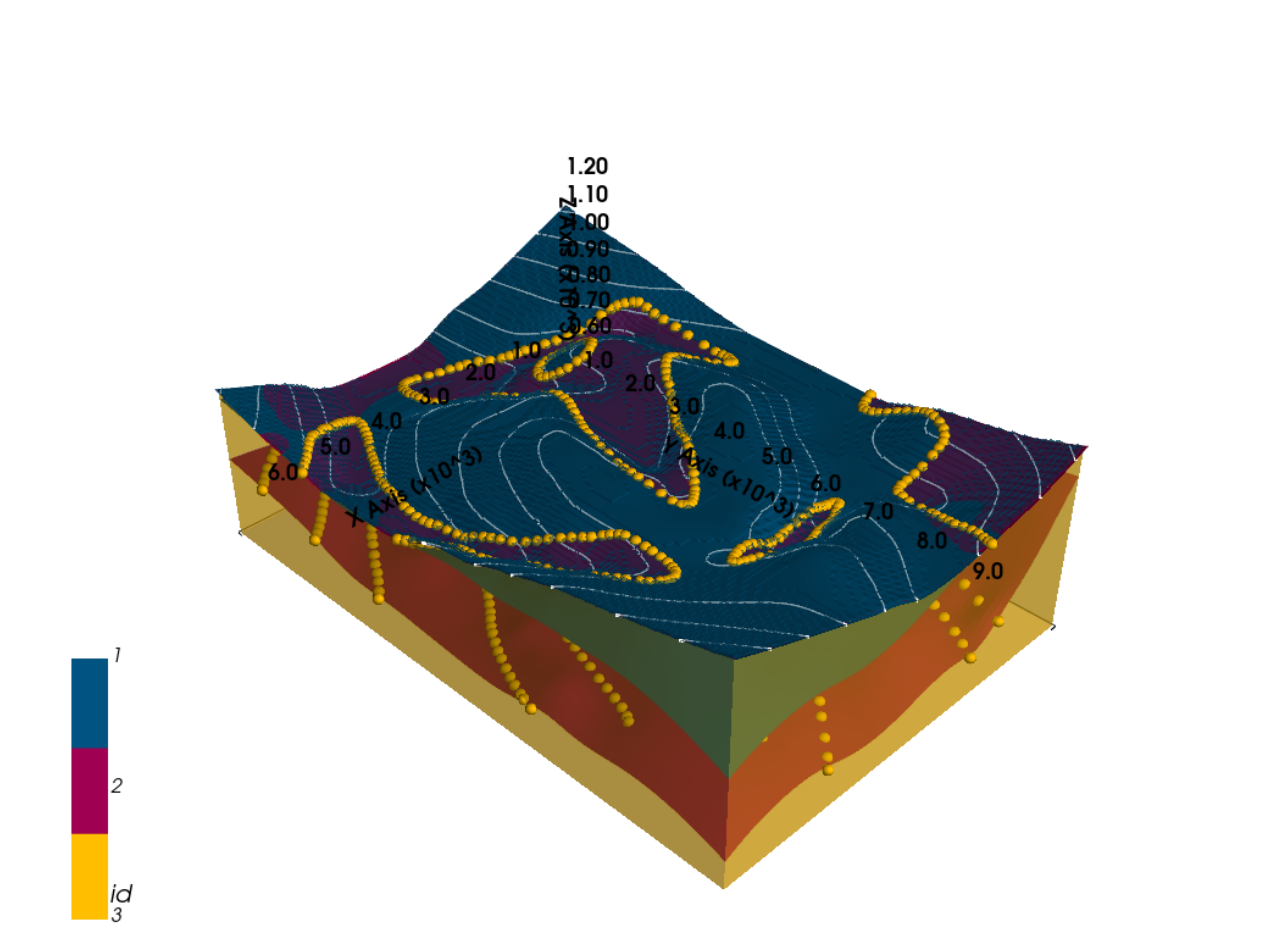
[36]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[36]:
```
<gempy.plot.visualization_2d.Plot2D at 0x151a5bcdbb0>
```



Plotting 3D Model

[37]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True, ve=5)
```



[ ]:

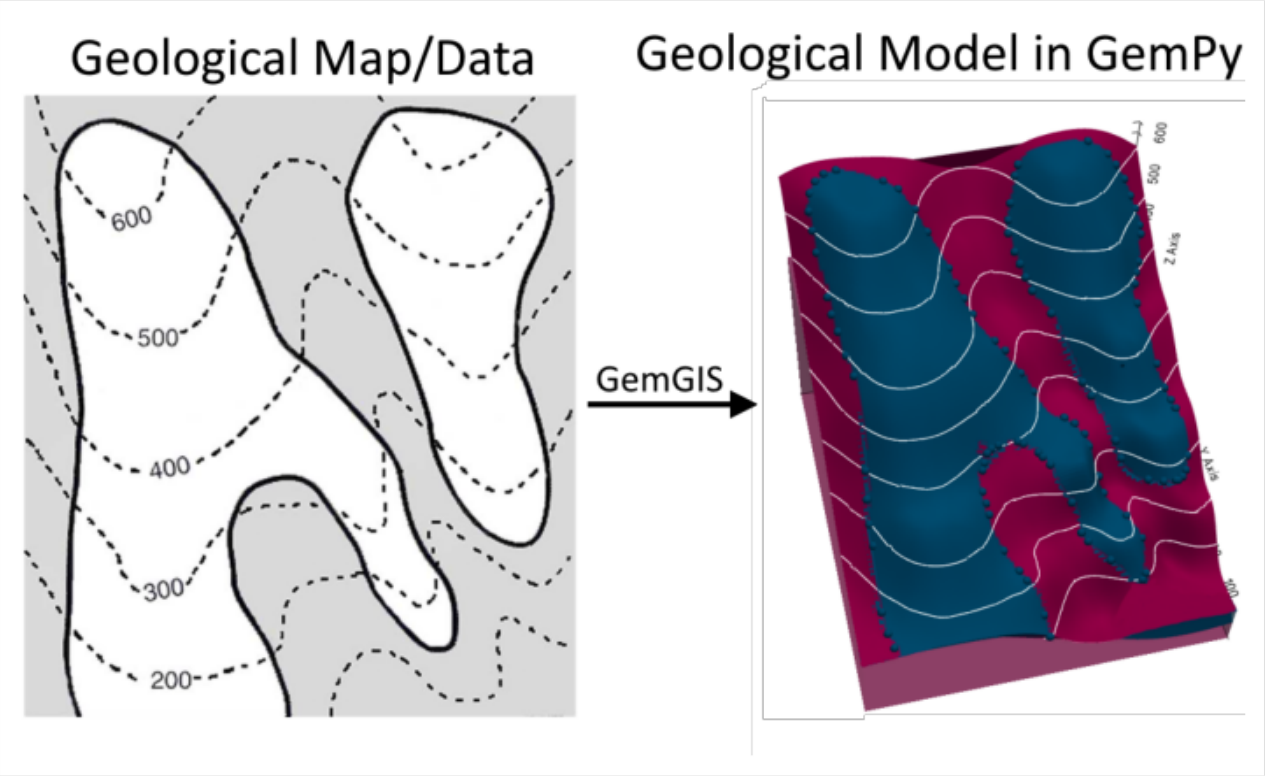
7.23 Example 23 - Planar dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 700 m wide (W-E extent) and 788 m high (N-S extent). The model represents southwards dipping planar layers.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../../getting_started/images/cover_example23.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.23.1 Licensing

7.23.2 Import GemGIS

If you have installed `GemGIS` via pip, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.23.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded form an external server using pooch. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

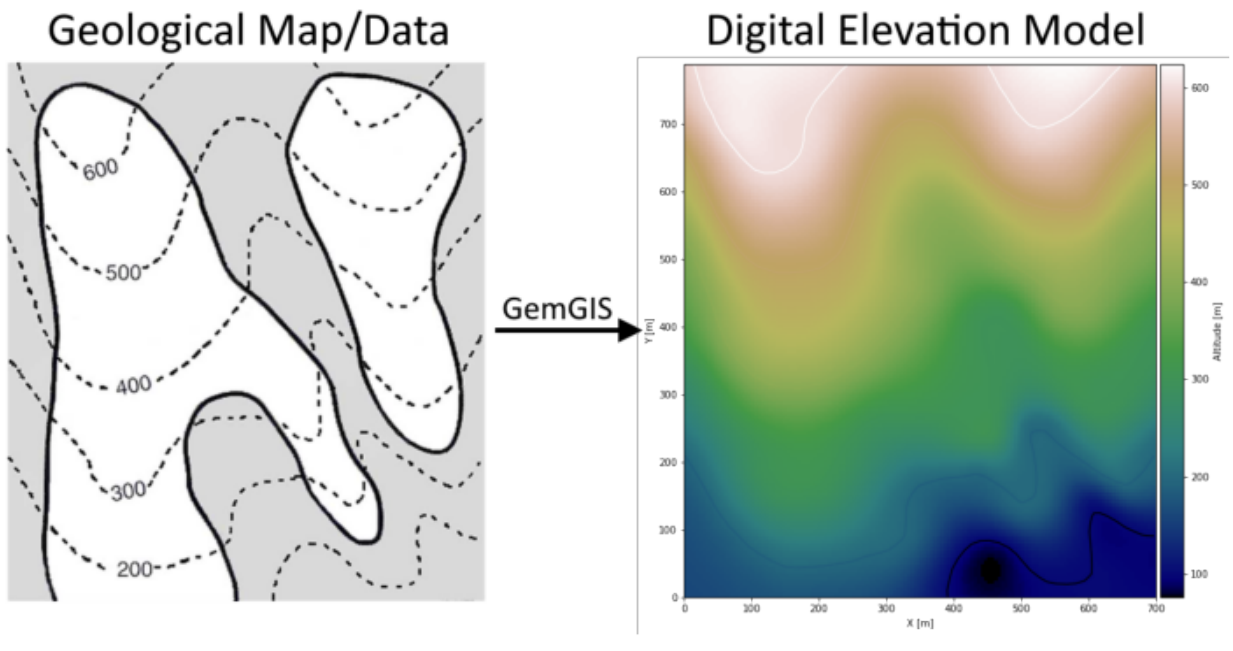
[4]:
```
file_path = 'data/example23/'
gg.download_gemgis_data.download_tutorial_data(filename="example23_planar_dipping_layers.
↪zip", dirpath=file_path)
```

```
Downloading file 'example23_planar_dipping_layers.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2Fexample23_planar_dipping_layers.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\example\data\example23'.
```

7.23.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the SciPy Radial Basis Function interpolation wrapped in GemGIS. The respective function used for that is gg.vector.interpolate_raster().

[5]:
```
img = mpimg.imread('../../getting_started/images/dem_example23.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo23.shp')
topo.head()
```

[6]:
```
     id    Z                                           geometry
0  None  600  LINESTRING (15.116 787.112, 22.878 758.803, 29...
1  None  600  LINESTRING (450.485 786.656, 457.791 773.642, ...
2  None  500  LINESTRING (1.189 660.177, 11.235 642.598, 21...
3  None  400  LINESTRING (0.790 532.842, 12.776 506.360, 26...
4  None  300  LINESTRING (3.872 351.115, 20.310 331.025, 34...
```

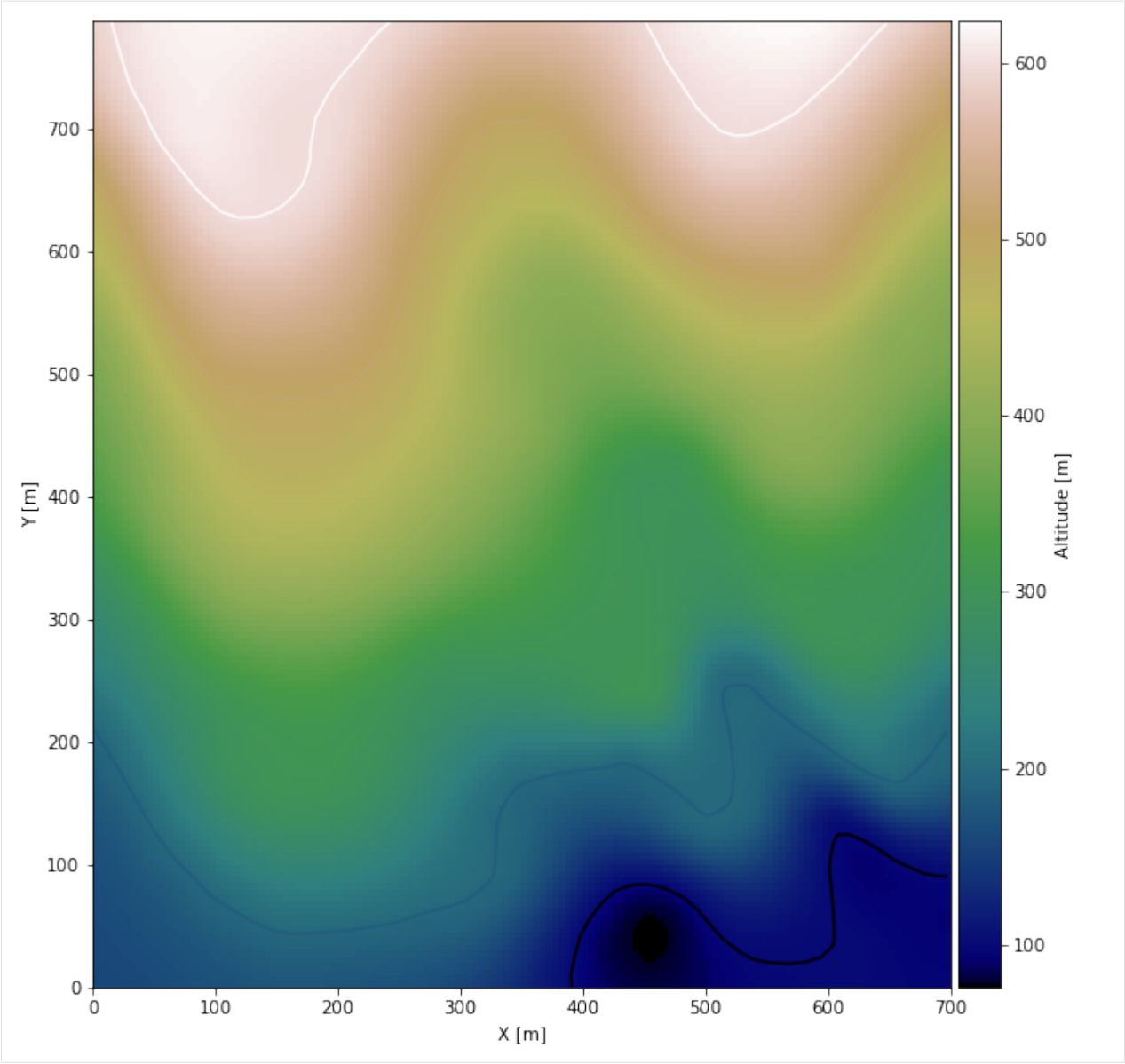
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10,10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0,700,0,788], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

[8]:
```
Text(98.40090121128625, 0.5, 'Y [m]')
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as as raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_raster, path = file_path +' raster23.tif', extent = [0, 700, 0, 788], crs =' EPSG : 4326', overwrite_file = True)$

Opening Raster

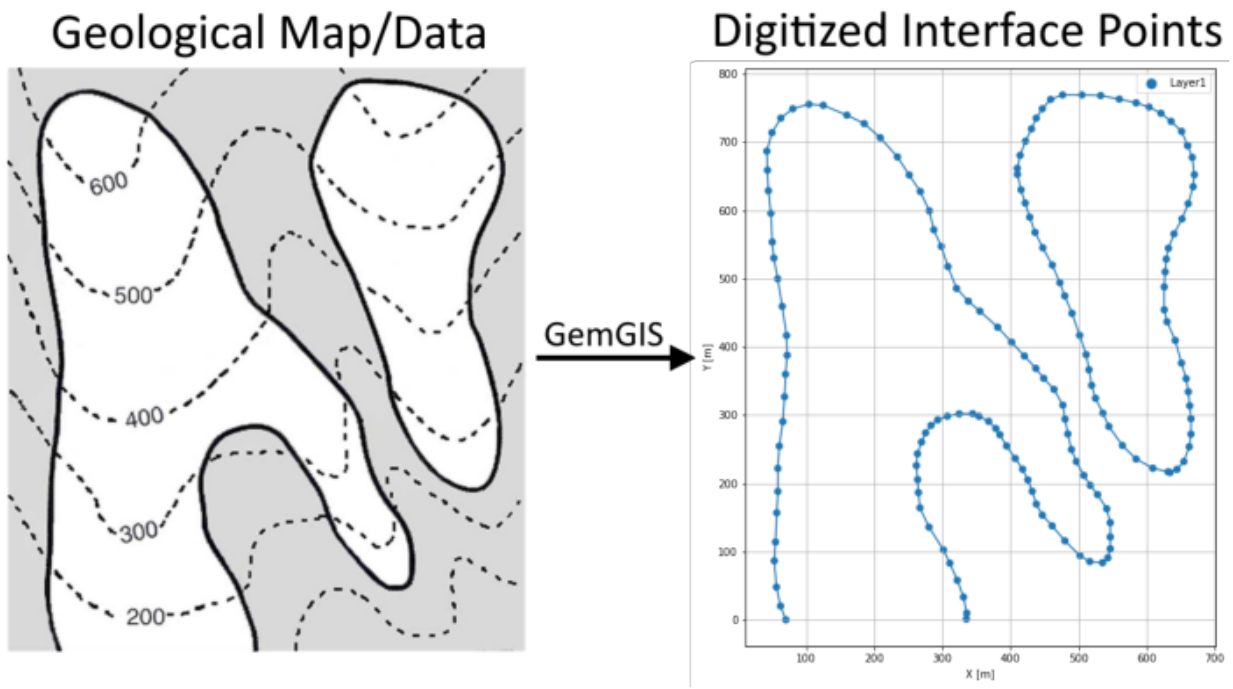
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster23.tif')
```

7.23.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../../getting_started/images/interfaces_example23.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces23.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Layer1	LINESTRING (70.166 0.062, 62.317 20.308, 56.38...
1	None	Layer1	LINESTRING (631.470 216.874, 609.097 222.125, ...

Extracting XY coordinates from Digital Elevation Model

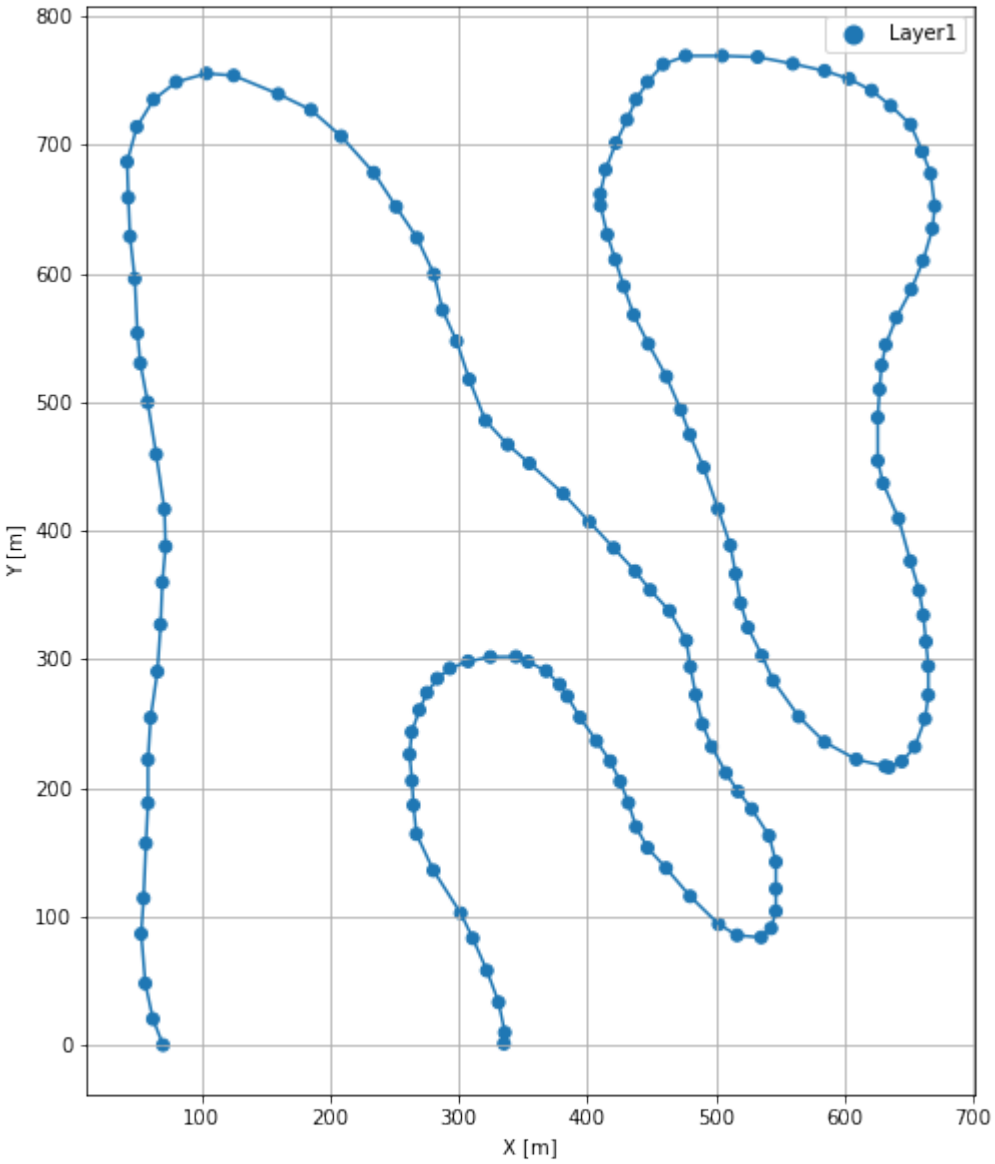
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

[12]:
```
    formation                 geometry      X      Y      Z
0      Layer1     POINT (70.166 0.062)  70.17   0.06 166.32
107    Layer1  POINT (472.573 494.031) 472.57 494.03 393.99
100    Layer1  POINT (524.854 324.632) 524.85 324.63 289.36
101    Layer1  POINT (519.147 343.581) 519.15 343.58 294.85
102    Layer1  POINT (515.265 366.411) 515.27 366.41 304.13
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
```

[13]:
```
Text(116.99284374939245, 0.5, 'Y [m]')
```



7.23.6 Orientations from Strike Lines

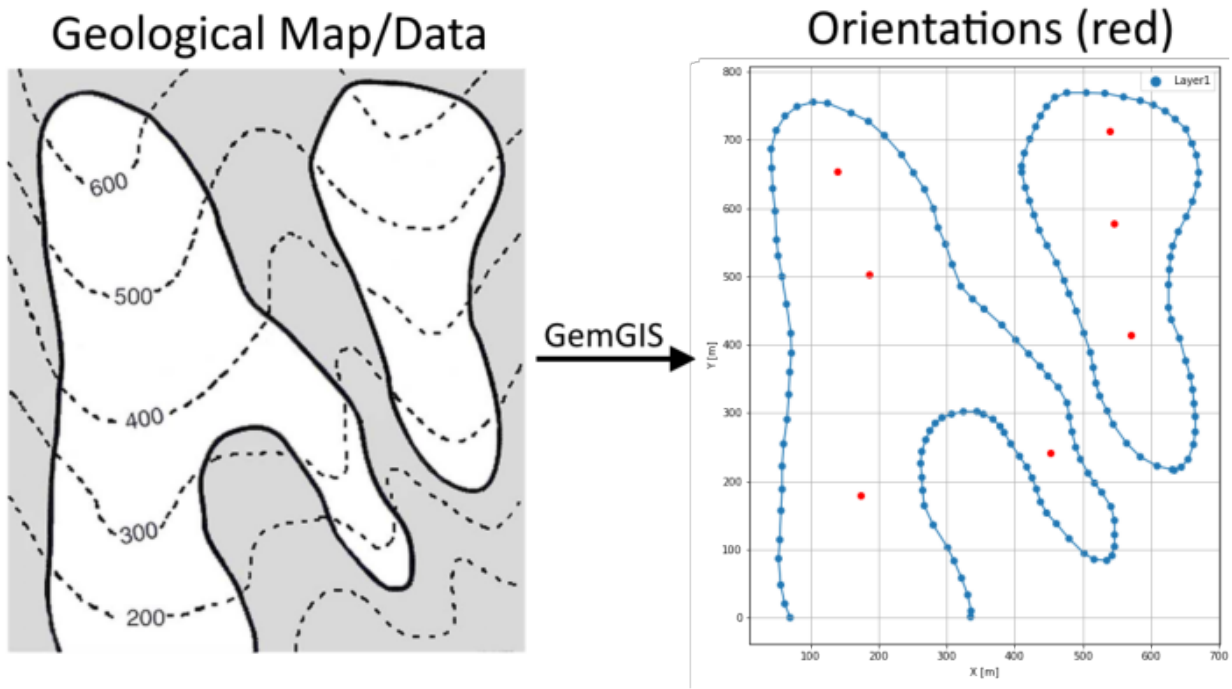
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
img = mpimg.imread('../../getting_started/images/orientations_example23.png')
plt.figure(figsize=(10, 10))
```

(continues on next page)

(continued from previous page)

```
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes23.shp')
strikes
```

[15]:

	id	Z	formation	geometry
0	3	600	Layer1a	LINESTRING (46.044 702.206, 190.042 725.406)
1	2	500	Layer1a	LINESTRING (49.510 572.074, 271.908 618.474)
2	1	400	Layer1a	LINESTRING (69.244 368.876, 355.641 452.075)
3	2	300	Layer1b	LINESTRING (62.310 272.344, 266.042 247.811)
4	1	200	Layer1b	LINESTRING (54.844 118.745, 312.974 79.279)
5	2	300	Layer1c	LINESTRING (391.907 261.677, 459.106 340.610)
6	1	200	Layer1c	LINESTRING (436.706 178.478, 524.172 187.011)
7	4	600	Layer1d	LINESTRING (463.373 765.139, 612.705 747.006)
8	3	500	Layer1d	LINESTRING (412.173 672.340, 670.304 668.073)
9	2	400	Layer1d	LINESTRING (471.906 495.275, 625.505 476.075)
10	1	300	Layer1d	LINESTRING (519.906 352.343, 662.838 332.077)

Calculate Orientations for each formation

[16]:
```
orientations_layer1a = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Layer1a'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_layer1a
```

[16]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 28.63   165.42 450.00  POINT (186.576 502.875)      1.00   Layer1a 186.58
1 39.48   168.98 550.00  POINT (139.376 654.540)      1.00   Layer1a 139.38

       Y
0 502.87
1 654.54
```

[17]:
```
orientations_layer1b = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Layer1b'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_layer1b
```

[17]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 33.10   187.99 250.00  POINT (174.042 179.545)      1.00   Layer1b 174.04

       Y
0 179.54
```

[18]:
```
orientations_layer1c = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Layer1c'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_layer1c
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 46.62   148.69 250.00  POINT (452.973 241.944)      1.00   Layer1c 452.97

       Y
0 241.94
```

[19]:
```
orientations_layer1d = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Layer1d'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_layer1d
```

[19]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 36.35   187.56 350.00  POINT (570.038 413.942)      1.00   Layer1d 570.04
1 29.60   182.57 450.00  POINT (544.972 577.941)      1.00   Layer1d 544.97
2 52.06   182.45 550.00  POINT (539.639 713.139)      1.00   Layer1d 539.64

       Y
0 413.94
1 577.94
2 713.14
```

Merging Orientations

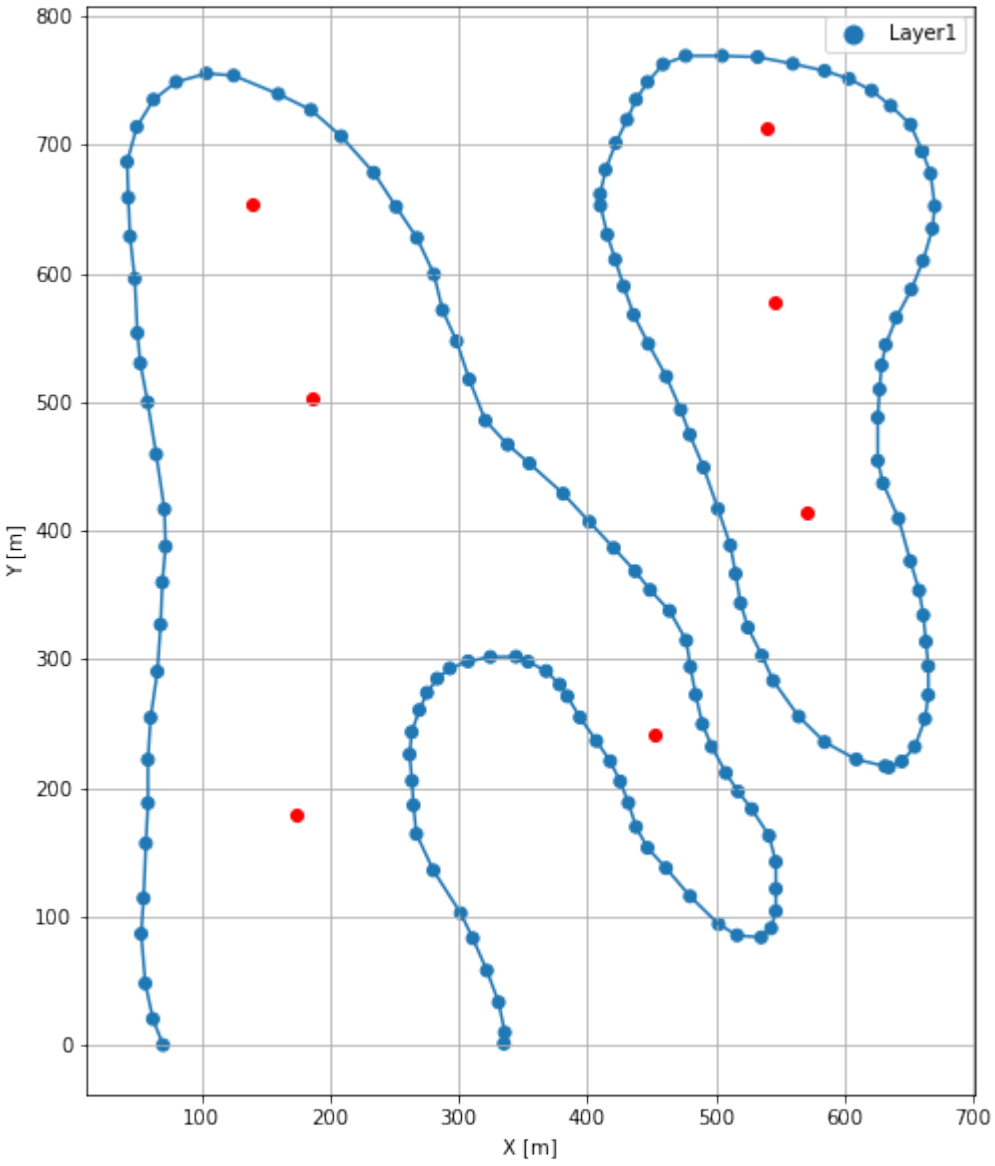
[20]:
```
import pandas as pd
orientations = pd.concat([orientations_layer1a, orientations_layer1b, orientations_
↪layer1c, orientations_layer1d]).reset_index()
orientations['formation'] = 'Layer1'
orientations
```

[20]:
```
   index   dip  azimuth      Z                 geometry  polarity formation  \
0      0 28.63   165.42 450.00  POINT (186.576 502.875)      1.00    Layer1
1      1 39.48   168.98 550.00  POINT (139.376 654.540)      1.00    Layer1
2      0 33.10   187.99 250.00  POINT (174.042 179.545)      1.00    Layer1
3      0 46.62   148.69 250.00  POINT (452.973 241.944)      1.00    Layer1
4      0 36.35   187.56 350.00  POINT (570.038 413.942)      1.00    Layer1
5      1 29.60   182.57 450.00  POINT (544.972 577.941)      1.00    Layer1
6      2 52.06   182.45 550.00  POINT (539.639 713.139)      1.00    Layer1

       X      Y
0 186.58 502.87
1 139.38 654.54
2 174.04 179.54
3 452.97 241.94
4 570.04 413.94
5 544.97 577.94
6 539.64 713.14
```

Plotting the Orientations

[21]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
```

[21]:
```
Text(116.99284374939245, 0.5, 'Y [m]')
```



7.23.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[22]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[23]:
```
geo_model = gp.create_model('Model23')
geo_model
```

[23]:
```
Model23  2022-04-07 09:15
```

Initiate Data

[24]:
```
gp.init_data(geo_model, [0,700,0,788,0,400], [100,100,100],
             surface_points_df = interfaces_coords[interfaces_coords['Z']!=0].
↪sample(n=100),
             orientations_df = orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[24]:
```
Model23  2022-04-07 09:15
```

Model Surfaces

[25]:
```
geo_model.surfaces
```

[25]:
```
  surface          series  order_surfaces    color  id
0  Layer1  Default series               1  #015482   1
```

Mapping the Stack to Surfaces

[26]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Layer1'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[26]:
```
    surface   series  order_surfaces    color  id
0    Layer1  Strata1               1  #015482   1
1  Basement  Strata1               2  #9f0052   2
```

Showing the Number of Data Points

[27]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[27]:
```
    surface   series  order_surfaces    color  id  No. of Interfaces  No. of Orientations
0    Layer1  Strata1               1  #015482   1                100                    7
1  Basement  Strata1               2  #9f0052   2                  0                    0
```

Loading Digital Elevation Model

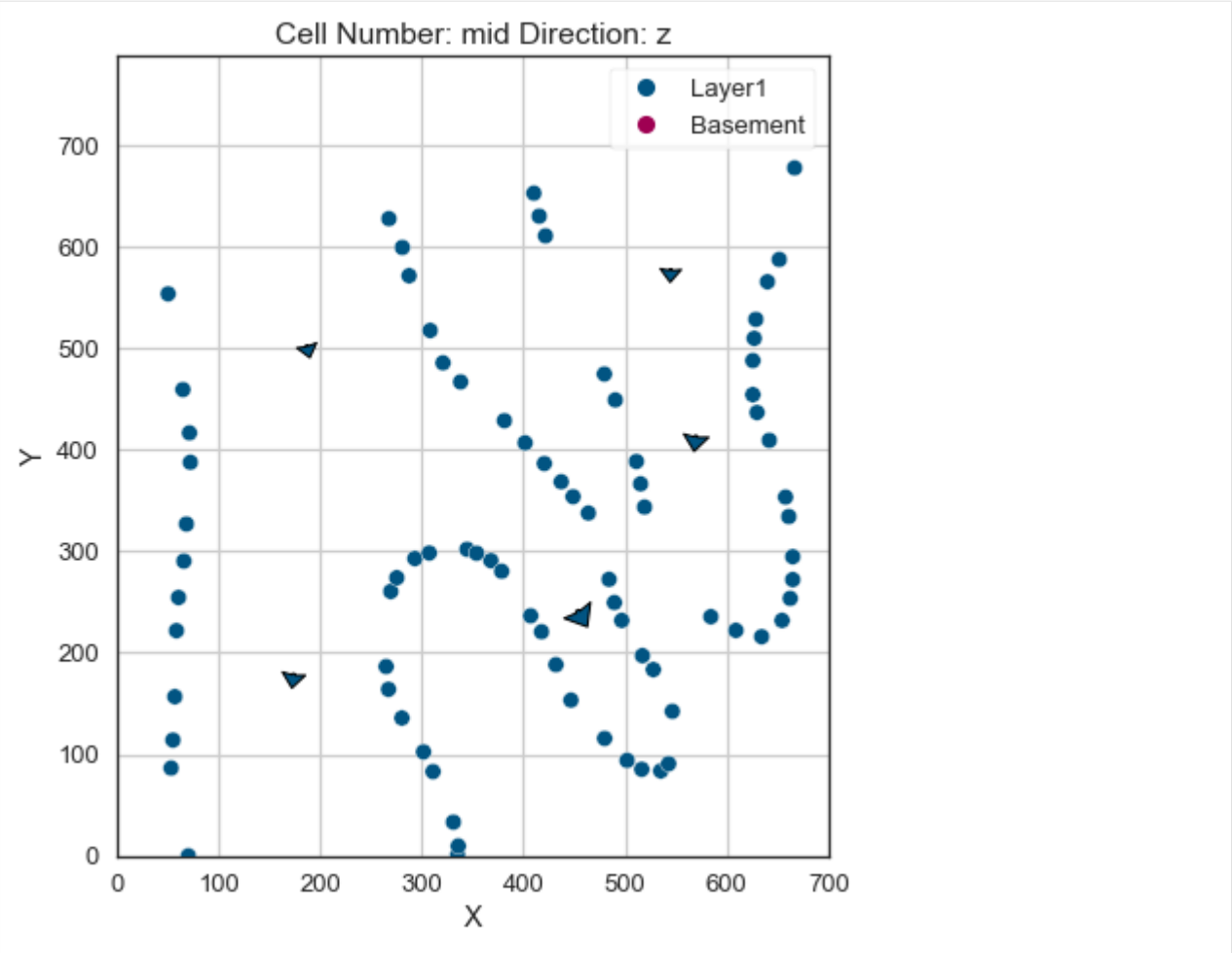
[28]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster23.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

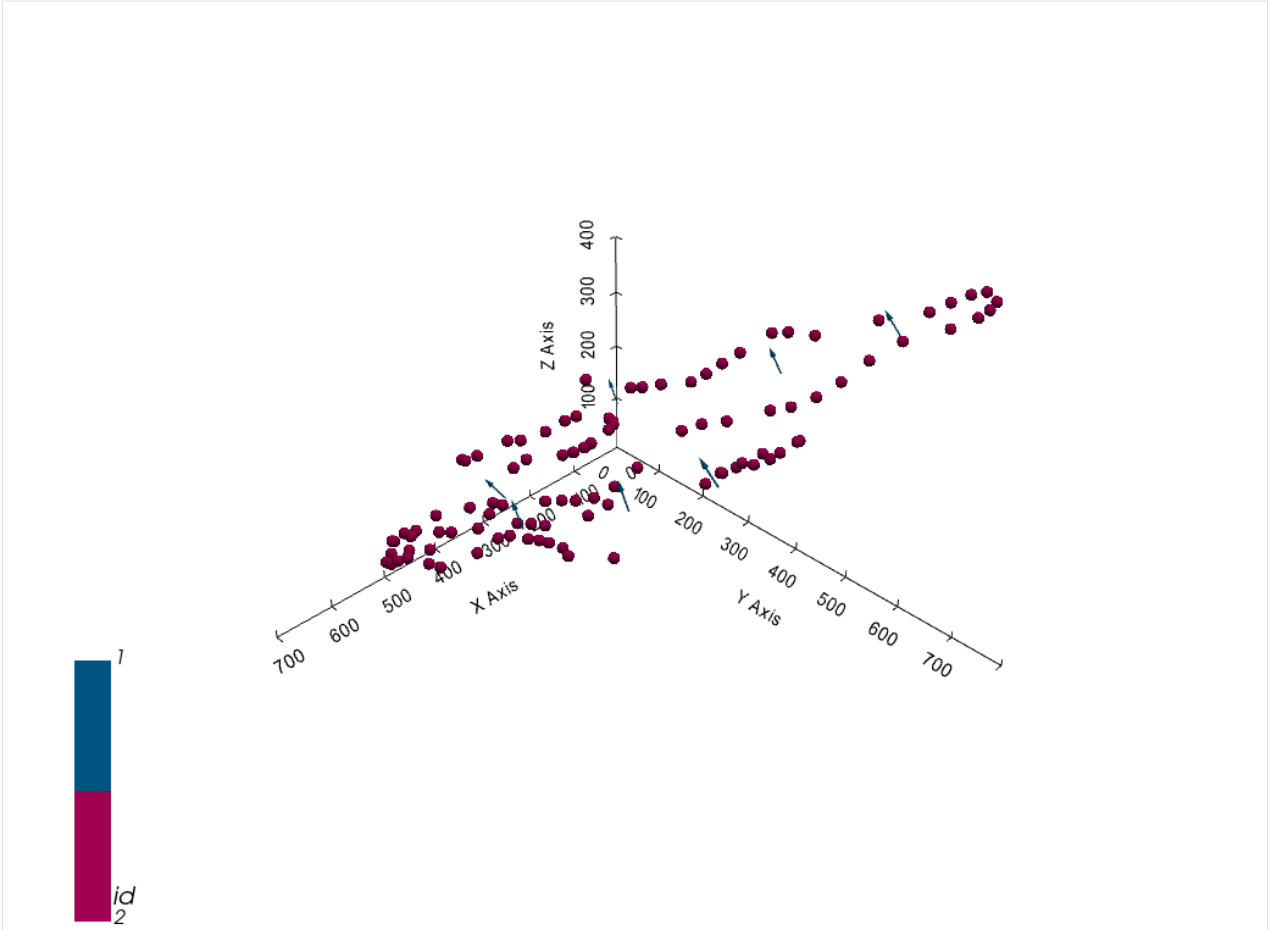
[28]:
```
Grid Object. Values:
array([[  3.5       ,   3.94      ,   2.        ],
       [  3.5       ,   3.94      ,   6.        ],
       [  3.5       ,   3.94      ,  10.        ],
       ...,
       [697.5       , 775.53164557, 559.49066162],
       [697.5       , 780.51898734, 563.62854004],
       [697.5       , 785.50632911, 567.61871338]])
```

Plotting Input Data

[29]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[30]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[30]:
```
<gempy.plot.vista.GemPyToVista at 0x167bcc5b100>
```

Setting the Interpolator

[31]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                   values
range             1127.36
$C_o$            30260.57
drift equations      [3]
```

[31]:
```
<gempy.core.interpolator.InterpolatorModel at 0x167b4d80430>
```

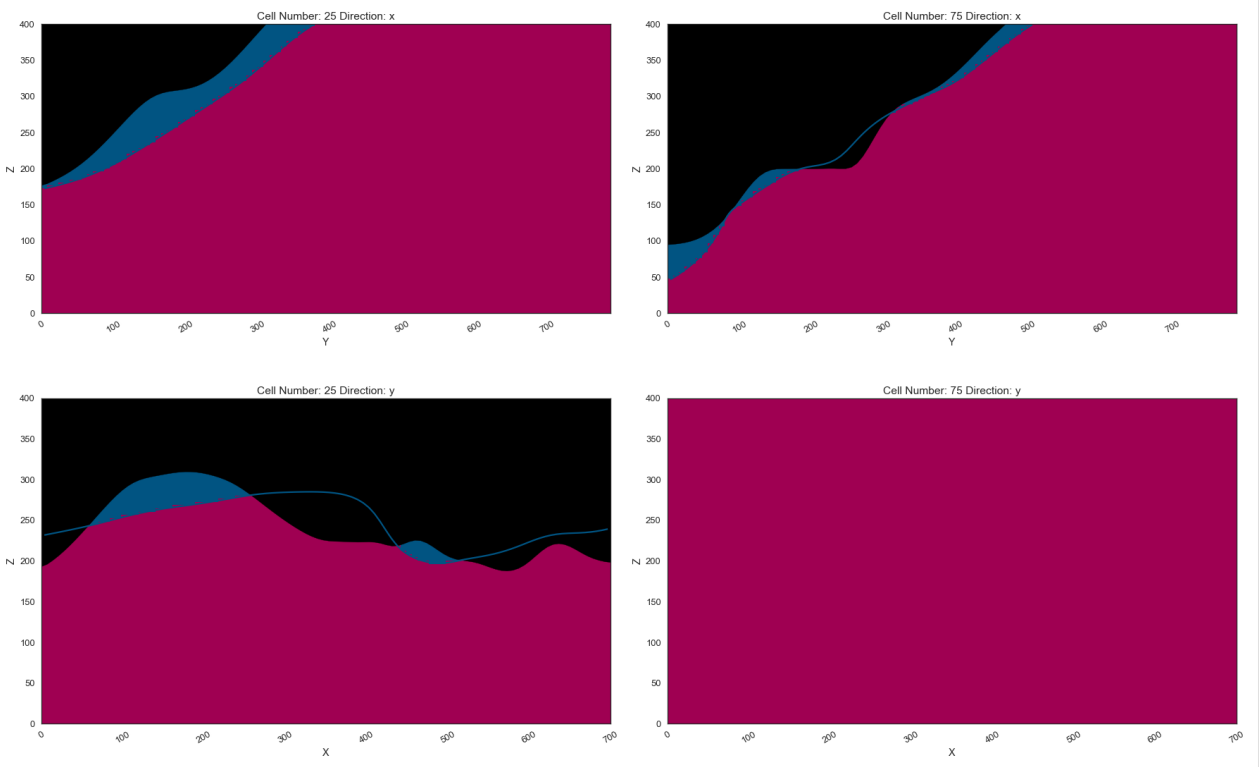
Computing Model

[32]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

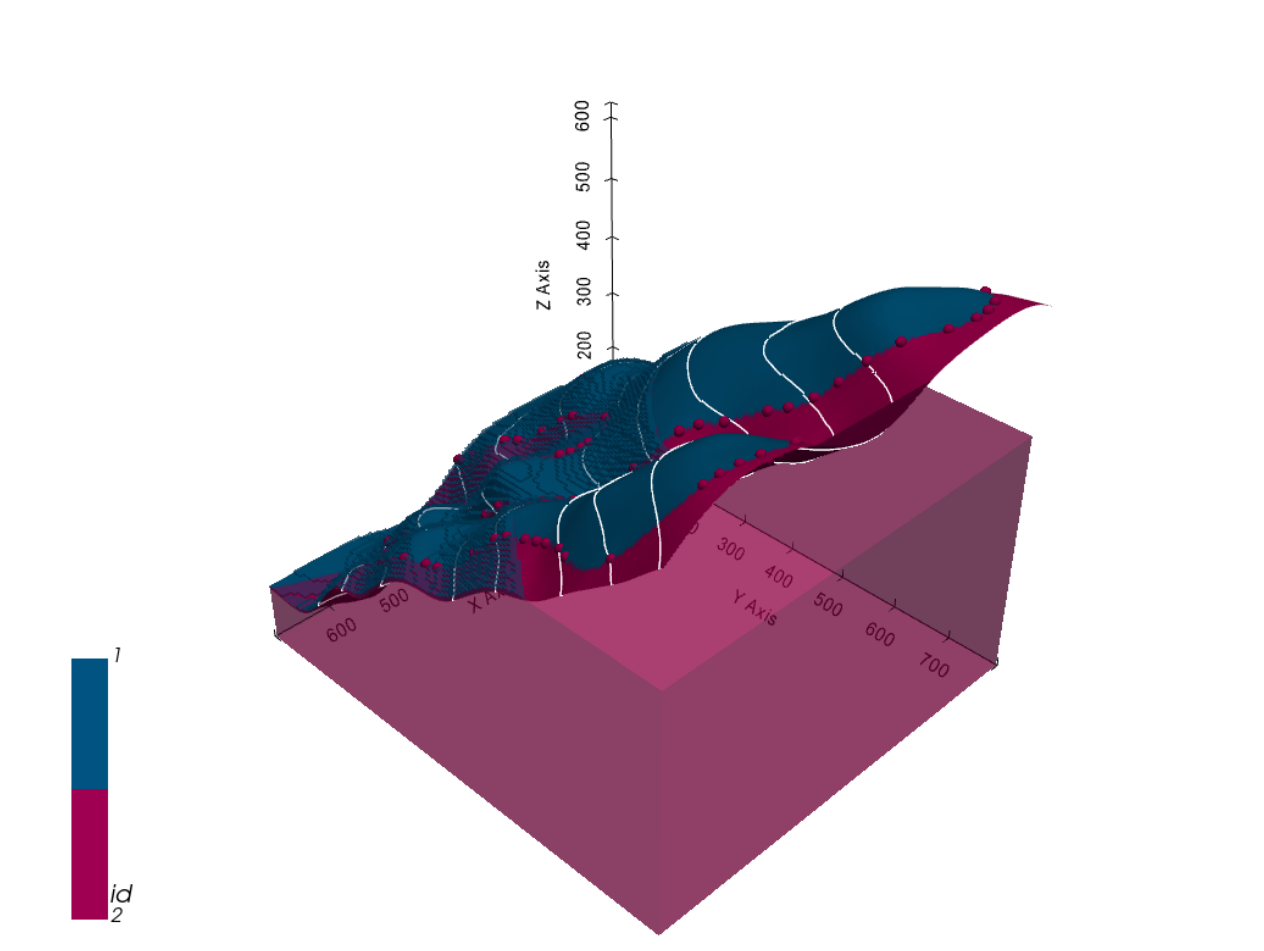
[33]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25,75,25,75], show_
↪topography=True, show_data=False)
```

[33]:
```
<gempy.plot.visualization_2d.Plot2D at 0x167bc470160>
```



Plotting 3D Model

[34]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

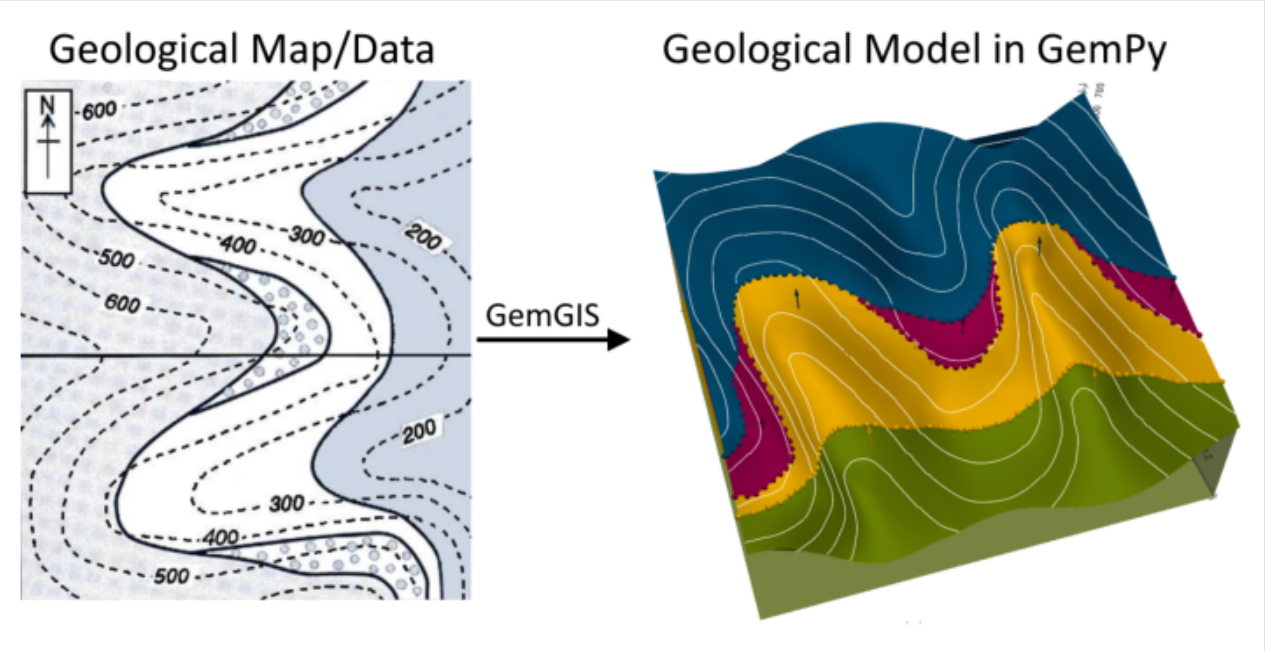
7.24 Example 24- Unconformable Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 1368 m wide (W-E extent) and 1568 m high (N-S extent). The model represents unconformable layers that dip to the west.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example24.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.24.1 Licensing

7.24.2 Import GemGIS

If you have installed `GemGIS` via pip, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.24.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded form an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

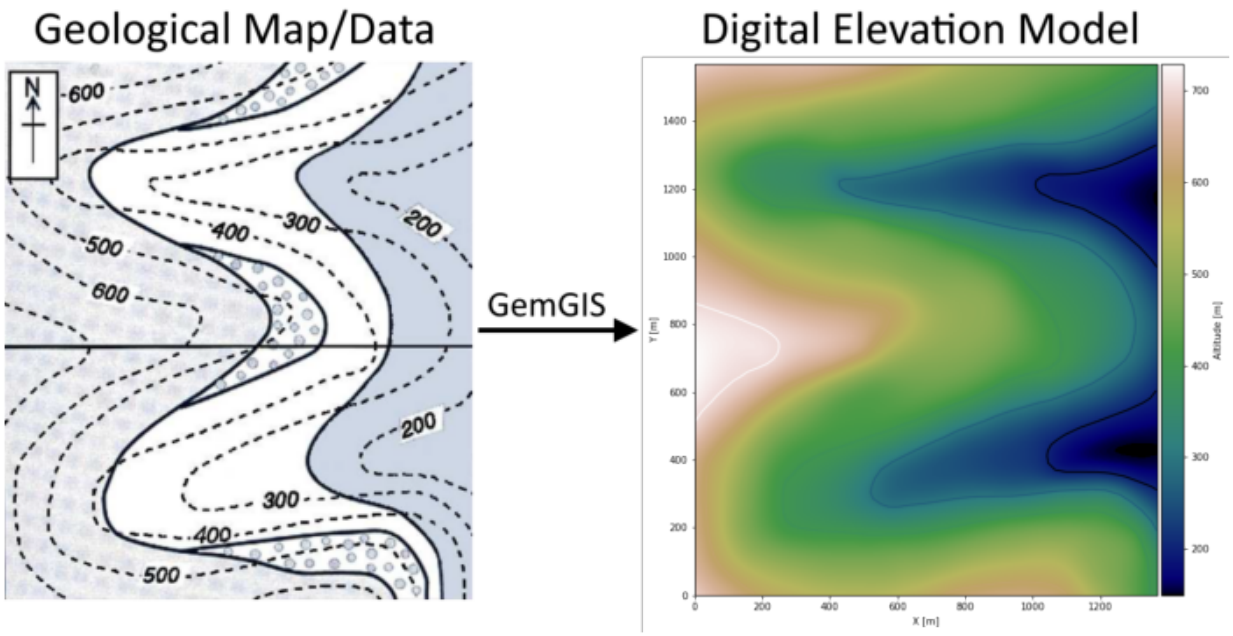
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example24/'
gg.download_gemgis_data.download_tutorial_data(filename="example24_unconformable_layers.
↪zip", dirpath=file_path)
```

7.24.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example24.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo24.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	600	LINESTRING (168.911 1471.213, 255.690 1480.424...
1	None	500	LINESTRING (174.728 1371.344, 230.965 1389.767...
2	None	400	LINESTRING (167.941 1287.474, 213.027 1305.412...
3	None	200	LINESTRING (1365.397 1329.167, 1332.915 1313.6...
4	None	700	LINESTRING (4.079 860.850, 49.165 845.821, 86...

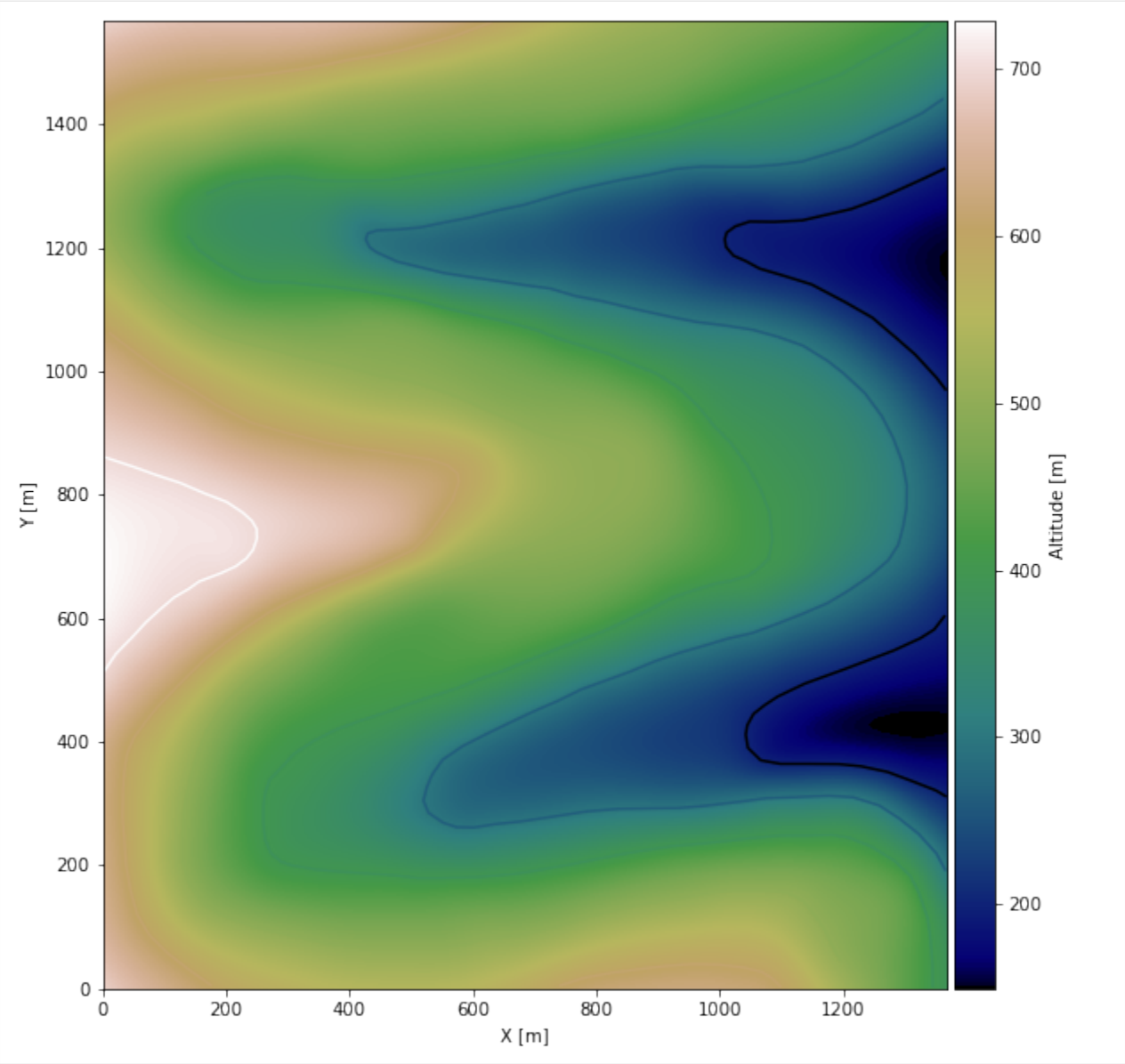
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10,10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0,1368,0,1568], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

[8]:
```
Text(95.40145646110935, 0.5, 'Y [m]')
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as as raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster24.tif', extent = [0, 1368, 0, 1568], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

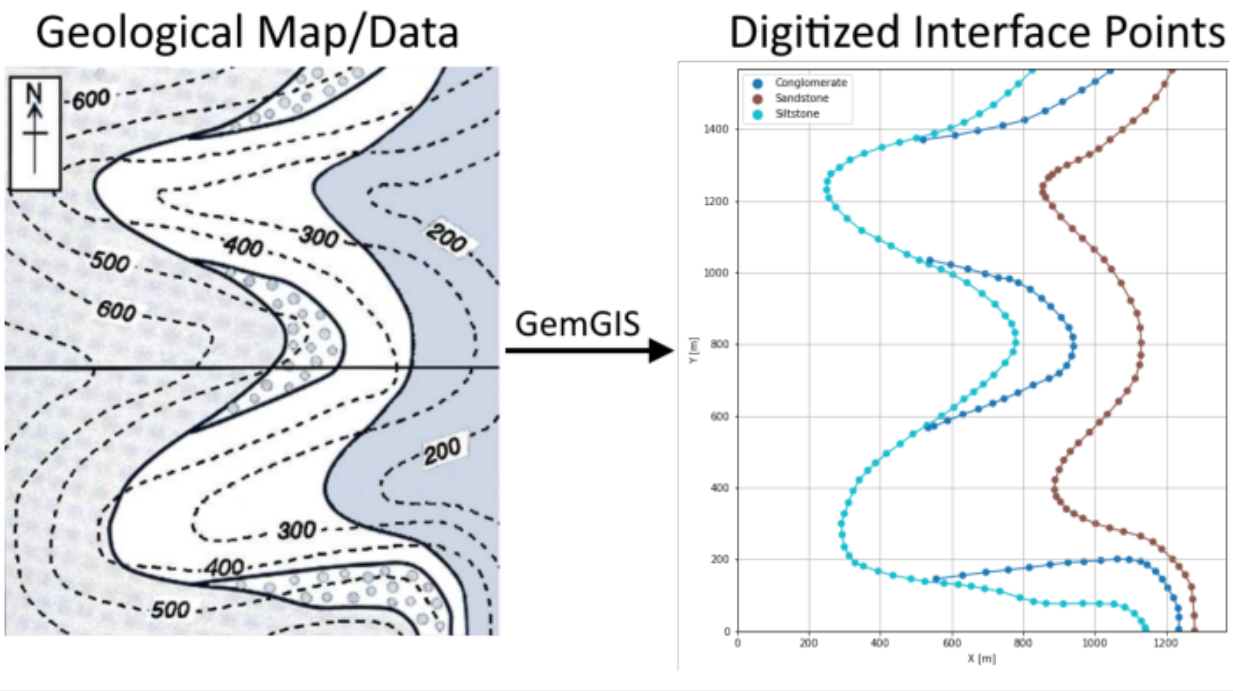
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster24.tif')
```

7.24.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example24.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces24.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Conglomerate	LINESTRING (520.148 1370.617, 609.351 1383.222...
1	None	Conglomerate	LINESTRING (539.055 1034.651, 597.716 1022.531...
2	None	Conglomerate	LINESTRING (556.508 146.013, 630.683 155.709, ...
3	None	Siltstone	LINESTRING (825.572 1565.022, 786.303 1526.723...
4	None	Sandstone	LINESTRING (1216.321 1565.507, 1195.959 1526.2...

Extracting XY coordinates from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

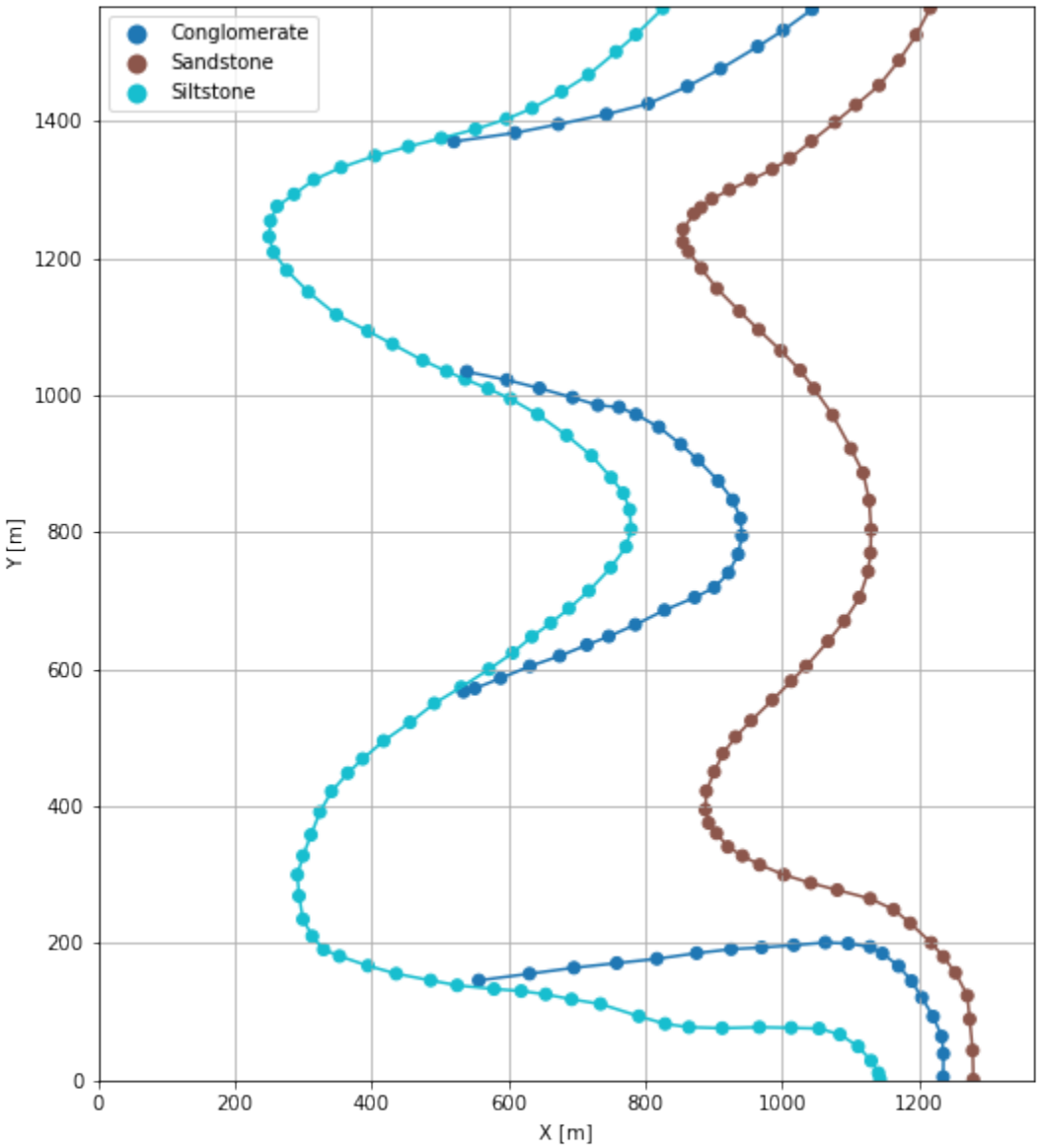
[12]:

	formation	geometry	X	Y	Z
99	Siltstone	POINT (571.537 600.270)	571.54	600.27	439.67
94	Siltstone	POINT (717.462 714.683)	717.46	714.68	494.31
88	Siltstone	POINT (749.943 880.000)	749.94	880.00	503.49
89	Siltstone	POINT (767.881 857.214)	767.88	857.21	504.65
90	Siltstone	POINT (777.092 832.974)	777.09	832.97	504.41

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0,1368)
plt.ylim(0,1568)
```

[13]:
```
(0.0, 1568.0)
```



7.24.6 Orientations from Strike Lines

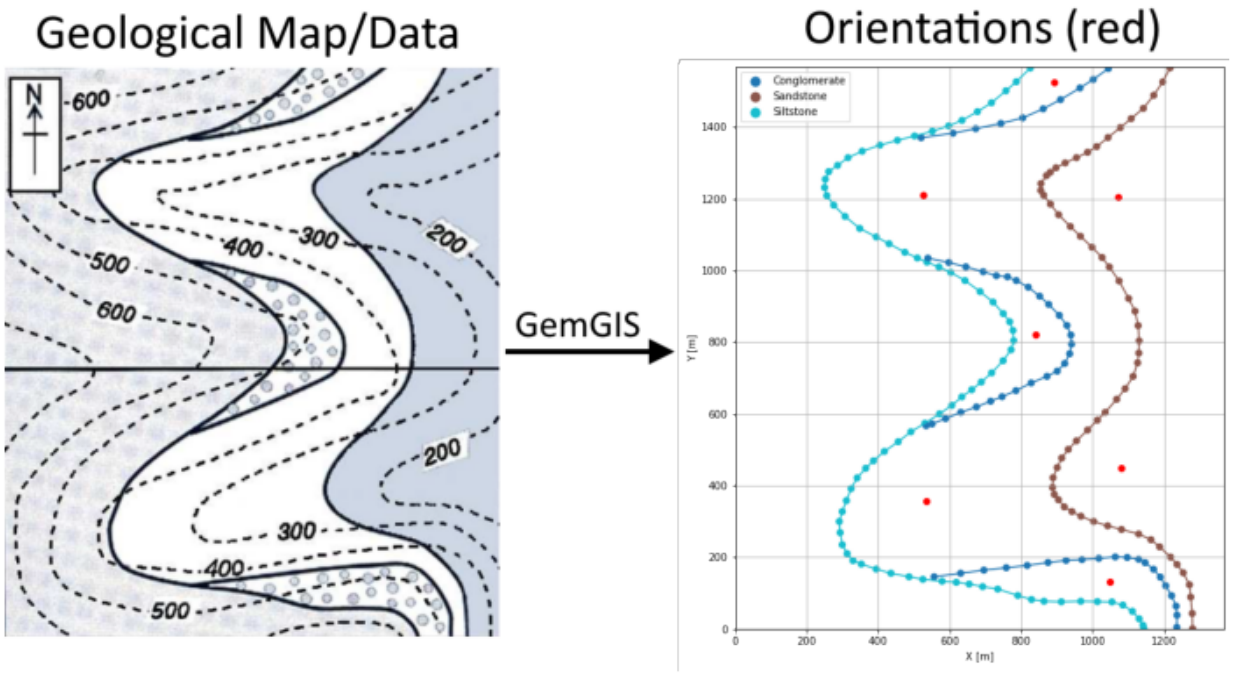
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/orientations_example24.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes24.shp')
strikes
```

[15]:

	id	formation	Z	geometry
0	1	Sandstone2	300	LINESTRING (998.646 566.334, 1000.100 300.179)
1	1	Sandstone1	300	LINESTRING (987.980 1332.803, 987.980 1074.889)
2	2	Sandstone2	400	LINESTRING (1161.539 249.275, 1158.630 686.564)
3	2	Sandstone1	400	LINESTRING (1150.873 1463.699, 1154.267 953.689)
4	1	Siltstone2	400	LINESTRING (313.623 1314.380, 314.593 1144.216)
5	2	Siltstone2	500	LINESTRING (737.339 894.059, 742.671 1489.878)
6	1	Siltstone1	400	LINESTRING (325.743 396.169, 328.167 190.129)
7	2	Siltstone1	500	LINESTRING (736.369 737.468, 747.035 104.805)
8	2	Conglomerate2	500	LINESTRING (875.022 883.878, 875.022 756.860)
9	1	Conglomerate2	500	LINESTRING (803.271 886.787, 803.271 757.830)
10	2	Conglomerate3	500	LINESTRING (937.076 1562.113, 936.591 1510.240)
11	1	Conglomerate3	500	LINESTRING (848.843 1559.689, 848.843 1467.092)
12	1	Conglomerate1	500	LINESTRING (1089.303 175.585, 1088.819 81.534)
13	2	Conglomerate1	500	LINESTRING (1006.403 177.040, 1007.372 89.291)

Calculate Orientations for each formation

[16]:
```
orientations_sand1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Sandstone1'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_sand1
```

[16]:
```
    dip  azimuth      Z                  geometry  polarity   formation  \
0 31.41   269.70 350.00  POINT (1070.275 1206.270)      1.00  Sandstone1

        X       Y
0 1070.27 1206.27
```

[17]:
```
orientations_sand2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Sandstone2'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_sand2
```

[17]:
```
    dip  azimuth      Z                  geometry  polarity   formation  \
0 31.88   269.64 350.00  POINT (1079.729 450.588)      1.00  Sandstone2

        X      Y
0 1079.73 450.59
```

[18]:
```
orientations_silt1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Siltstone1'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_silt1
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity   formation      X  \
0 13.51   269.06 450.00  POINT (534.329 357.143)      1.00  Siltstone1 534.33

       Y
0 357.14
```

[19]:
```
orientations_silt2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Siltstone2'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_silt2
```

[19]:
```
    dip  azimuth      Z                  geometry  polarity   formation  \
0 13.24   270.45 450.00  POINT (527.057 1210.633)      1.00  Siltstone2

       X       Y
0 527.06 1210.63
```

[20]:
```
orientations_conglomerate1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Conglomerate1'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_conglomerate1
```

[20]:
```
   dip  azimuth      Z                 geometry  polarity      formation  \
0 0.00     0.00 500.00  POINT (1047.974 130.863)      1.00  Conglomerate1
```

(continues on next page)

(continued from previous page)

```
        X      Y
0 1047.97 130.86
```

[21]:
```
orientations_conglomerate2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Conglomerate2'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_conglomerate2
```

[21]:
```
   dip  azimuth      Z                 geometry  polarity      formation  \
0 0.00   270.00 500.00  POINT (839.147 821.339)      1.00  Conglomerate2

       X      Y
0 839.15 821.34
```

[22]:
```
orientations_conglomerate3 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Conglomerate3'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_conglomerate3
```

[22]:
```
   dip  azimuth      Z                  geometry  polarity      formation  \
0 0.00    90.54 500.00  POINT (892.838 1524.784)      1.00  Conglomerate3

       X       Y
0 892.84 1524.78
```

Merging Orientations

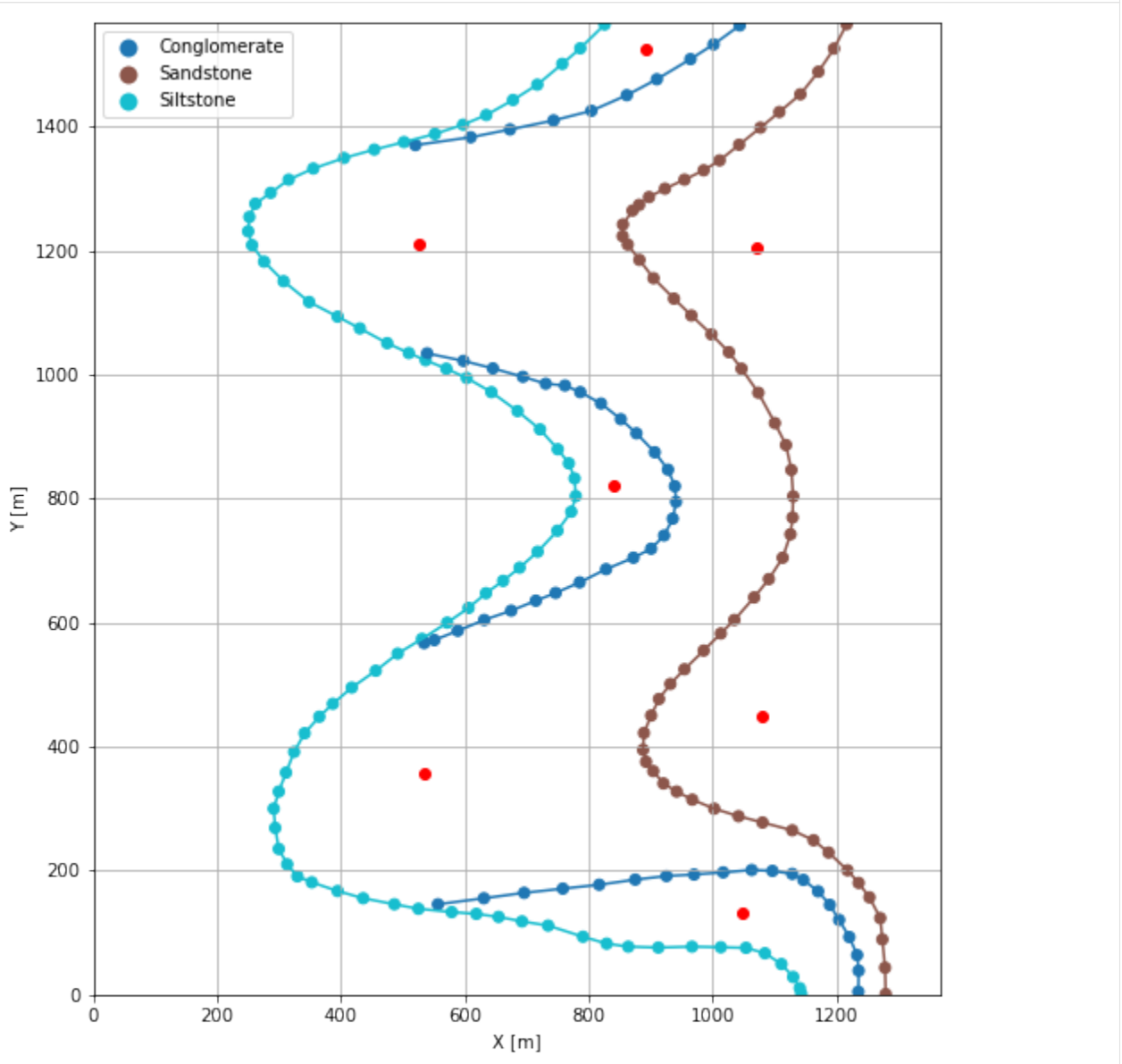
[23]:
```
import pandas as pd
orientations = pd.concat([orientations_sand1, orientations_sand2, orientations_silt1,␣
↪orientations_silt2, orientations_conglomerate1, orientations_conglomerate2,␣
↪orientations_conglomerate3]).reset_index()
orientations['formation'] = ['Sandstone', 'Sandstone', 'Siltstone', 'Siltstone',
↪'Conglomerate', 'Conglomerate', 'Conglomerate']
orientations
```

[23]:
```
   index   dip  azimuth      Z                   geometry  polarity  \
0      0 31.41   269.70 350.00  POINT (1070.275 1206.270)      1.00
1      0 31.88   269.64 350.00   POINT (1079.729 450.588)      1.00
2      0 13.51   269.06 450.00    POINT (534.329 357.143)      1.00
3      0 13.24   270.45 450.00   POINT (527.057 1210.633)      1.00
4      0  0.00     0.00 500.00   POINT (1047.974 130.863)      1.00
5      0  0.00   270.00 500.00    POINT (839.147 821.339)      1.00
6      0  0.00    90.54 500.00   POINT (892.838 1524.784)      1.00

      formation       X       Y
0     Sandstone 1070.27 1206.27
1     Sandstone 1079.73  450.59
2     Siltstone  534.33  357.14
3     Siltstone  527.06 1210.63
4  Conglomerate 1047.97  130.86
5  Conglomerate  839.15  821.34
6  Conglomerate  892.84 1524.78
```

Plotting the Orientations

[24]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0,1368)
plt.ylim(0,1568)
```

[24]:
```
(0.0, 1568.0)
```



7.24.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[25]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[26]:
```
geo_model = gp.create_model('Model24')
geo_model
```

[26]:
```
Model24  2022-04-08 08:28
```

Initiate Data

[27]:
```
gp.init_data(geo_model, [0,1368,0,1568,0,800], [100,100,100],
             surface_points_df = interfaces_coords[interfaces_coords['Z']!=0],
             orientations_df = orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[27]:
```
Model24  2022-04-08 08:28
```

Model Surfaces

[28]:
```
geo_model.surfaces
```

[28]:

	surface	series	order_surfaces	color	id
0	Siltstone	Default series	1	#015482	1
1	Sandstone	Default series	2	#9f0052	2
2	Conglomerate	Default series	3	#ffbe00	3

Mapping the Stack to Surfaces

[29]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Siltstone'),
                          'Strata2': ('Conglomerate'),
                          'Strata3': ('Sandstone'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Claystone')
```

[29]:
```
        surface   series  order_surfaces    color  id
0     Siltstone  Strata1               1  #015482   1
2  Conglomerate  Strata2               1  #9f0052   2
1     Sandstone  Strata3               1  #ffbe00   3
3     Claystone  Strata3               2  #728f02   4
```

Adding additional Orientations

[30]:
```
geo_model.add_orientations(X=600, Y=50, Z=400, surface='Conglomerate', orientation = [0,
↪0,1])
geo_model.add_orientations(X=150, Y=50, Z=400, surface='Conglomerate', orientation = [0,
↪0,1])
geo_model.add_orientations(X=600, Y=275, Z=400, surface='Conglomerate', orientation = [0,
↪0,1])
geo_model.add_orientations(X=150, Y=275, Z=400, surface='Conglomerate', orientation = [0,
↪0,1])
geo_model.add_orientations(X=600, Y=850, Z=400, surface='Conglomerate', orientation = [0,
↪0,1])
geo_model.add_orientations(X=150, Y=850, Z=400, surface='Conglomerate', orientation = [0,
↪0,1])
geo_model.add_orientations(X=600, Y=1250, Z=400, surface='Conglomerate', orientation =␣
↪[0,0,1])
geo_model.add_orientations(X=150, Y=1250, Z=400, surface='Conglomerate', orientation =␣
↪[0,0,1])
geo_model.add_orientations(X=600, Y=1500, Z=400, surface='Conglomerate', orientation =␣
↪[0,0,1])
geo_model.add_orientations(X=150, Y=1500, Z=400, surface='Conglomerate', orientation =␣
↪[0,0,1])

geo_model.add_orientations(X=600, Y=50, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=150, Y=50, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=600, Y=275, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=150, Y=275, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=600, Y=850, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=150, Y=850, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=600, Y=1250, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=150, Y=1250, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=600, Y=1500, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])
geo_model.add_orientations(X=150, Y=1500, Z=500, surface='Siltstone', orientation = [270,
↪13.5,1])

geo_model.add_orientations(X=600, Y=50, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
```

(continues on next page)

(continued from previous page)

```
geo_model.add_orientations(X=150, Y=50, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=600, Y=275, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=150, Y=275, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=600, Y=850, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=150, Y=850, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=600, Y=1250, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=150, Y=1250, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=600, Y=1500, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
geo_model.add_orientations(X=150, Y=1500, Z=300, surface='Sandstone', orientation = [270,
↪31.5,1])
```

[30]:

	X	Y	Z	G_x	G_y	G_z	smooth	surface
2	534.33	357.14	450.00	-0.23	-0.00	0.97	0.01	Siltstone
3	527.06	1210.63	450.00	-0.23	0.00	0.97	0.01	Siltstone
17	600.00	50.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
18	150.00	50.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
19	600.00	275.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
20	150.00	275.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
21	600.00	850.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
22	150.00	850.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
23	600.00	1250.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
24	150.00	1250.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
25	600.00	1500.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
26	150.00	1500.00	500.00	-0.23	0.00	0.97	0.01	Siltstone
4	1047.97	130.86	500.00	0.00	0.00	1.00	0.01	Conglomerate
5	839.15	821.34	500.00	0.00	0.00	1.00	0.01	Conglomerate
6	892.84	1524.78	500.00	0.00	0.00	1.00	0.01	Conglomerate
7	600.00	50.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
8	150.00	50.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
9	600.00	275.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
10	150.00	275.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
11	600.00	850.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
12	150.00	850.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
13	600.00	1250.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
14	150.00	1250.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
15	600.00	1500.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
16	150.00	1500.00	400.00	0.00	0.00	1.00	0.01	Conglomerate
0	1070.27	1206.27	350.00	-0.52	-0.00	0.85	0.01	Sandstone
1	1079.73	450.59	350.00	-0.53	-0.00	0.85	0.01	Sandstone
27	600.00	50.00	300.00	-0.52	0.00	0.85	0.01	Sandstone
28	150.00	50.00	300.00	-0.52	0.00	0.85	0.01	Sandstone
29	600.00	275.00	300.00	-0.52	0.00	0.85	0.01	Sandstone
30	150.00	275.00	300.00	-0.52	0.00	0.85	0.01	Sandstone
31	600.00	850.00	300.00	-0.52	0.00	0.85	0.01	Sandstone

(continues on next page)

(continued from previous page)

```
32  150.00  850.00 300.00 -0.52  0.00 0.85    0.01     Sandstone
33  600.00 1250.00 300.00 -0.52  0.00 0.85    0.01     Sandstone
34  150.00 1250.00 300.00 -0.52  0.00 0.85    0.01     Sandstone
35  600.00 1500.00 300.00 -0.52  0.00 0.85    0.01     Sandstone
36  150.00 1500.00 300.00 -0.52  0.00 0.85    0.01     Sandstone
```

Showing the Number of Data Points

[31]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[31]:
```
        surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0     Siltstone  Strata1               1  #015482   1                 80               ␣
↪  12
2  Conglomerate  Strata2               1  #9f0052   2                 57               ␣
↪  13
1     Sandstone  Strata3               1  #ffbe00   3                 61               ␣
↪  12
3     Claystone  Strata3               2  #728f02   4                  0               ␣
↪   0
```

Loading Digital Elevation Model

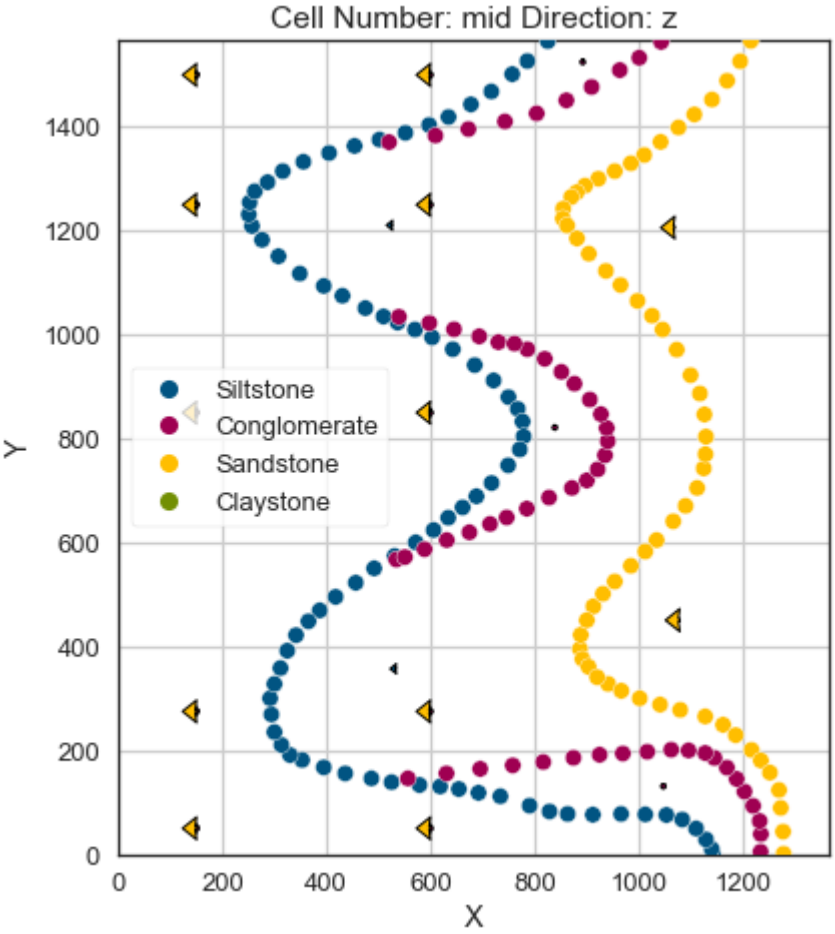
[32]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster24.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

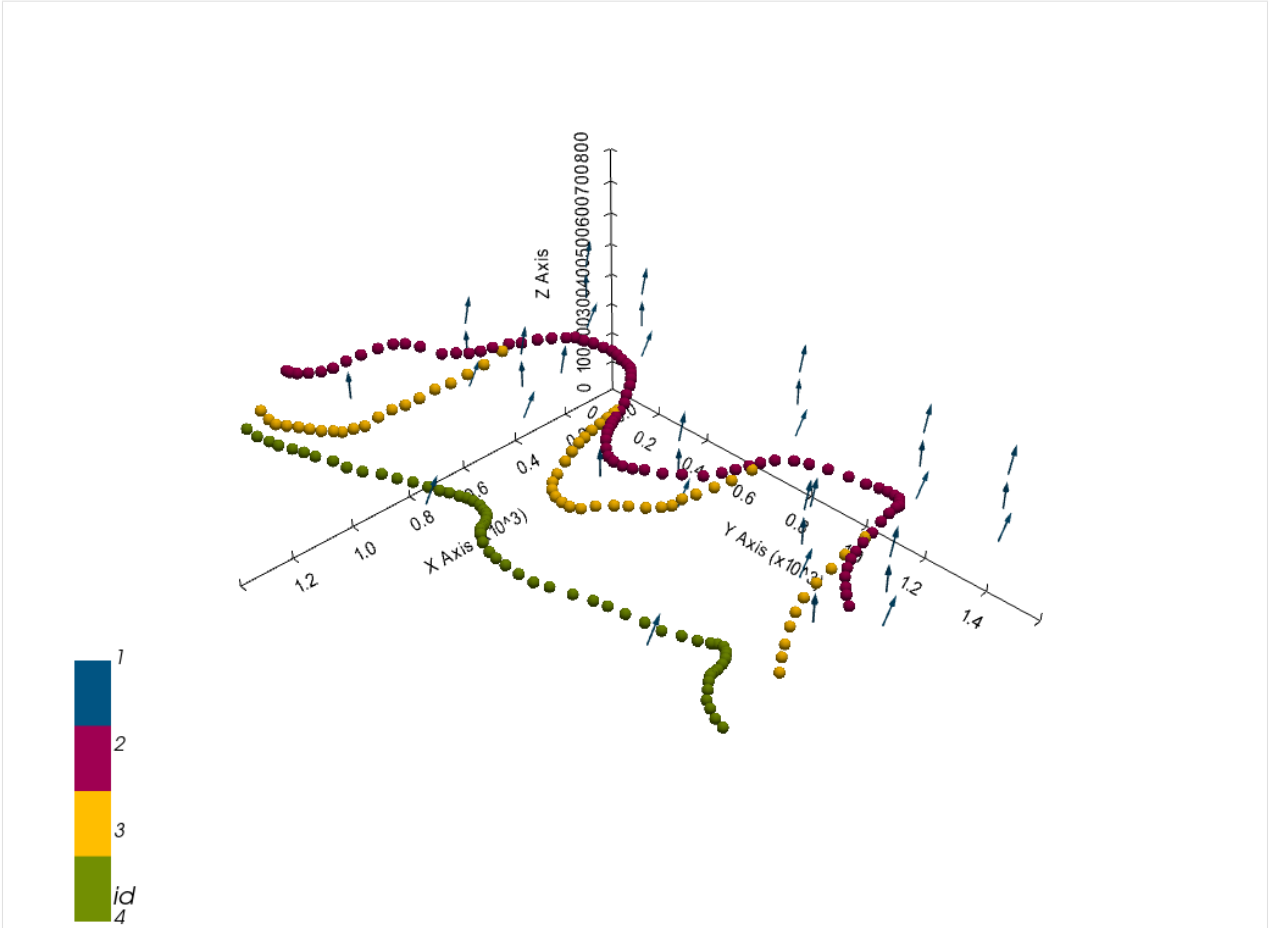
[32]:
```
Grid Object. Values:
array([[   6.84      ,    7.84      ,    4.        ],
       [   6.84      ,    7.84      ,   12.        ],
       [   6.84      ,    7.84      ,   20.        ],
       ...,
       [1365.49450549, 1555.47603834,  383.11444092],
       [1365.49450549, 1560.485623  ,  386.23483276],
       [1365.49450549, 1565.49520767,  389.29962158]])
```

Plotting Input Data

[33]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[34]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[34]:
```
<gempy.plot.vista.GemPyToVista at 0x20f8cfe3340>
```

Setting the Interpolator

[35]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                      values
range               2229.36
$C_o$             118334.48
drift equations   [3, 3, 3]
```

[35]:
```
<gempy.core.interpolator.InterpolatorModel at 0x20f86f139d0>
```

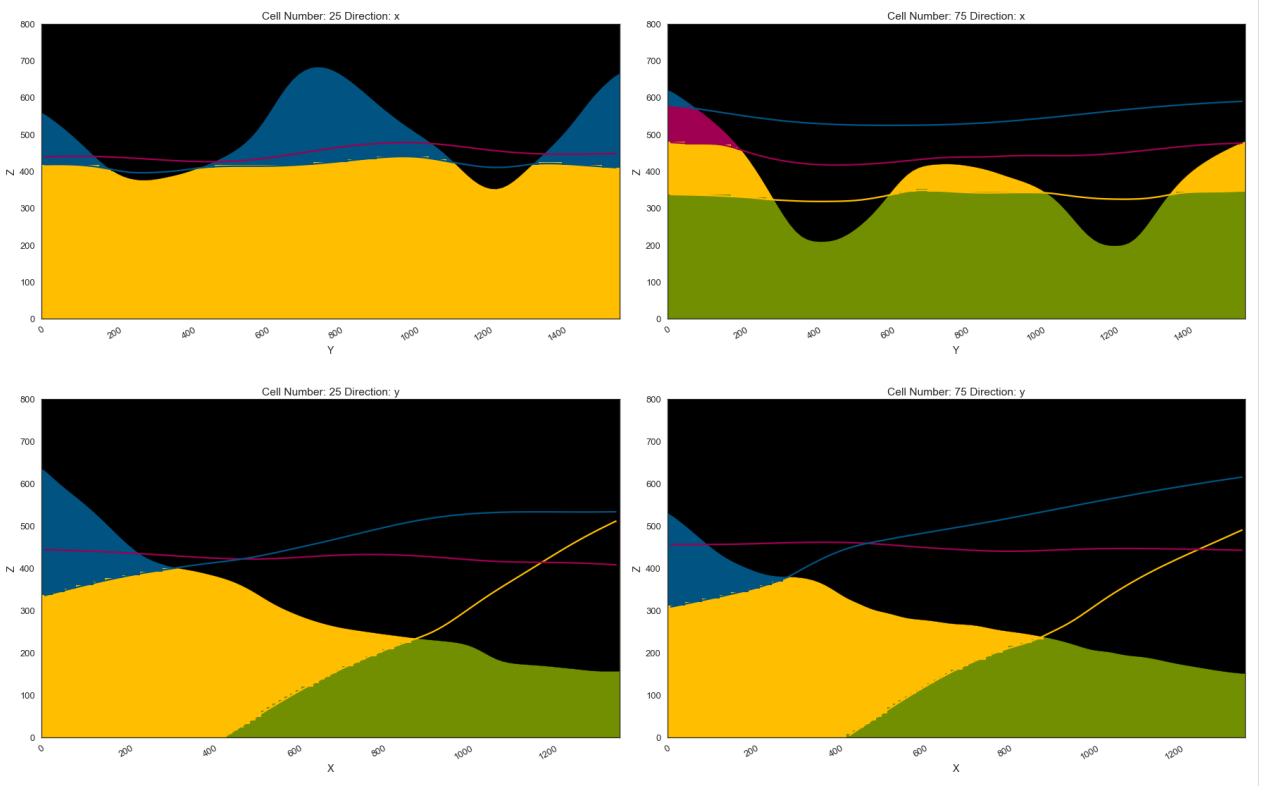
Computing Model

[36]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

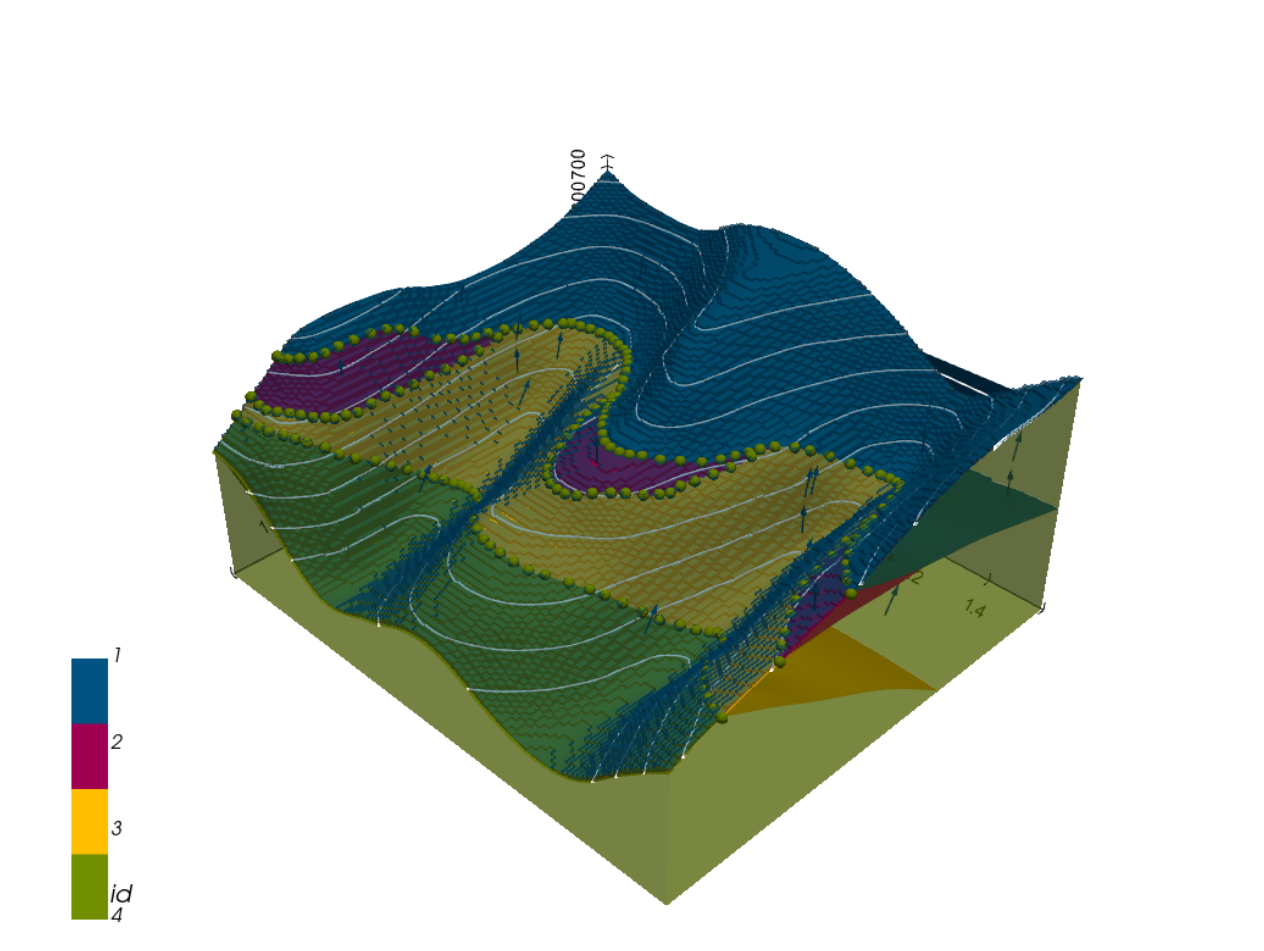
[37]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25,75,25,75], show_
↪topography=True, show_data=False)
```

[37]:
```
<gempy.plot.visualization_2d.Plot2D at 0x20f8b580220>
```



Plotting 3D Model

[38]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

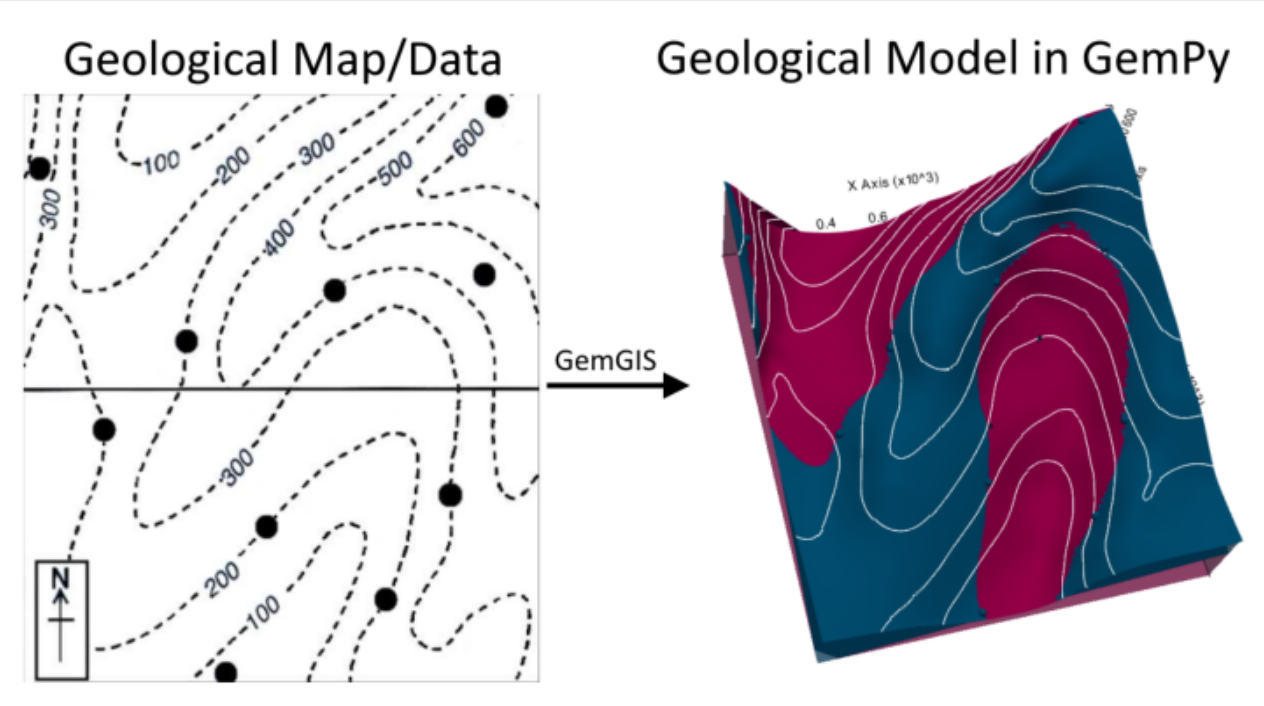
7.25 Example 25 - Planar Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 1370 m wide (W-E extent) and 1561 m high (N-S extent). The model represents planar layers dipping towards the southwest.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example25.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.25.1 Licensing

Computational Geosciences and Reservoir Engineering, RWTH Aachen University, Authors: Alexander Juestel. For more information contact: alexander.juestel(at)rwth-aachen.de

This work is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/)

7.25.2 Import GemGIS

If you have installed `GemGIS` via pip, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.25.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded form an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

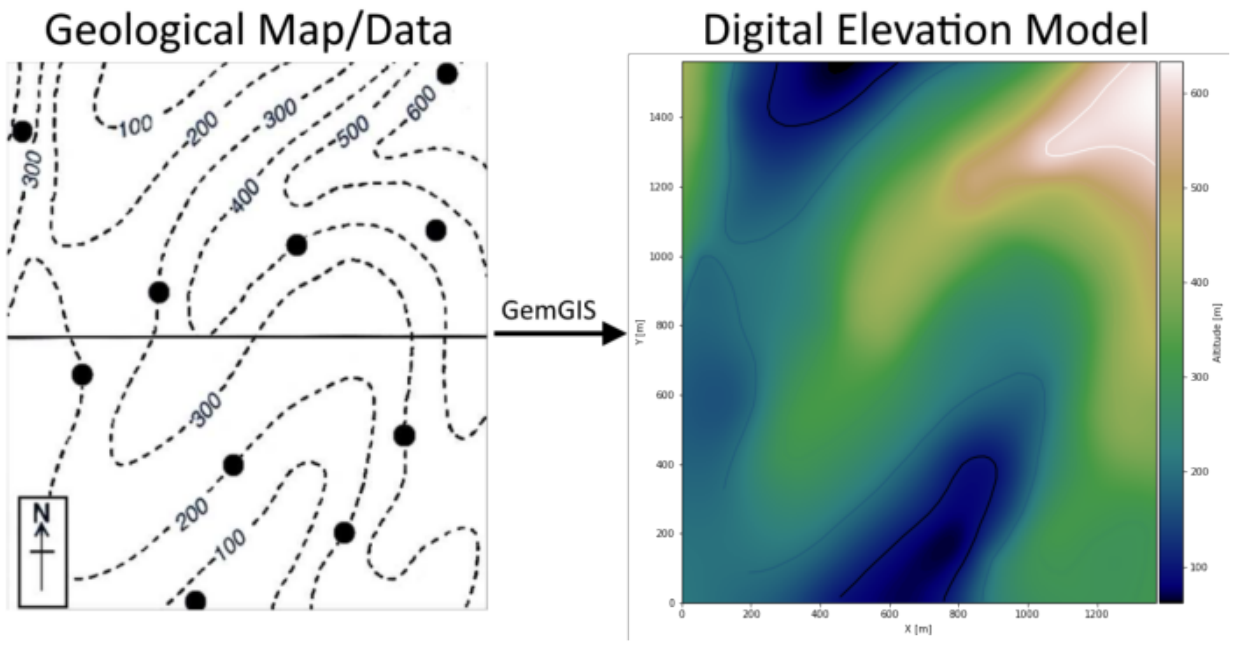
[4]:
```
file_path = 'data/example25/'
gg.download_gemgis_data.download_tutorial_data(filename="example25_planar_dipping_layers.
↪zip", dirpath=file_path)
```

```
Downloading file 'example25_planar_dipping_layers.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2Fexample25_planar_dipping_layers.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\data\example25'.
```

7.25.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
img = mpimg.imread('../images/dem_example25.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo25.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	100	LINESTRING (457.460 15.883, 503.059 59.552, 53...
1	None	200	LINESTRING (194.243 87.056, 230.432 90.433, 26...
2	None	200	LINESTRING (2.921 841.967, 15.225 879.122, 25...
3	None	400	LINESTRING (4.369 1340.899, 13.054 1365.026, 1...
4	None	300	LINESTRING (3.765 1021.105, 17.276 1062.120, 3...

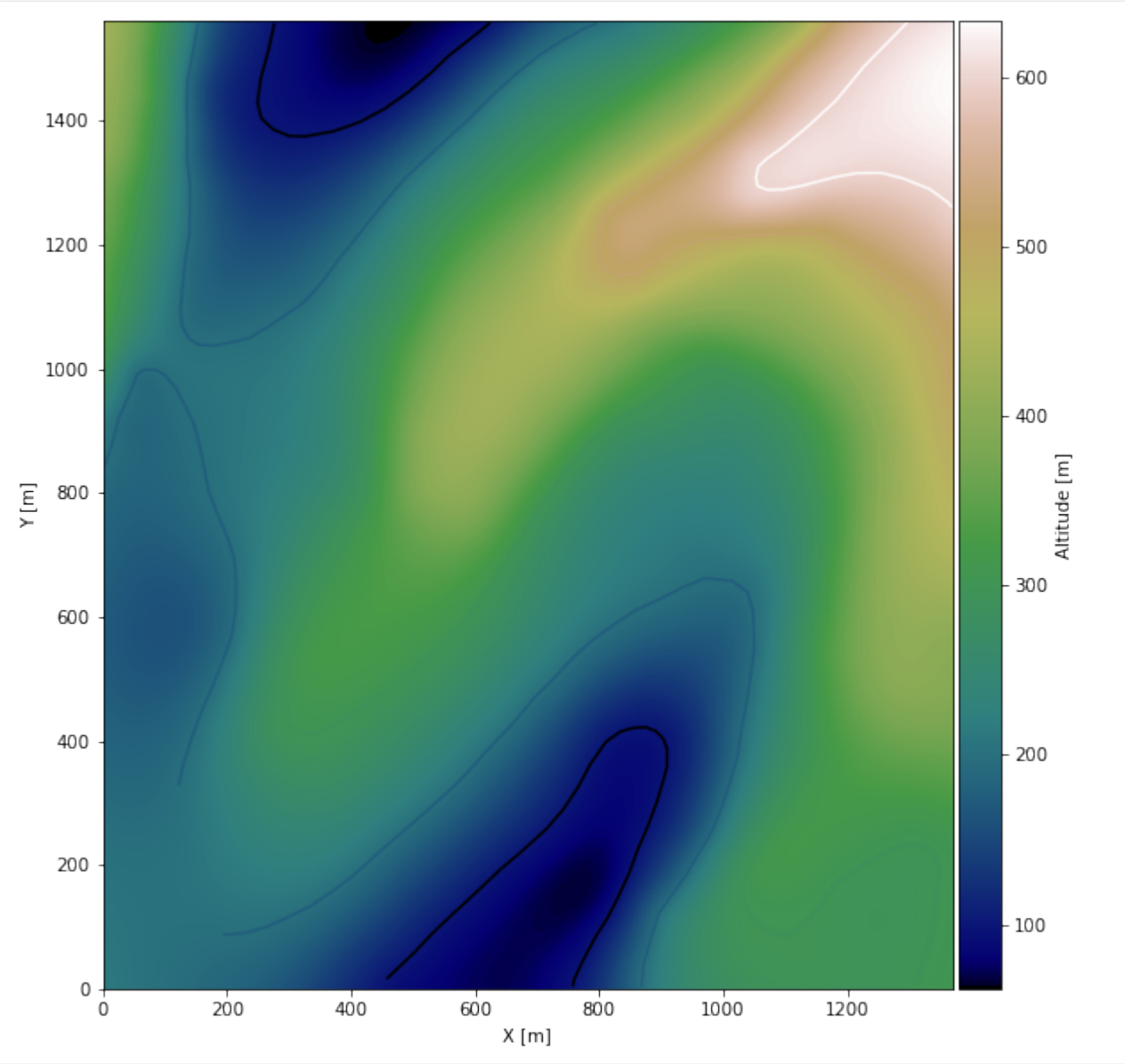
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10,10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0,1370,0,1561], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

[8]:
```
Text(94.03793584379343, 0.5, 'Y [m]')
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as as raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster25.tif', extent = [0, 1370, 0, 1561], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

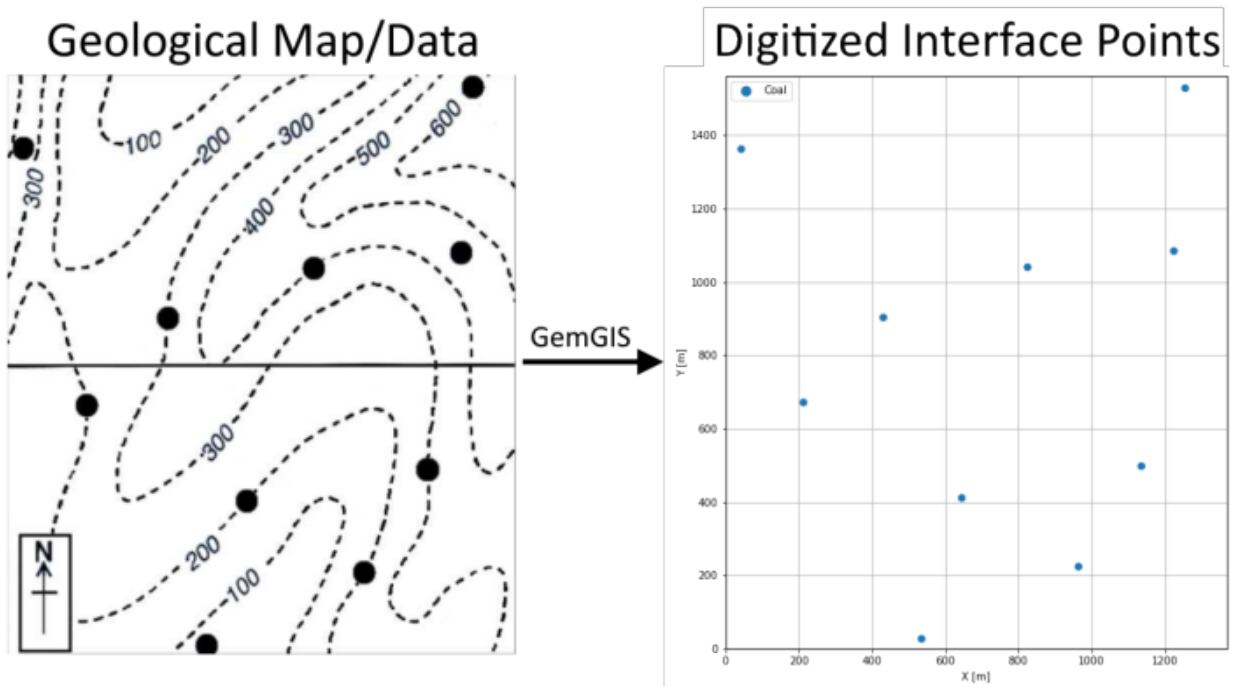
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster25.tif')
```

7.25.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../images/interfaces_example25.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces25.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Coal	POINT (535.509 26.619)
1	None	Coal	POINT (963.992 223.490)
2	None	Coal	POINT (1135.771 497.565)
3	None	Coal	POINT (645.525 410.710)
4	None	Coal	POINT (213.182 671.274)

Extracting XY coordinates from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

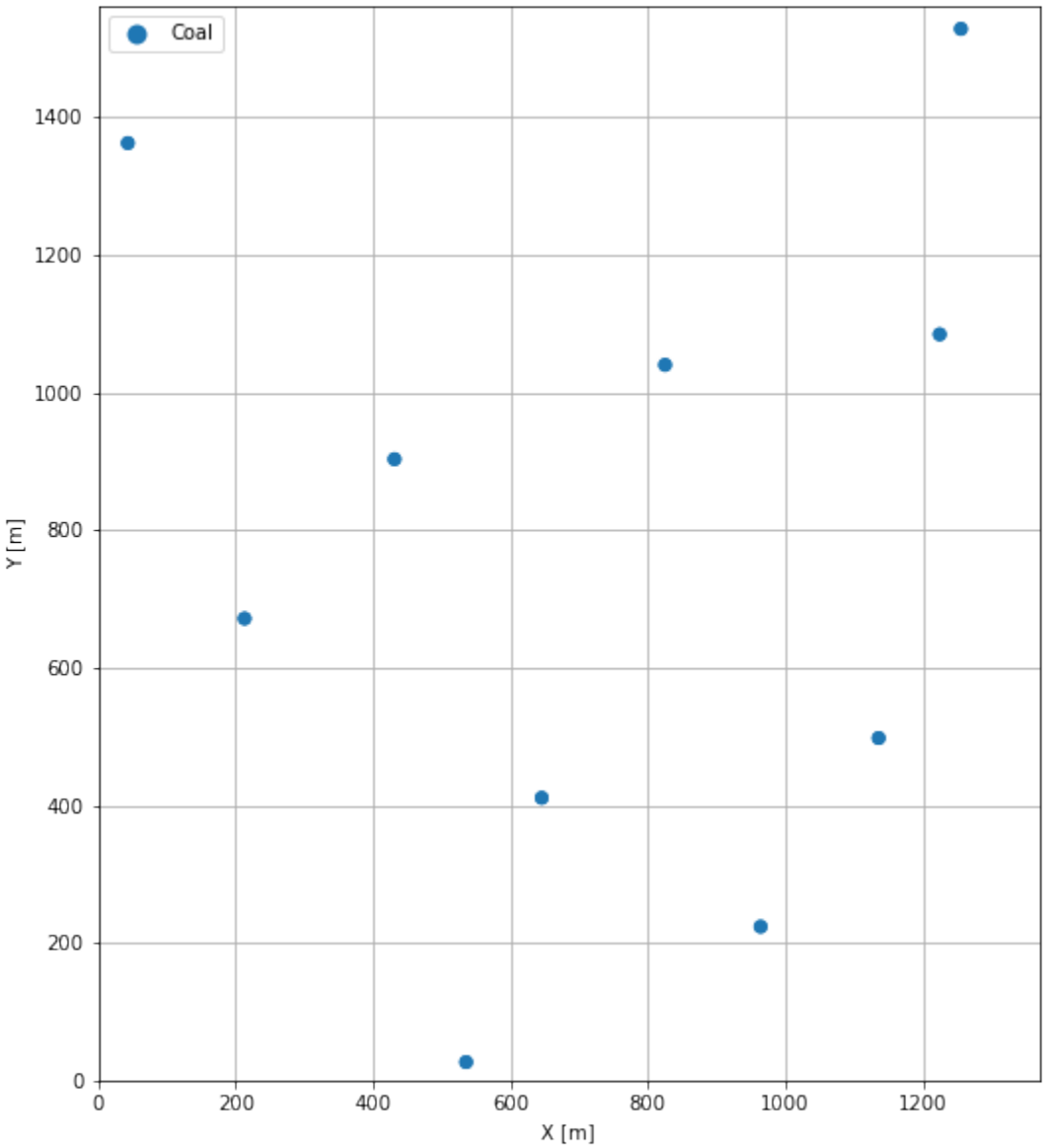
[12]:

	formation	geometry	X	Y	Z
0	Coal	POINT (535.509 26.619)	535.51	26.62	83.73
1	Coal	POINT (963.992 223.490)	963.99	223.49	198.26
2	Coal	POINT (1135.771 497.565)	1135.77	497.56	301.38
3	Coal	POINT (645.525 410.710)	645.52	410.71	199.98
4	Coal	POINT (213.182 671.274)	213.18	671.27	199.13

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0,1370)
plt.ylim(0,1561)
```

[13]:
```
(0.0, 1561.0)
```



7.25.6 Orientations from Strike Lines

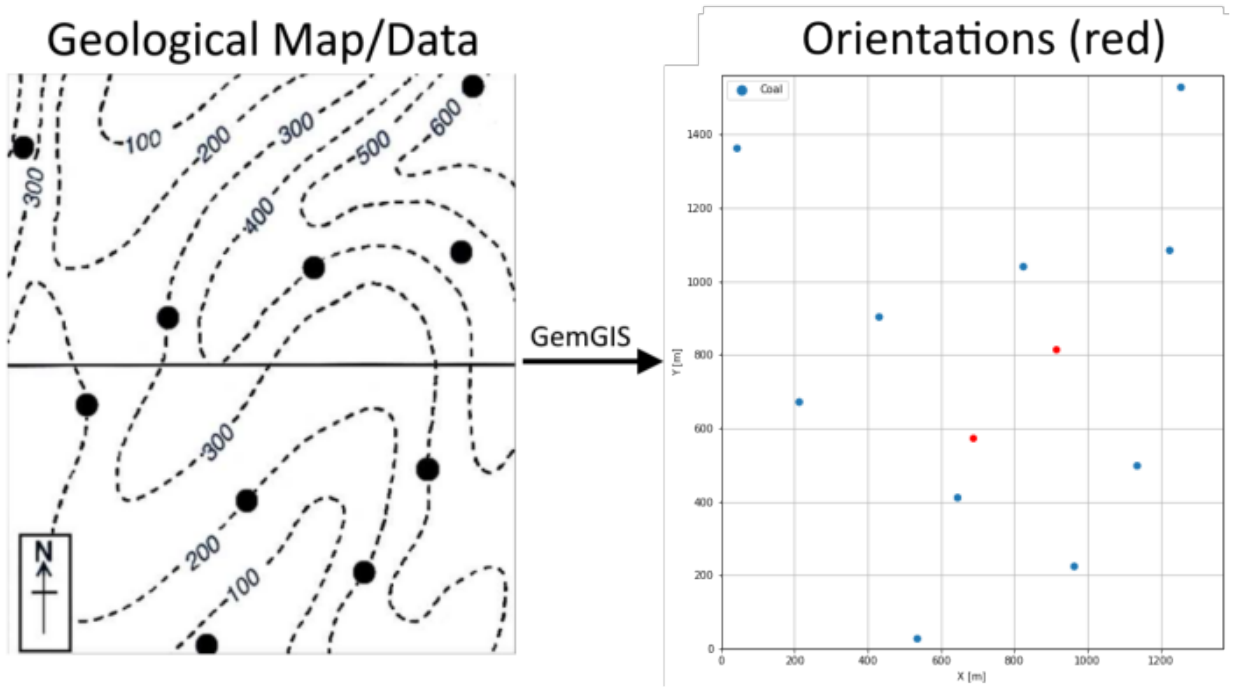
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
img = mpimg.imread('../images/orientations_example25.png')
plt.figure(figsize=(10, 10))
```

(continues on next page)

(continued from previous page)

```
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes25.shp')
strikes
```

[15]:
```
   id formation    Z                                        geometry
0   1      Coal  200    LINESTRING (214.147 670.791, 968.817 221.077)
1   2      Coal  300   LINESTRING (432.248 905.299, 1137.701 501.907)
2   3      Coal  400  LINESTRING (828.885 1042.336, 1251.577 804.934)
```

Calculate Orientations for each formation

[16]:
```
orientations_coal = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Coal'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_coal
```

[16]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 17.71   210.32 250.00  POINT (688.228 574.769)      1.00      Coal 688.23
1 17.57   209.65 350.00  POINT (912.603 813.619)      1.00      Coal 912.60

       Y
0 574.77
1 813.62
```

Merging Orientations

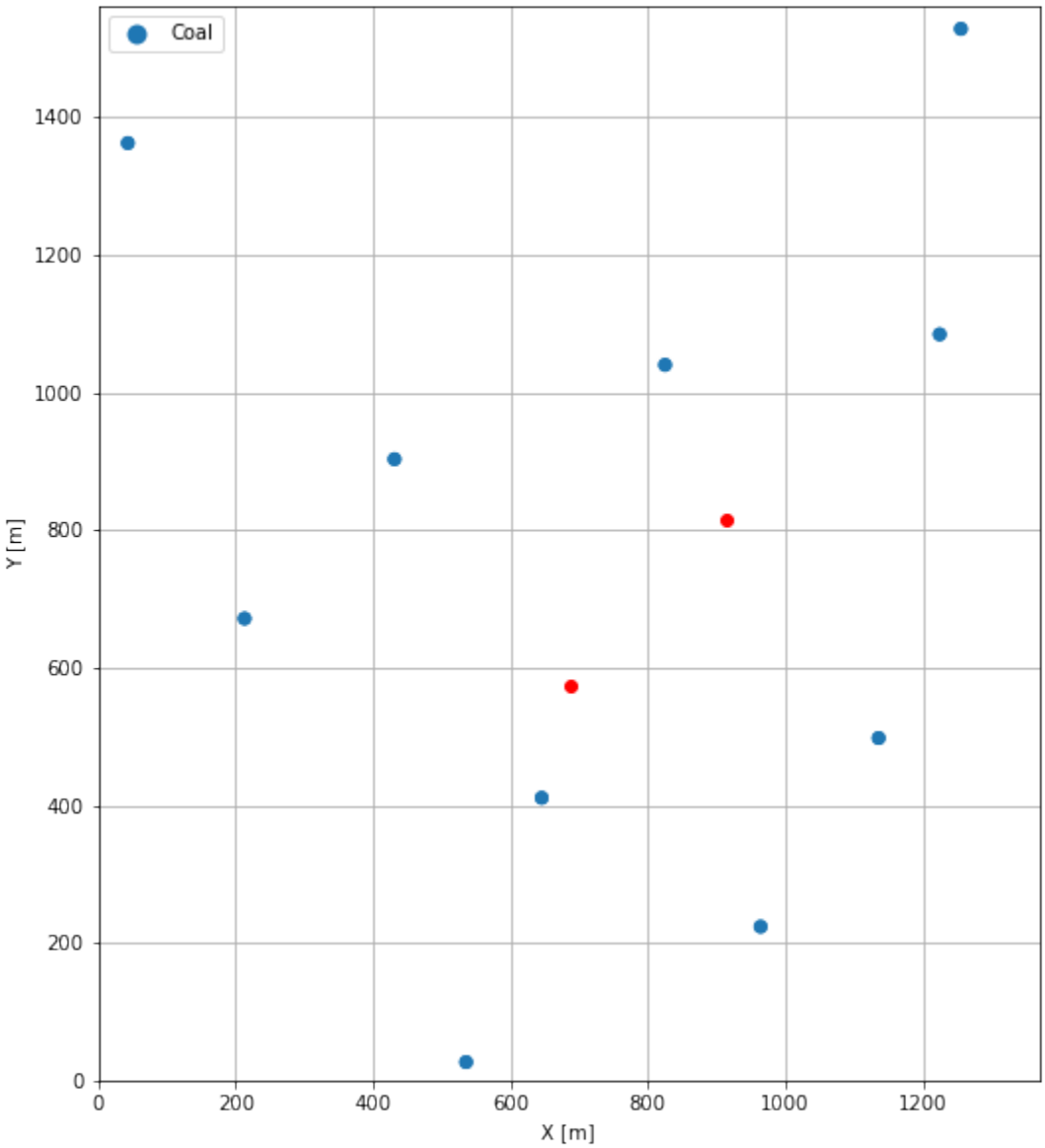
[17]:
```
import pandas as pd
orientations = pd.concat([orientations_coal]).reset_index()
orientations['formation'] = 'Coal'
orientations
```

[17]:
```
   index   dip  azimuth      Z                 geometry  polarity formation  \
0      0 17.71   210.32 250.00  POINT (688.228 574.769)      1.00      Coal
1      1 17.57   209.65 350.00  POINT (912.603 813.619)      1.00      Coal

       X      Y
0 688.23 574.77
1 912.60 813.62
```

Plotting the Orientations

[18]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0,1370)
plt.ylim(0,1561)
```

[18]:
```
(0.0, 1561.0)
```



7.25.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[19]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[20]:
```
geo_model = gp.create_model('Model25')
geo_model
```

[20]:
```
Model25  2022-04-09 09:01
```

Initiate Data

[21]:
```
gp.init_data(geo_model, [0,1370,0,1561,0,800], [100,100,100],
             surface_points_df = interfaces_coords[interfaces_coords['Z']!=0],
             orientations_df = orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[21]:
```
Model25  2022-04-09 09:01
```

Model Surfaces

[22]:
```
geo_model.surfaces
```

[22]:

	surface	series	order_surfaces	color	id
0	Coal	Default series	1	#015482	1

Mapping the Stack to Surfaces

[23]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Coal'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[23]:

	surface	series	order_surfaces	color	id
0	Coal	Strata1	1	#015482	1
1	Basement	Strata1	2	#9f0052	2

Showing the Number of Data Points

[24]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[24]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Coal	Strata1	1	#015482	1	10	2
1	Basement	Strata1	2	#9f0052	2	0	0

Loading Digital Elevation Model

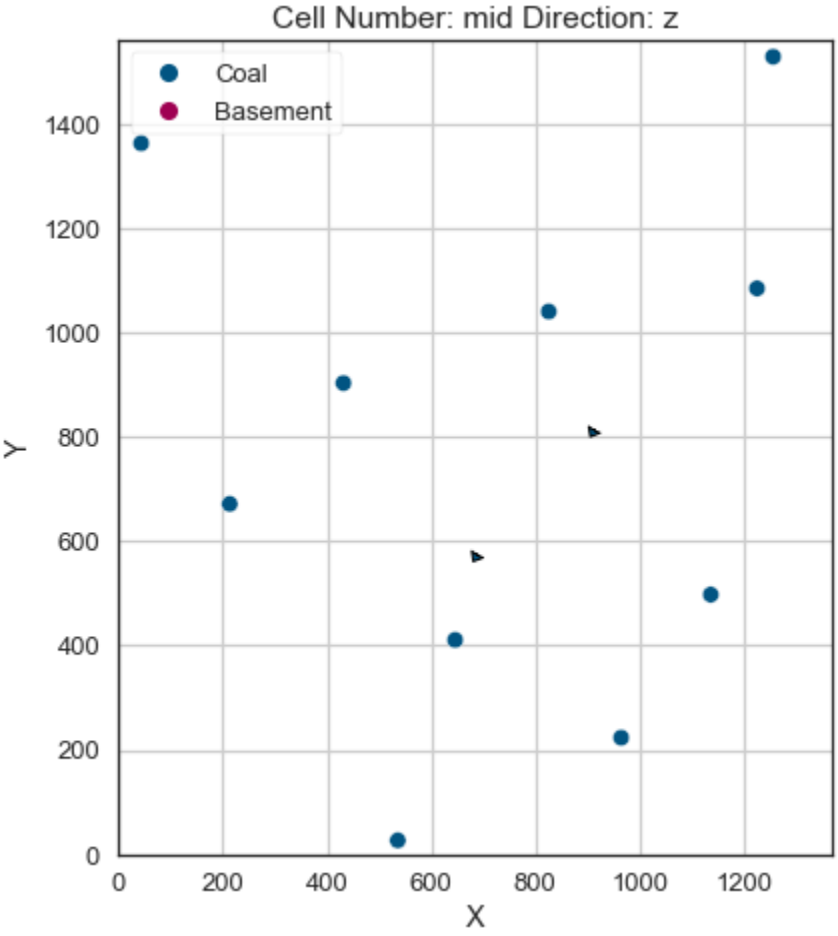
[25]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster25.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

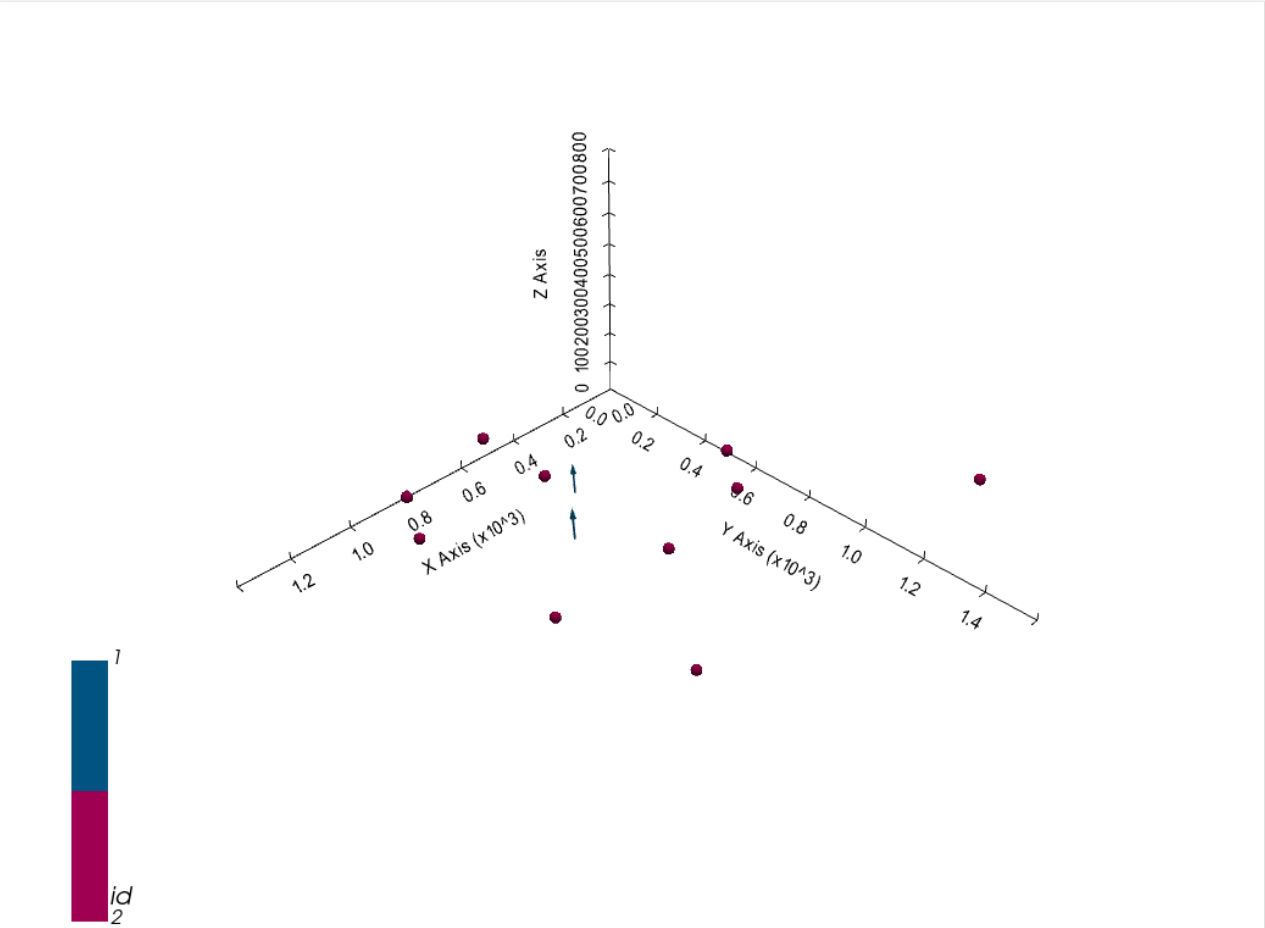
[25]:
```
Grid Object. Values:
array([[   6.85      ,    7.805     ,    4.        ],
       [   6.85      ,    7.805     ,   12.        ],
       [   6.85      ,    7.805     ,   20.        ],
       ...,
       [1367.5       , 1548.49198718,  622.15332031],
       [1367.5       , 1553.49519231,  620.90863037],
       [1367.5       , 1558.49839744,  619.59674072]])
```

Plotting Input Data

[26]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[27]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[27]:
```
<gempy.plot.vista.GemPyToVista at 0x2582e21b460>
```

Setting the Interpolator

[28]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                    values
range              2225.67
$C_o$            117943.36
drift equations        [3]
```

[28]:
```
<gempy.core.interpolator.InterpolatorModel at 0x258274bbdc0>
```

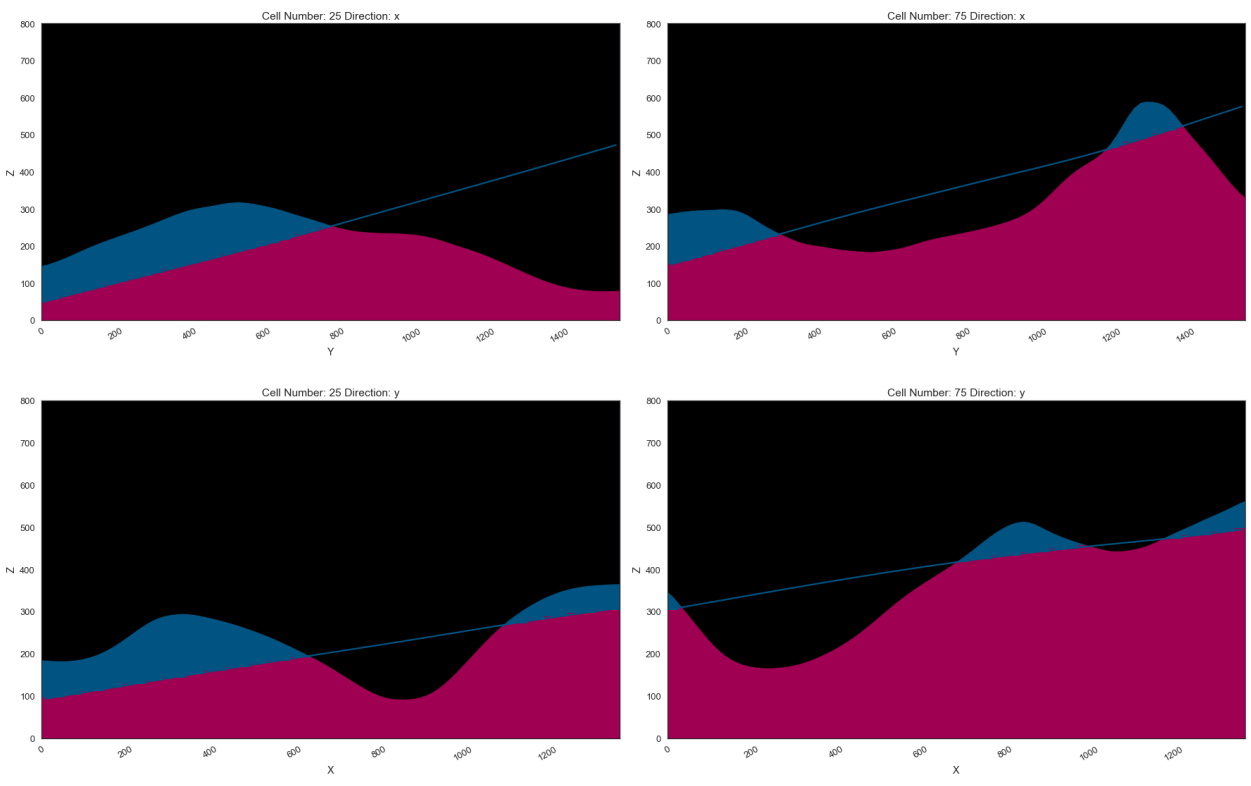
Computing Model

[29]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

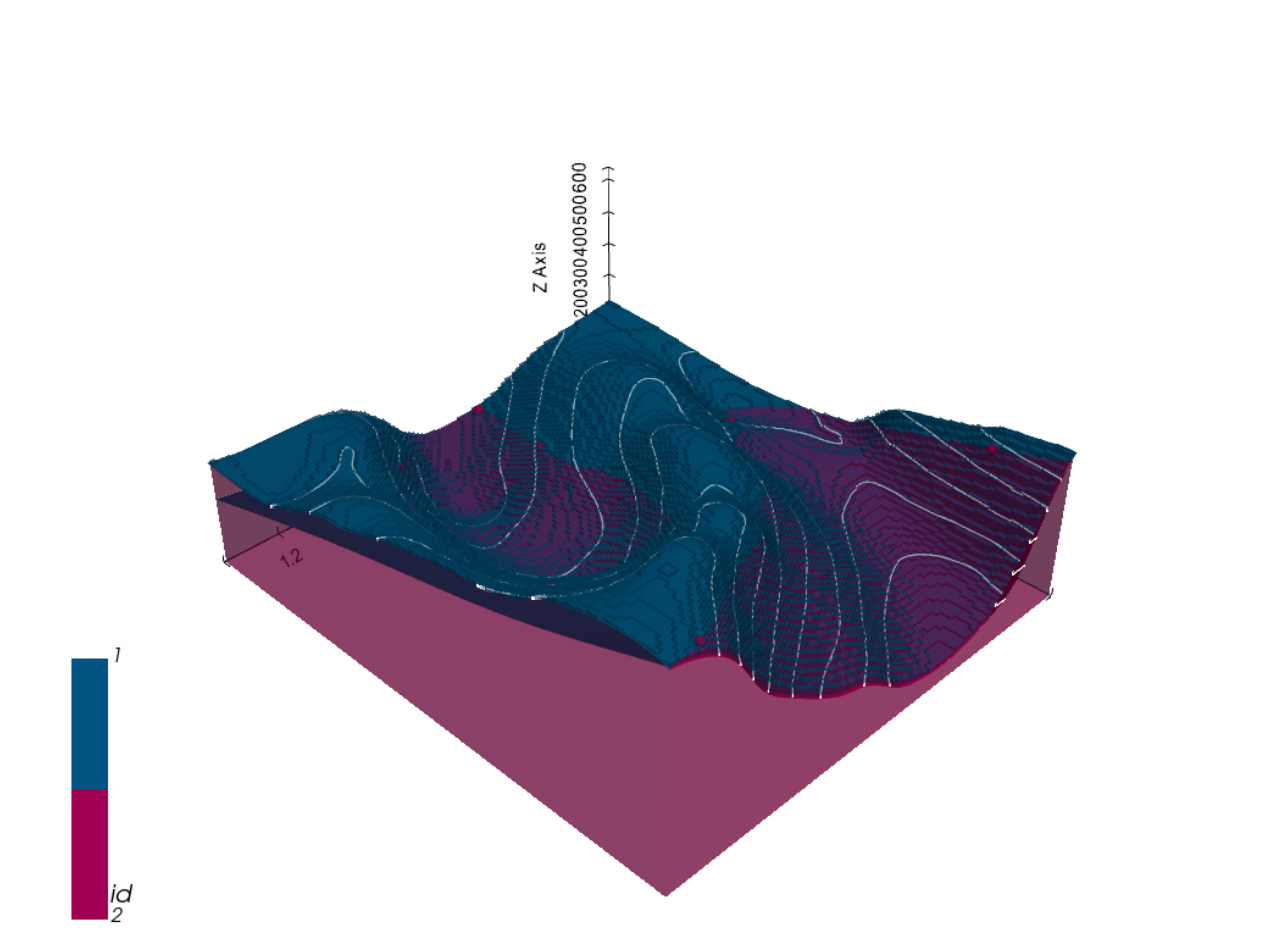
[30]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25,75,25,75], show_
↪topography=True, show_data=False)
```

[30]:
```
<gempy.plot.visualization_2d.Plot2D at 0x25834904490>
```



Plotting 3D Model

[31]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

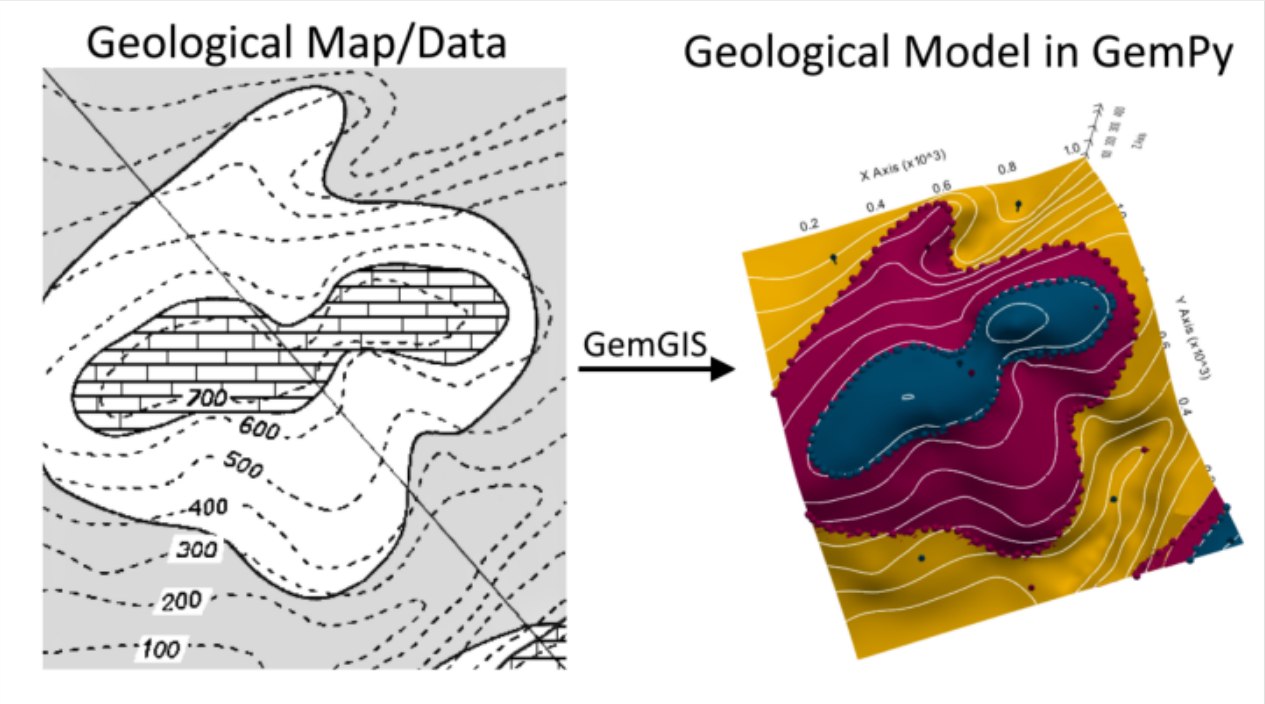
7.26 Example 26 - Unconformable Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 1022 m wide (W-E extent) and 1172 m high (N-S extent). The model represents unconformable folded layers.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example26.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.26.1 Licensing

7.26.2 Import GemGIS

If you have installed `GemGIS` via pip, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.26.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded form an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

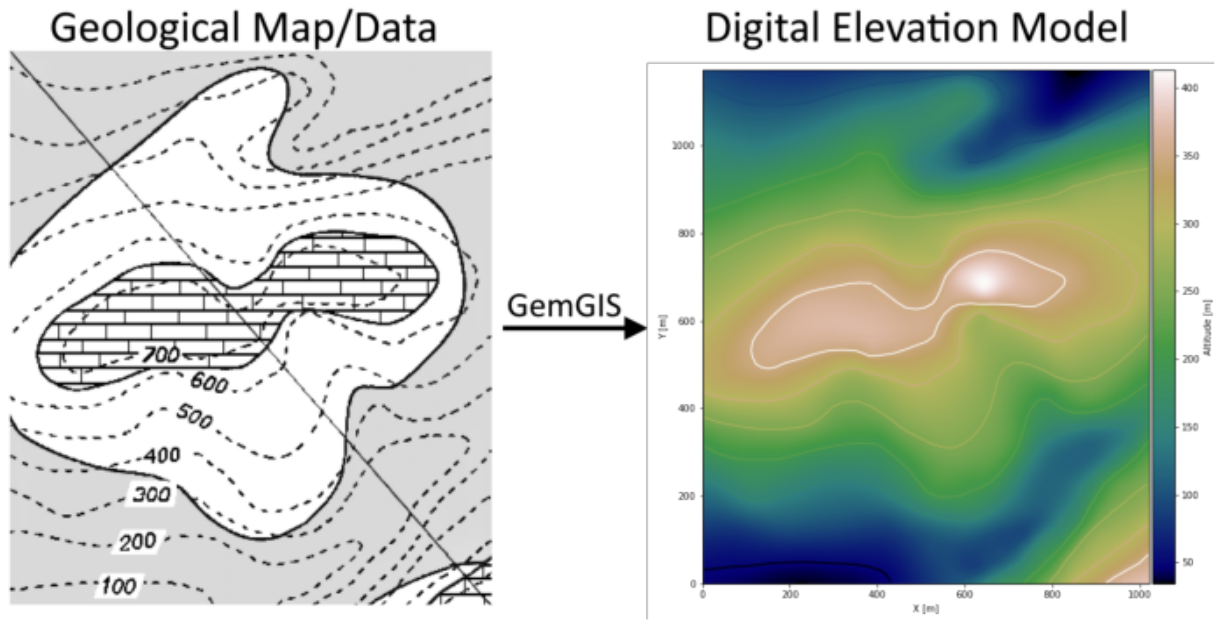
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example26/'
gg.download_gemgis_data.download_tutorial_data(filename="example26_unconformable_folded_
↪layers.zip", dirpath=file_path)
```

7.26.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example26.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo26.shp')
topo['Z'] = topo['Z']*0.5
topo.head()
```

[6]:

	id	Z	geometry
0	None	50.00	LINESTRING (1.874 31.940, 31.950 34.114, 67.10...
1	None	100.00	LINESTRING (2.961 153.696, 32.675 153.696, 65...
2	None	350.00	LINESTRING (924.099 1.864, 942.217 14.546, 974...
3	None	300.00	LINESTRING (844.378 1.139, 873.367 23.606, 899...
4	None	250.00	LINESTRING (793.284 1.864, 816.113 29.766, 834...

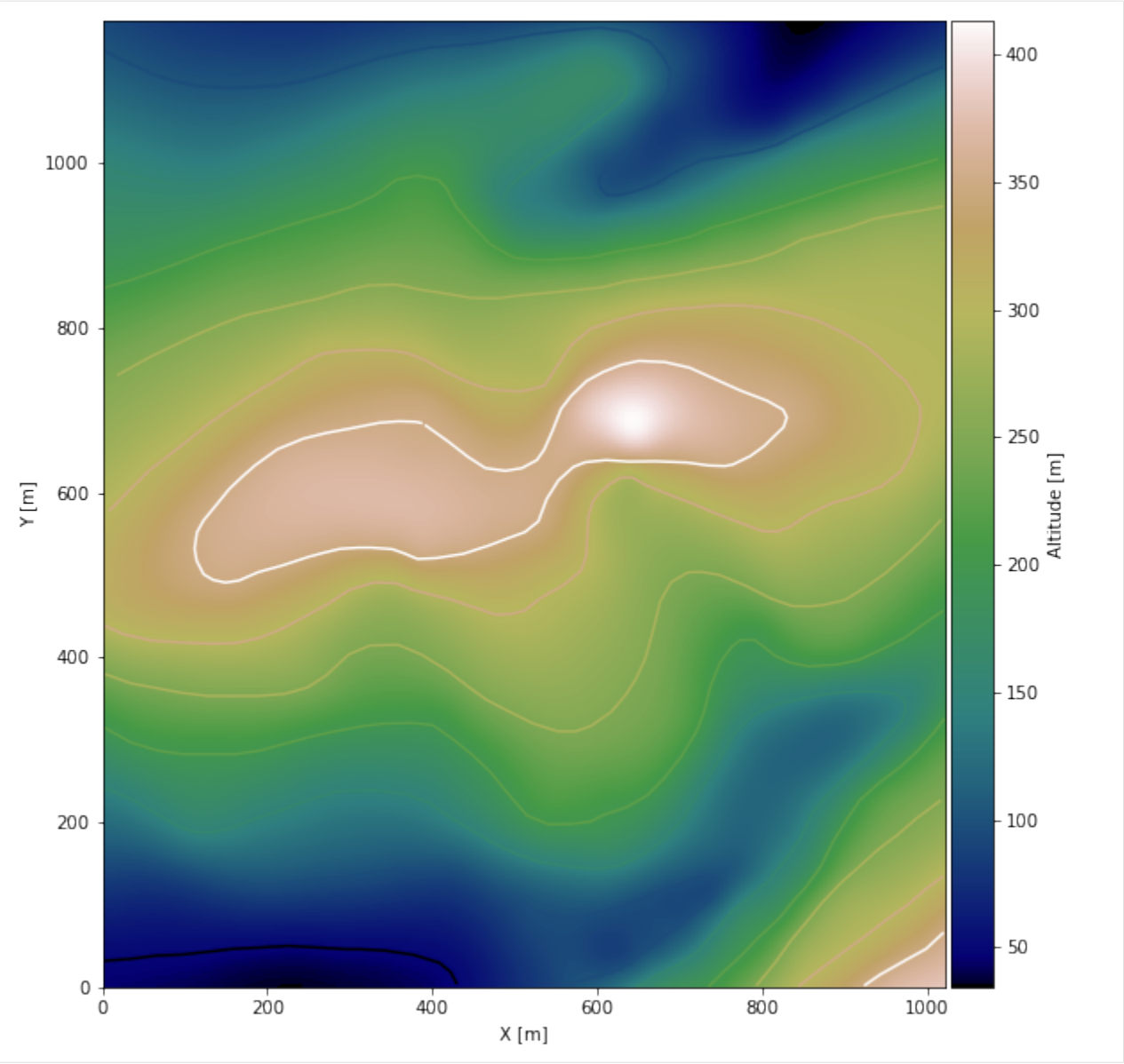
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10,10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0,1022,0,1172], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

[8]:
```
Text(94.3788293897884, 0.5, 'Y [m]')
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as as raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster26.tif', extent = [0, 1022, 0, 1172], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

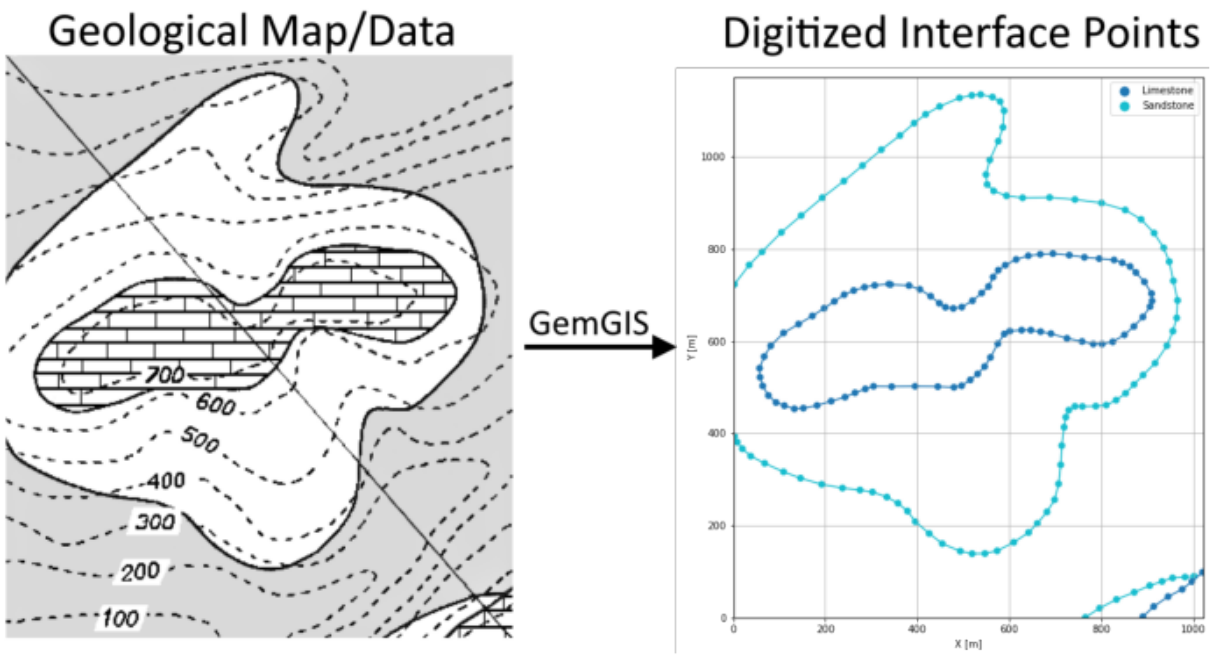
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster26.tif')
```

7.26.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/interfaces_example26.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces26.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Limestone	LINESTRING (341.774 722.613, 381.997 720.438, ...
1	None	Limestone	LINESTRING (890.761 2.226, 914.315 24.330, 945...
2	None	Claystone	LINESTRING (765.744 0.777, 796.546 20.707, 833...
3	None	Claystone	LINESTRING (2.236 722.975, 34.487 765.010, 62...

Extracting XY coordinates from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

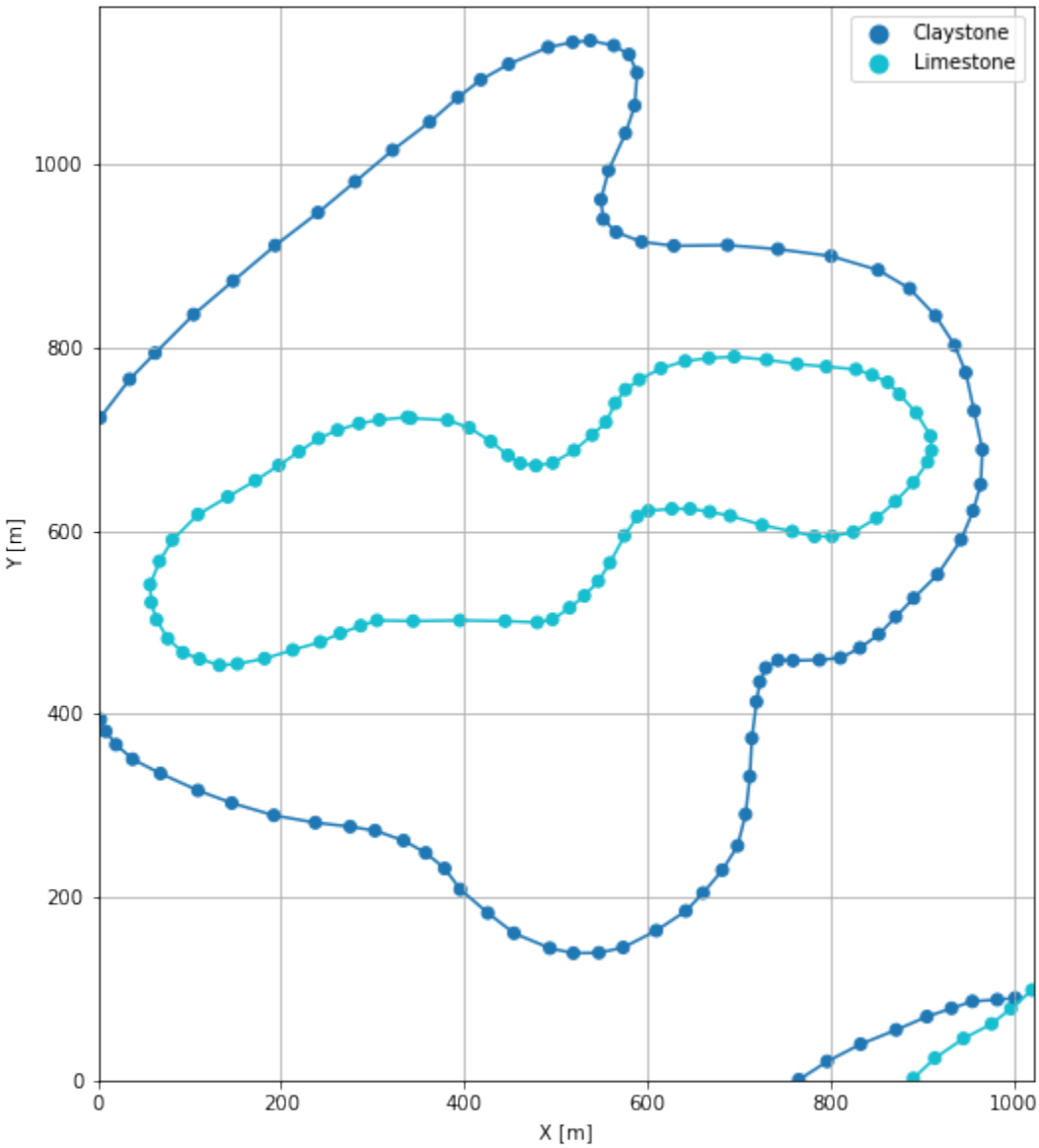
[12]:

	formation	geometry	X	Y	Z
0	Limestone	POINT (341.774 722.613)	341.77	722.61	338.52
54	Limestone	POINT (287.057 496.133)	287.06	496.13	312.83
62	Limestone	POINT (92.828 466.781)	92.83	466.78	329.27
61	Limestone	POINT (110.946 459.896)	110.95	459.90	328.27
60	Limestone	POINT (132.688 453.011)	132.69	453.01	325.96

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0,1022)
plt.ylim(0,1172)
```

[13]:
```
(0.0, 1172.0)
```



7.26.6 Orientations from Strike Lines

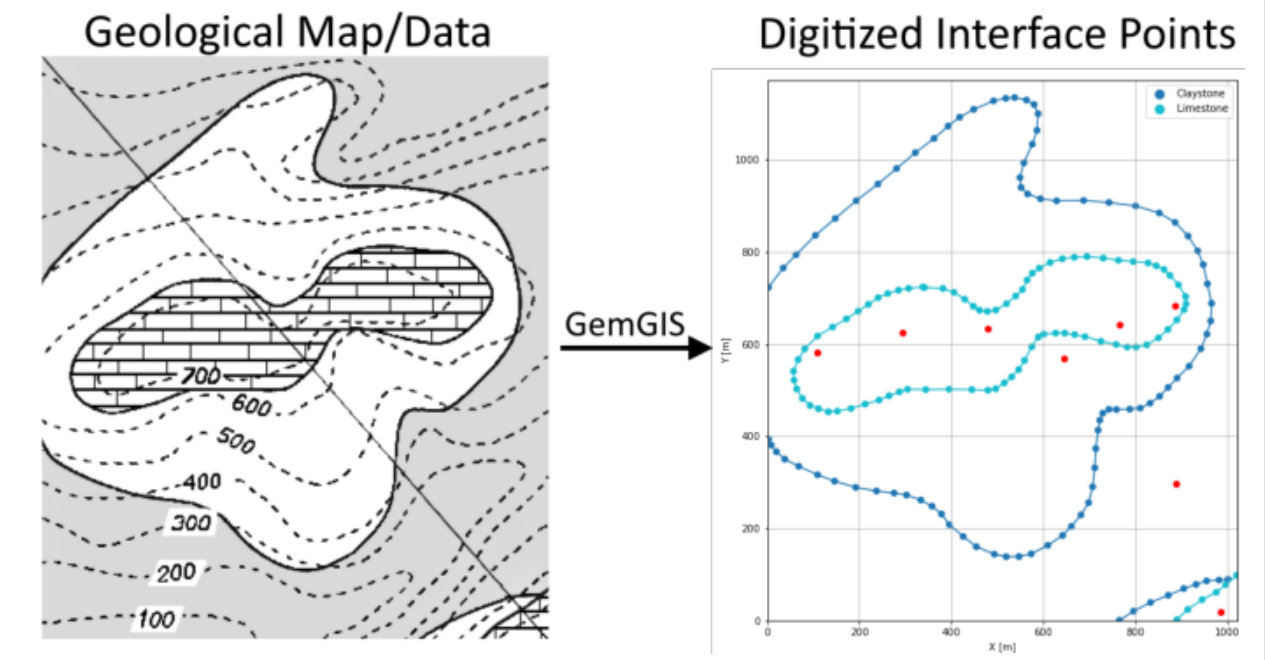
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
```

(continues on next page)

(continued from previous page)

```
img = mpimg.imread('../images/orientations_example26.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes26.shp')
strikes['Z'] = strikes['Z']*0.5
strikes
```

[15]:

	id	formation	Z	geometry
0	2	Limestone2	350.00	LINESTRING (208.061 655.212, 205.887 509.540)
1	1	Limestone2	350.00	LINESTRING (750.887 734.208, 750.887 632.021)
2	4	Claystone1	300.00	LINESTRING (953.088 757.400, 950.914 613.178)
3	3	Claystone1	250.00	LINESTRING (823.361 895.099, 821.911 464.607)
4	2	Limestone1	350.00	LINESTRING (962.510 24.693, 963.235 0.052)
5	1	Limestone1	350.00	LINESTRING (1009.618 51.508, 1009.618 2.226)
6	2	Claystone1	200.00	LINESTRING (712.476 909.594, 707.403 296.468)
7	1	Claystone1	150.00	LINESTRING (587.097 917.566, 576.226 147.898)
8	1	Claystone2	150.00	LINESTRING (387.070 1070.485, 385.621 221.096)
9	2	Claystone2	200.00	LINESTRING (202.988 919.016, 199.364 287.047)
10	3	Claystone2	250.00	LINESTRING (16.006 744.355, 12.382 374.740)
11	1	Claystone3	250.00	LINESTRING (822.636 462.433, 821.187 34.114)
12	2	Claystone3	300.00	LINESTRING (951.639 606.655, 955.987 85.570)

Calculate Orientations for each formation

[16]:
```
orientations_claystone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Claystone1'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_claystone1
```

[16]:
```
    dip  azimuth      Z                 geometry  polarity   formation      X  \
0 21.73   270.68 175.00  POINT (645.801 567.882)      1.00  Claystone1 645.80
1 24.25   270.38 225.00  POINT (766.288 641.442)      1.00  Claystone1 766.29
2 21.26   270.26 275.00  POINT (887.319 682.571)      1.00  Claystone1 887.32

       Y
0 567.88
1 641.44
2 682.57
```

[17]:
```
orientations_claystone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Claystone2'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_claystone2
```

[17]:
```
    dip  azimuth      Z                 geometry  polarity   formation      X  \
0 15.22    90.18 175.00  POINT (293.761 624.411)      1.00  Claystone2 293.76
1 15.05    90.39 225.00  POINT (107.685 581.289)      1.00  Claystone2 107.68

       Y
0 624.41
1 581.29
```

[18]:
```
orientations_claystone3 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Claystone3'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_claystone3
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity   formation      X  \
0 21.01   269.79 275.00  POINT (887.862 297.193)      1.00  Claystone3 887.86

       Y
0 297.19
```

[19]:
```
orientations_limestone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Limestone1'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_limestone1
```

[19]:
```
   dip  azimuth      Z                geometry  polarity   formation      X  \
0 0.00    88.32 350.00  POINT (986.245 19.620)      1.00  Limestone1 986.24

      Y
0 19.62
```

[20]:
```
orientations_limestone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Limestone2'].sort_values(by='Z',␣
↪ascending=True).reset_index())
orientations_limestone2
```

[20]:
```
   dip  azimuth      Z                 geometry  polarity   formation      X  \
0 0.00    90.86 350.00  POINT (478.930 632.745)      1.00  Limestone2 478.93

       Y
0 632.75
```

Merging Orientations

[21]:
```
import pandas as pd
orientations = pd.concat([orientations_claystone1, orientations_claystone2, orientations_
↪claystone3, orientations_limestone1, orientations_limestone2]).reset_index()
orientations['formation'] = ['Claystone', 'Claystone', 'Claystone', 'Claystone',
↪'Claystone', 'Claystone', 'Limestone', 'Limestone']
orientations
```

[21]:
```
   index   dip  azimuth      Z                 geometry  polarity  formation  \
0      0 21.73   270.68 175.00  POINT (645.801 567.882)      1.00  Claystone
1      1 24.25   270.38 225.00  POINT (766.288 641.442)      1.00  Claystone
2      2 21.26   270.26 275.00  POINT (887.319 682.571)      1.00  Claystone
3      0 15.22    90.18 175.00  POINT (293.761 624.411)      1.00  Claystone
4      1 15.05    90.39 225.00  POINT (107.685 581.289)      1.00  Claystone
5      0 21.01   269.79 275.00  POINT (887.862 297.193)      1.00  Claystone
6      0  0.00    88.32 350.00   POINT (986.245 19.620)      1.00  Limestone
7      0  0.00    90.86 350.00  POINT (478.930 632.745)      1.00  Limestone

       X      Y
0 645.80 567.88
1 766.29 641.44
2 887.32 682.57
3 293.76 624.41
4 107.68 581.29
5 887.86 297.19
6 986.24  19.62
7 478.93 632.75
```

Plotting the Orientations

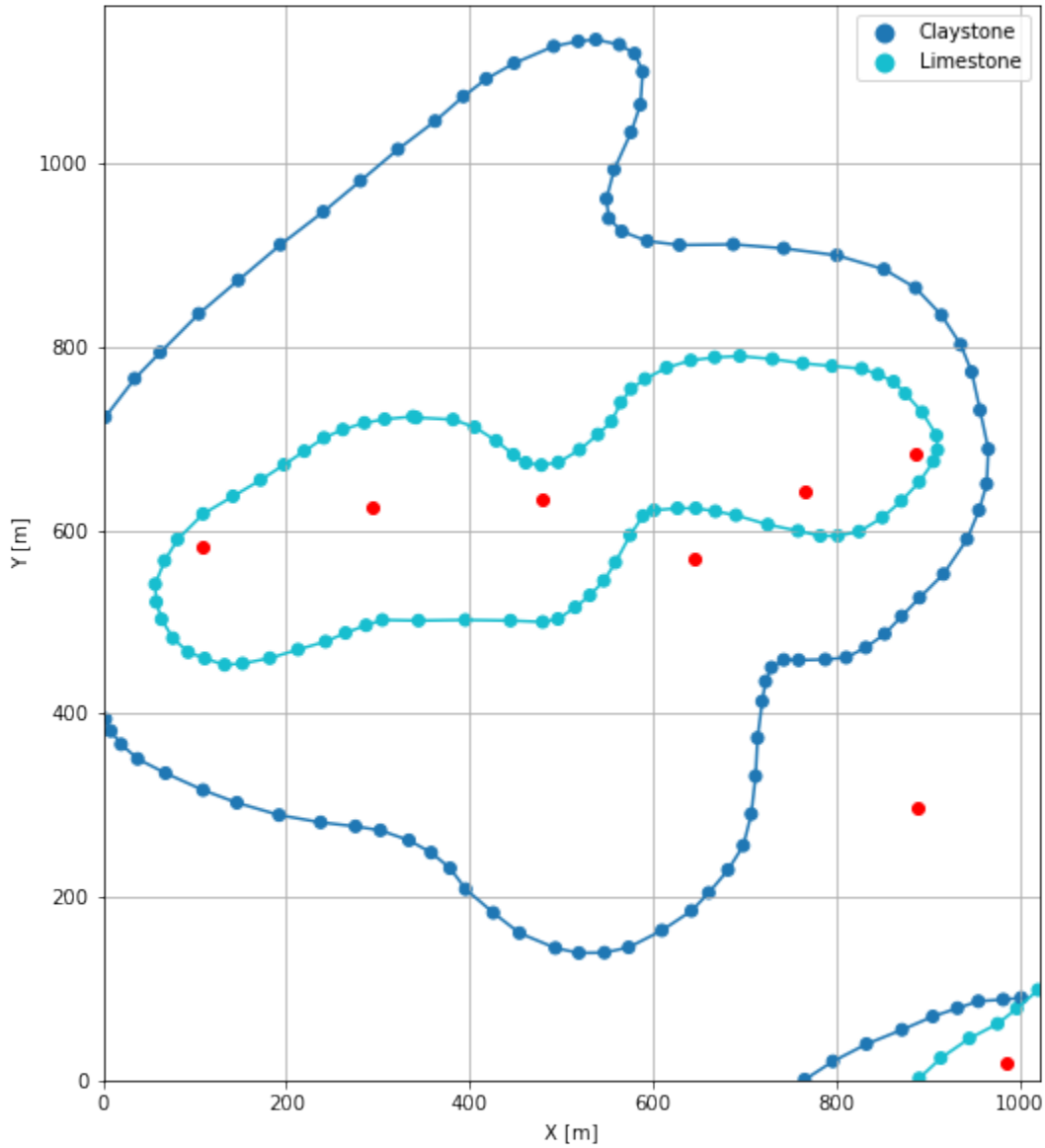
[22]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
```

(continues on next page)

(continued from previous page)

```
plt.xlim(0,1022)
plt.ylim(0,1172)
```

[22]: (0.0, 1172.0)



7.26.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[23]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[24]:
```
geo_model = gp.create_model('Model26')
geo_model
```

[24]:
```
Model26  2022-04-11 08:40
```

Initiate Data

[25]:
```
gp.init_data(geo_model, [0,1022,0,1172,0,900], [100,100,100],
             surface_points_df = interfaces_coords[interfaces_coords['Z']!=0],
             orientations_df = orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[25]:
```
Model26  2022-04-11 08:40
```

Model Surfaces

[26]:
```
geo_model.surfaces
```

[26]:

	surface	series	order_surfaces	color	id
0	Limestone	Default series	1	#015482	1
1	Claystone	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[27]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Limestone'),
                          'Strata2': ('Claystone')
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Sandstone')
```

[27]:
```
     surface   series  order_surfaces    color  id
0  Limestone  Strata1               1  #015482   1
1  Claystone  Strata2               1  #9f0052   2
2  Sandstone  Strata2               2  #ffbe00   3
```

Adding additional Orientations

[28]:
```
geo_model.add_orientations(X=250, Y=1000, Z=300, surface='Limestone', orientation = [0,0,
↪1])
geo_model.add_orientations(X=750, Y=1000, Z=300, surface='Limestone', orientation = [0,0,
↪1])
geo_model.add_orientations(X=250, Y=200, Z=300, surface='Limestone', orientation = [0,0,
↪1])
geo_model.add_orientations(X=750, Y=200, Z=300, surface='Limestone', orientation = [0,0,
↪1])
geo_model.add_orientations(X=500, Y=50, Z=100, surface='Claystone', orientation = [0,0,
↪1])
geo_model.add_orientations(X=500, Y=600, Z=350, surface='Claystone', orientation = [0,0,
↪1])
geo_model.add_orientations(X=500, Y=1000, Z=150, surface='Claystone', orientation = [0,0,
↪1])
```

[28]:
```
        X       Y      Z   G_x   G_y  G_z  smooth    surface
6  986.24   19.62 350.00  0.00  0.00 1.00    0.01  Limestone
7  478.93  632.75 350.00  0.00  0.00 1.00    0.01  Limestone
8  250.00 1000.00 300.00  0.00  0.00 1.00    0.01  Limestone
9  750.00 1000.00 300.00  0.00  0.00 1.00    0.01  Limestone
10 250.00  200.00 300.00  0.00  0.00 1.00    0.01  Limestone
11 750.00  200.00 300.00  0.00  0.00 1.00    0.01  Limestone
0  645.80  567.88 175.00 -0.37  0.00 0.93    0.01  Claystone
1  766.29  641.44 225.00 -0.41  0.00 0.91    0.01  Claystone
2  887.32  682.57 275.00 -0.36  0.00 0.93    0.01  Claystone
3  293.76  624.41 175.00  0.26 -0.00 0.96    0.01  Claystone
4  107.68  581.29 225.00  0.26 -0.00 0.97    0.01  Claystone
5  887.86  297.19 275.00 -0.36 -0.00 0.93    0.01  Claystone
12 500.00   50.00 100.00  0.00  0.00 1.00    0.01  Claystone
13 500.00  600.00 350.00  0.00  0.00 1.00    0.01  Claystone
14 500.00 1000.00 150.00  0.00  0.00 1.00    0.01  Claystone
```

Showing the Number of Data Points

[29]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[29]:
```
     surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0  Limestone  Strata1               1  #015482   1                 85                ␣
↪6
1  Claystone  Strata2               1  #9f0052   2                 90                ␣
↪9
2  Sandstone  Strata2               2  #ffbe00   3                  0                ␣
↪0
```

Loading Digital Elevation Model

[30]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster26.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[30]:
```
Grid Object. Values:
array([[   5.11      ,    5.86      ,    4.5       ],
       [   5.11      ,    5.86      ,   13.5       ],
       [   5.11      ,    5.86      ,   22.5       ],
       ...,
       [1019.49509804, 1159.4248927 ,   76.11084747],
       [1019.49509804, 1164.45493562,   73.8138504 ],
       [1019.49509804, 1169.48497854,   71.72492218]])
```

Defining Custom Section

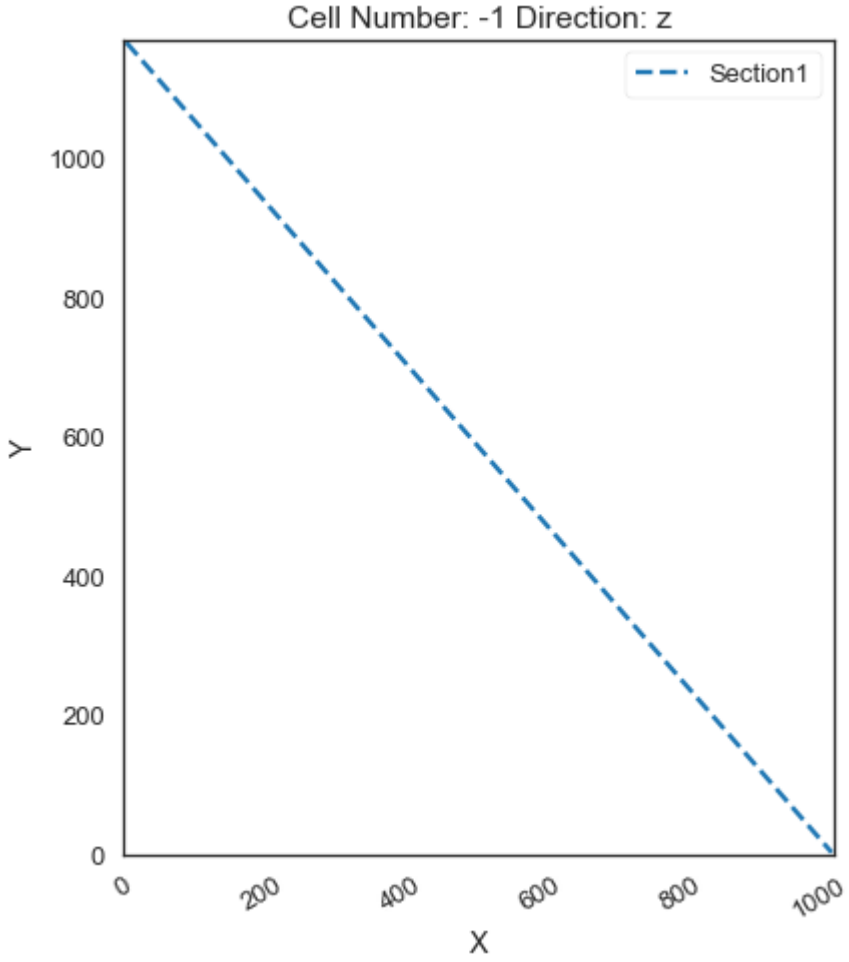
[31]:
```
custom_section = gpd.read_file(file_path + 'customsection26.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='name')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[31]:
```
                                          start                                  ␣
↪stop resolution    dist
Section1  [3.685512376796794, 1168.3245583161138]  [1018.3144876418132, 2.
↪95070672600923]  [100, 80] 1545.18
```

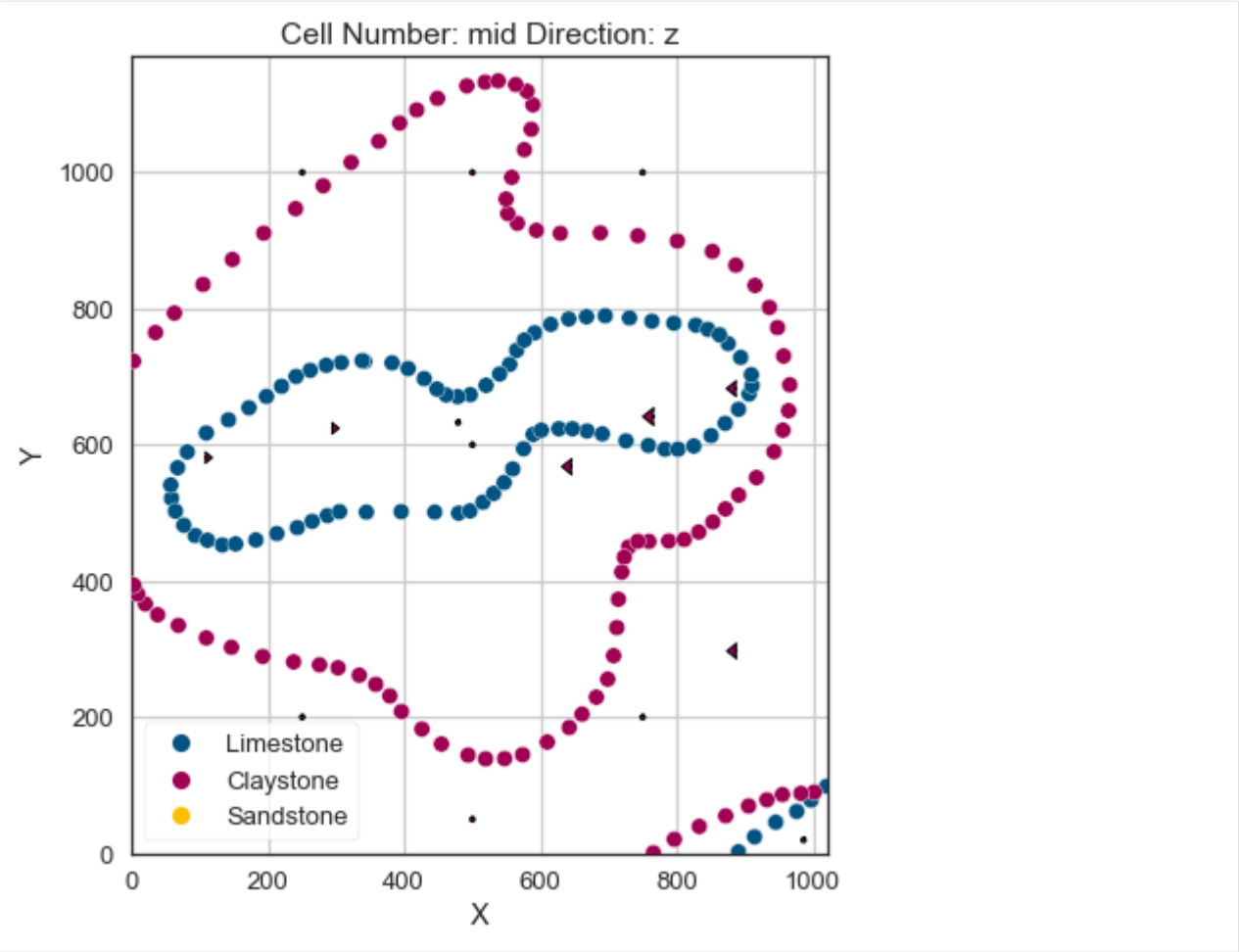
[32]:
```
gp.plot.plot_section_traces(geo_model)
```

[32]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1f10c056a90>
```

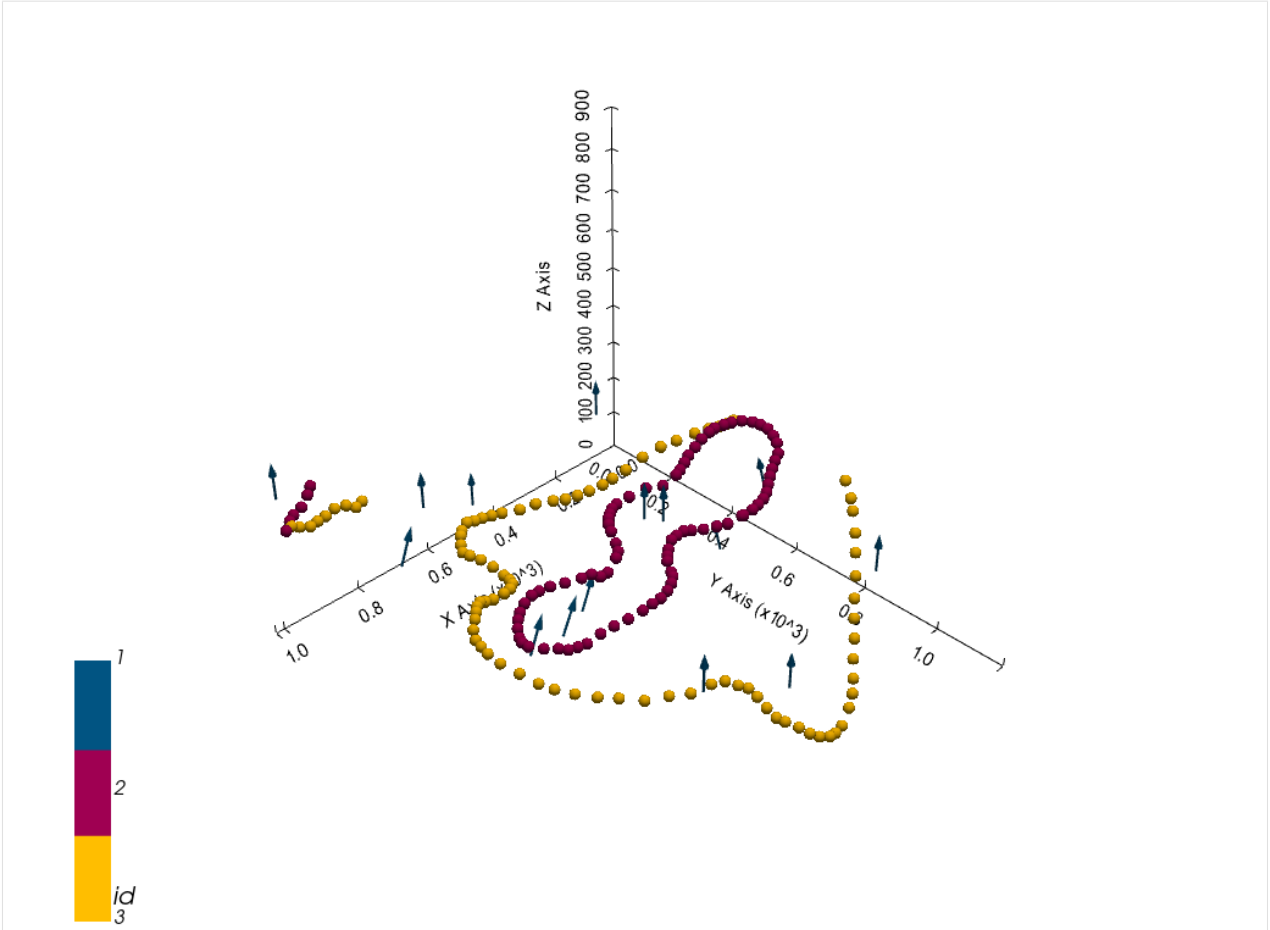


Plotting Input Data

[33]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[34]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[34]:
```
<gempy.plot.vista.GemPyToVista at 0x1f107c86250>
```

Setting the Interpolator

[35]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                   values
range             1796.68
$C_o$            76858.76
drift equations    [3, 3]
```

[35]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1f10600b7c0>
```

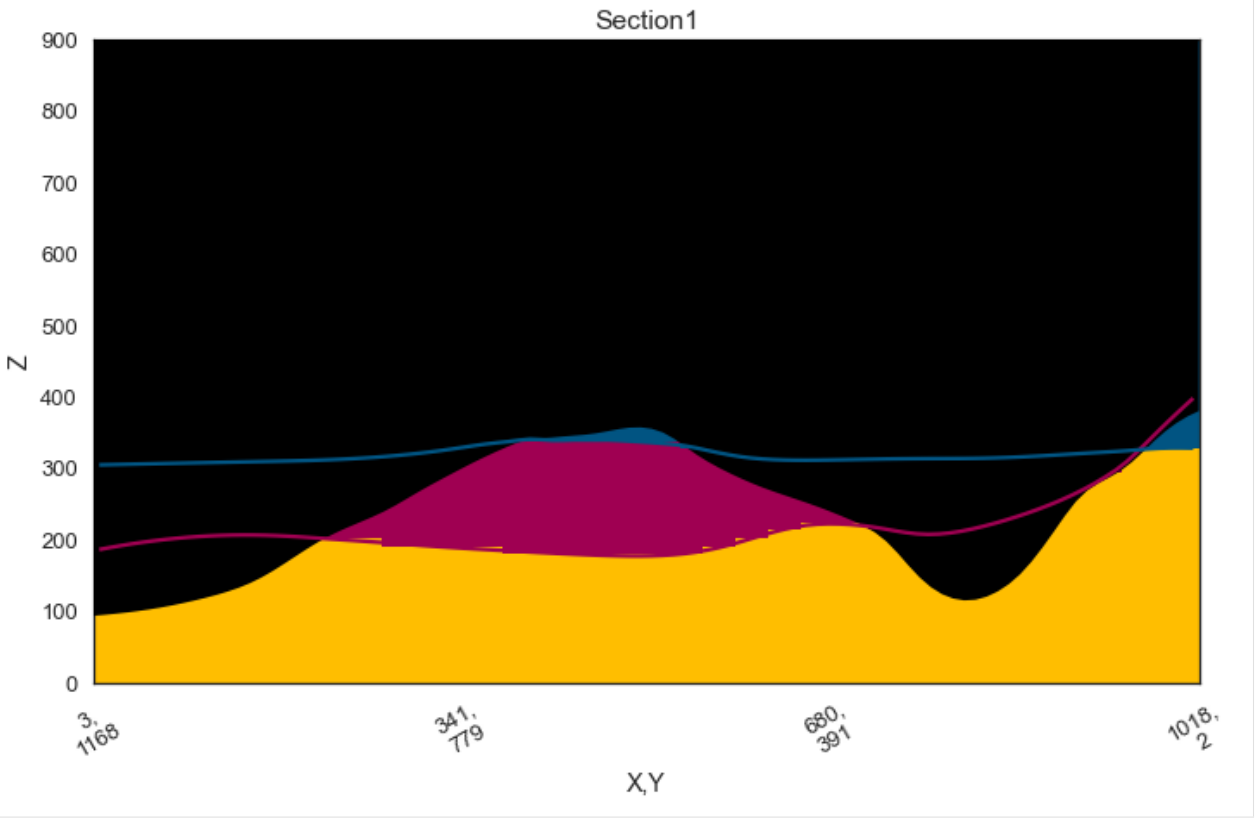
Computing Model

[36]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

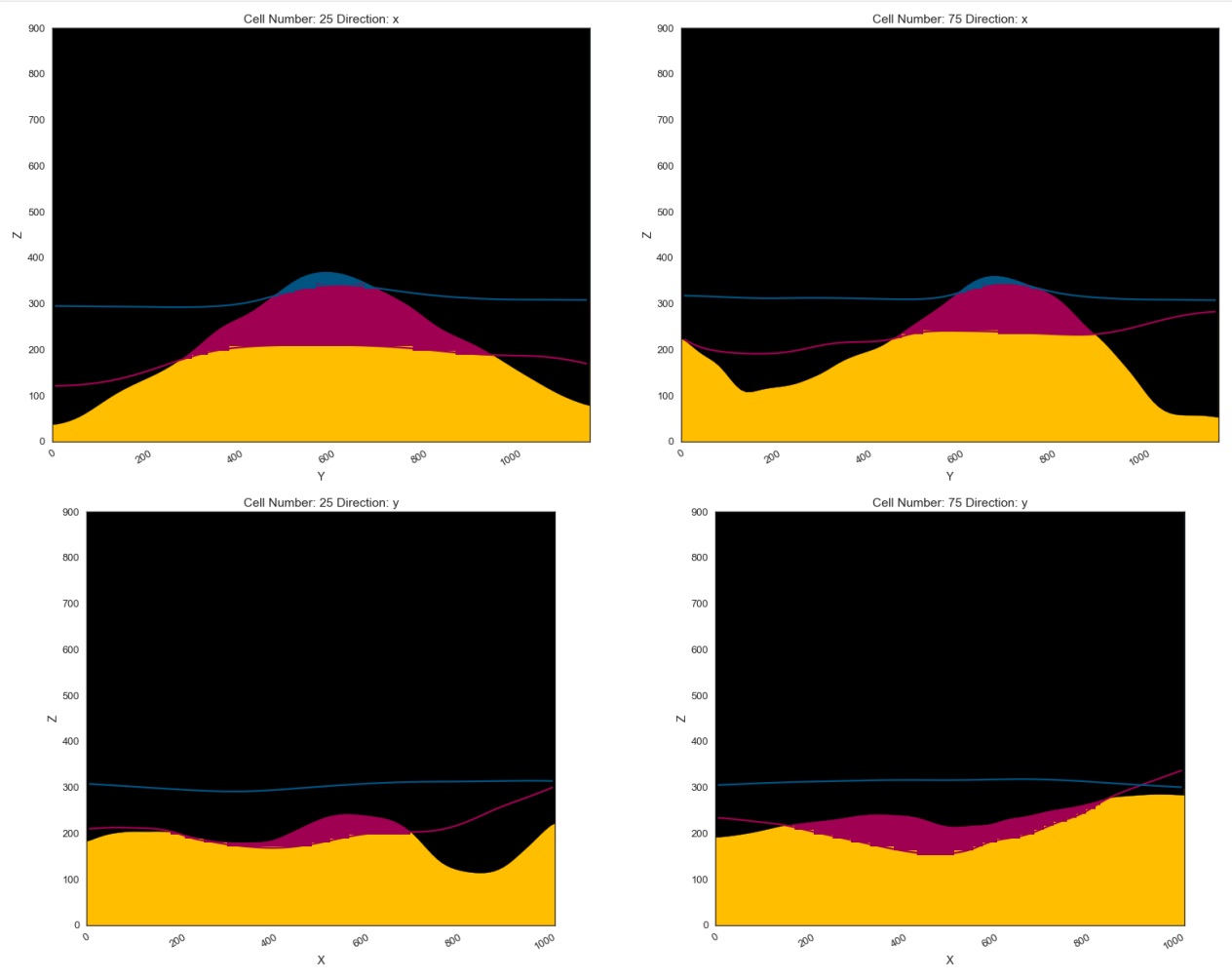
[37]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[37]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1f108e93040>
```



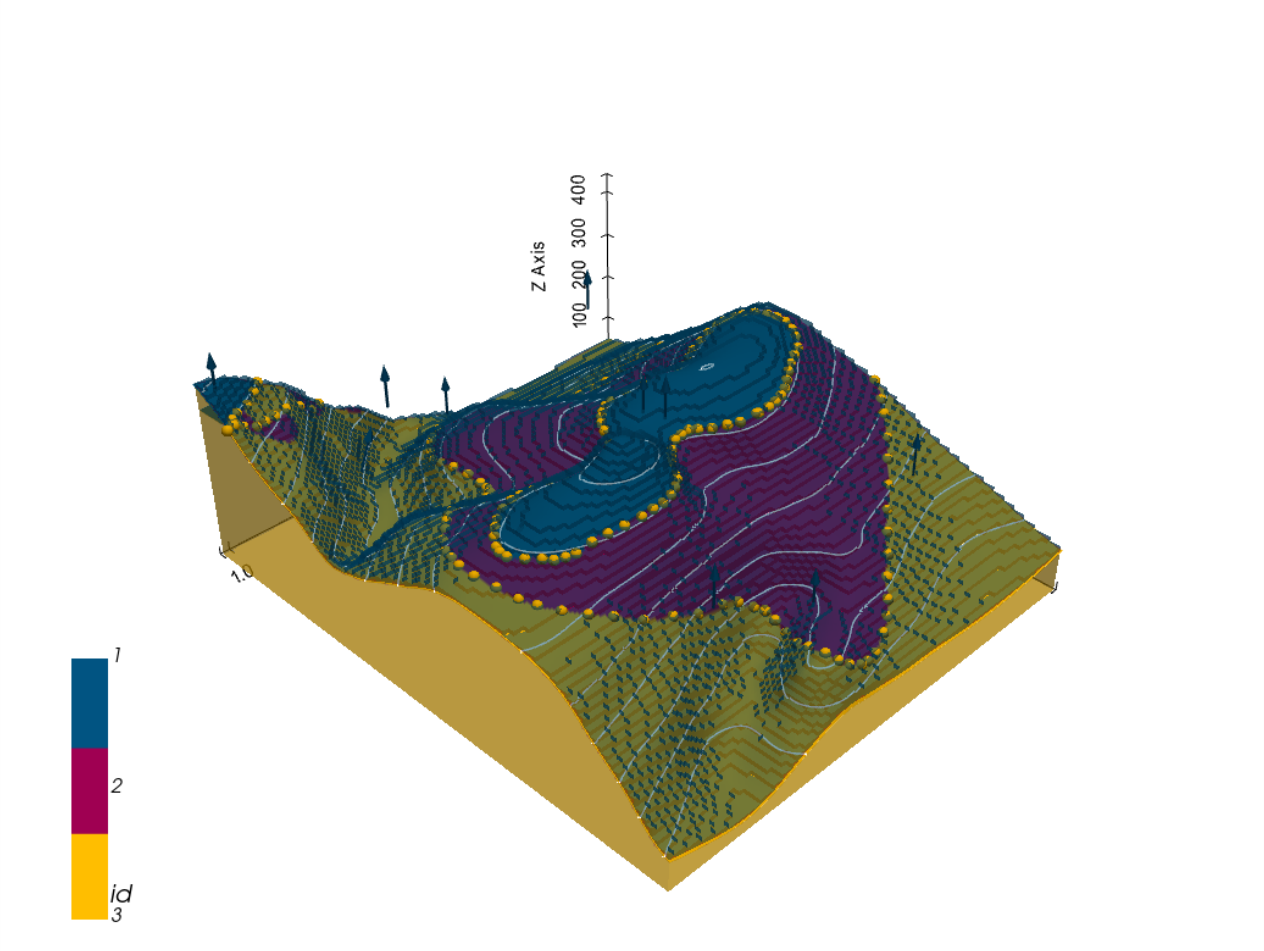
[38]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25,75,25,75], show_
↪topography=True, show_data=False)
```

[38]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1f108ff1c70>
```



Plotting 3D Model

[39]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



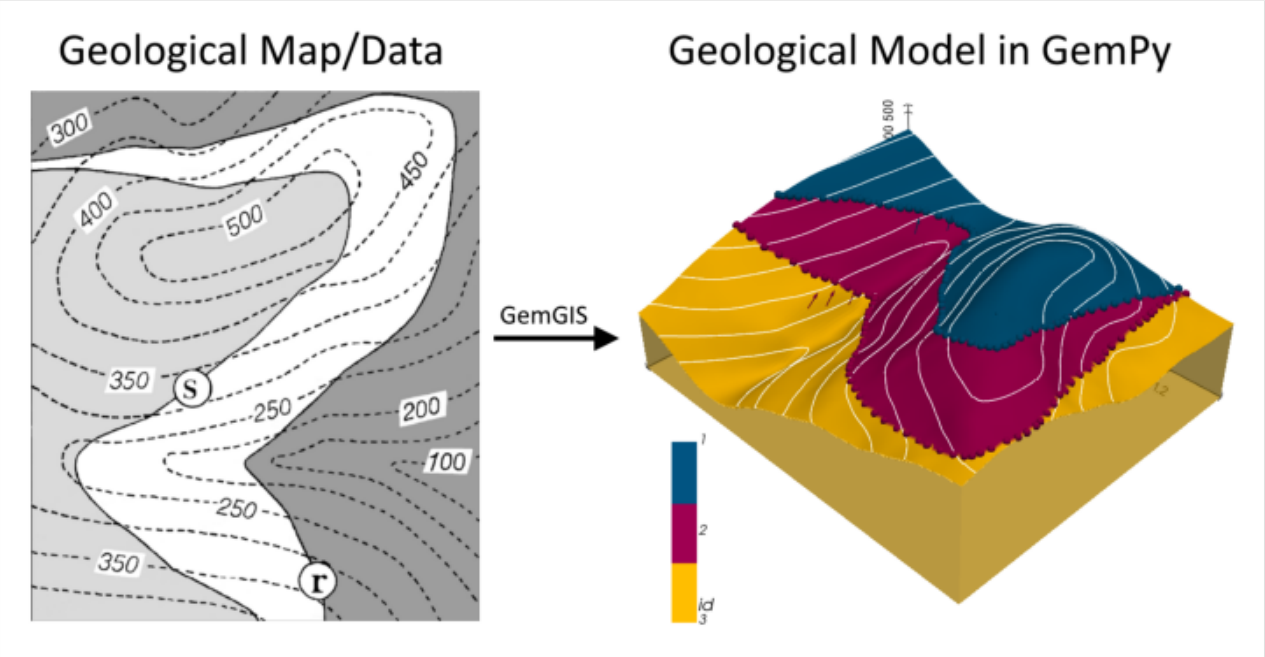
[ ]:

7.27 Example 27 - Planar Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 1187 m wide (W-E extent) and 1400 m high (N-S extent). The vertical model extents varies between 0 m and 600 m. The model represents two planar stratigraphic units (blue and red) dipping towards the western and southwestern directions above an unspecified basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and were used to calculate orientations for the `GemPy` model. These will be loaded into the model directly. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example27.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.27.1 Licensing

Computational Geosciences and Reservoir Engineering, RWTH Aachen University, Authors: Alexander Juestel. For more information contact: alexander.juestel(at)rwth-aachen.de

This work is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/)

7.27.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.27.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example27/'
gg.download_gemgis_data.download_tutorial_data(filename="example27_planar_dipping_layers.
↪zip", dirpath=file_path)
```

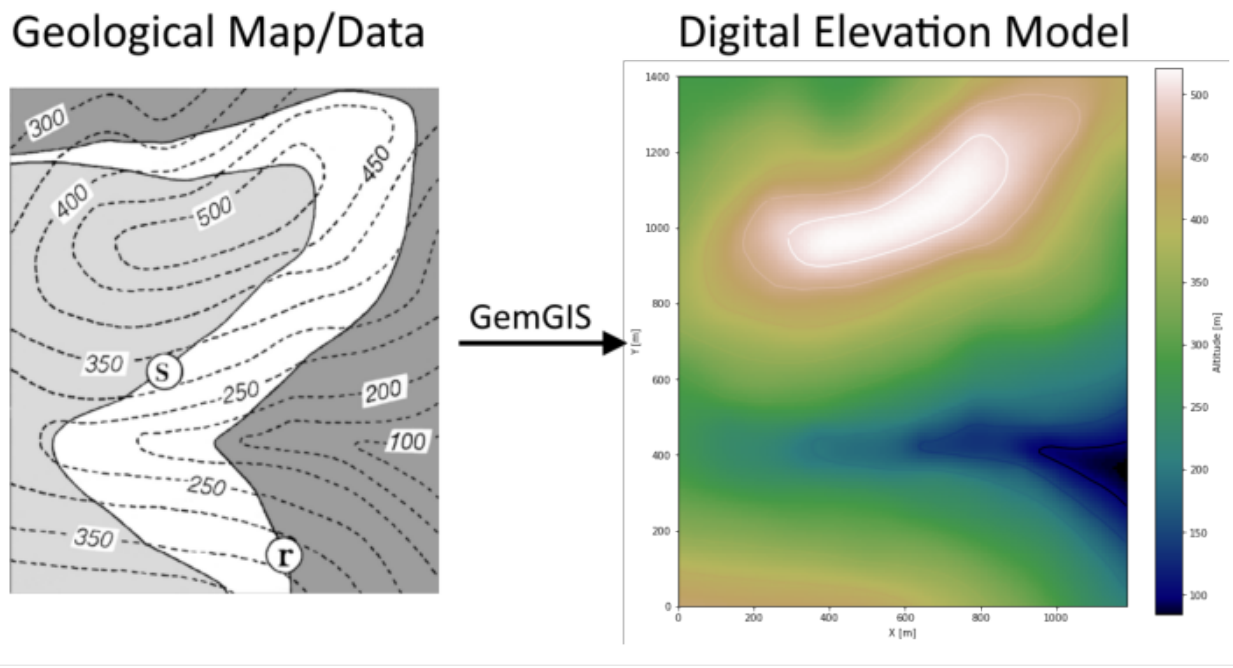
(continues on next page)

(continued from previous page)

7.27.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
img = mpimg.imread('../images/dem_example27.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo27.shp')
topo.head()
```

[6]:
```
     id    Z                                           geometry
0  None  300  LINESTRING (2.817 1234.197, 25.768 1257.148, 5...
1  None  350  LINESTRING (4.116 1001.226, 21.004 1029.373, 5...
2  None  350  LINESTRING (4.116 730.148, 47.852 706.764, 92...
3  None  300  LINESTRING (2.817 617.992, 45.687 591.144, 85...
4  None  500  LINESTRING (292.732 981.956, 305.723 995.271, ...
```

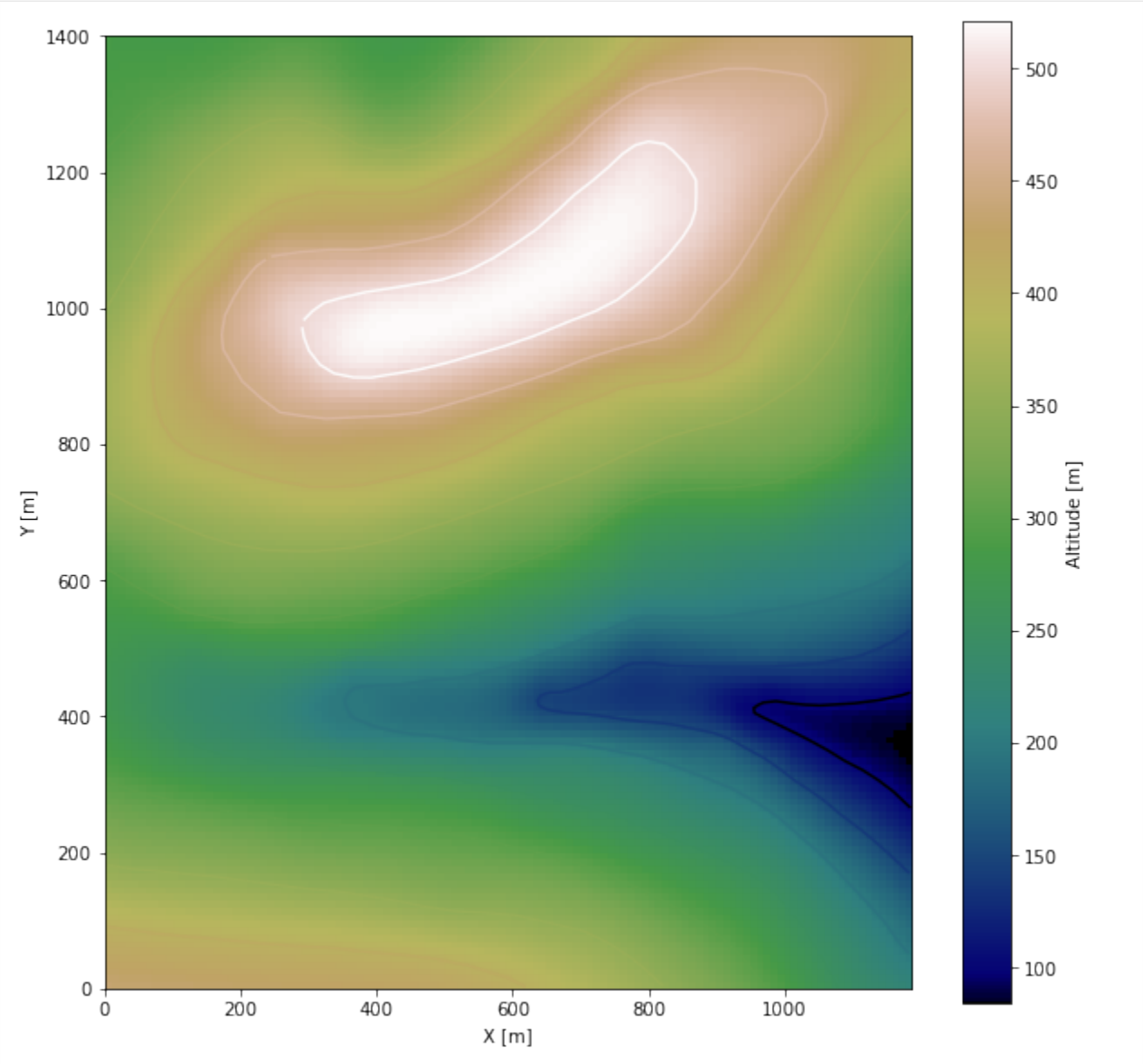
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 1187, 0, 1400], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1187)
plt.ylim(0, 1400)
```

[8]:
```
(0.0, 1400.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster27.tif', extent = [0, 1187, 0, 1400], crs ='$
$EPSG : 4326', overwrite_f ile = True)$

Opening Raster

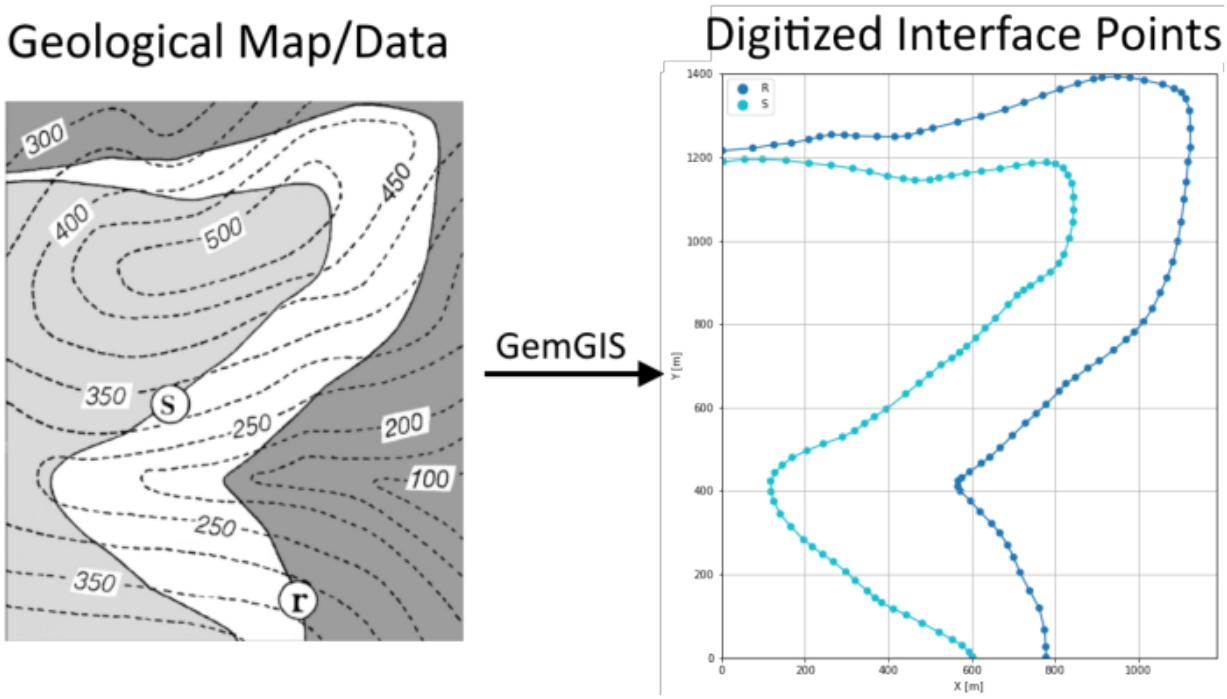
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster27.tif')
```

7.27.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../images/interfaces_example27.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces27.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	S	LINESTRING (4.008 1189.324, 55.403 1194.683, 9...
1	None	R	LINESTRING (2.059 1215.631, 74.890 1221.964, 1...

Extracting Z coordinate from Digital Elevation Model

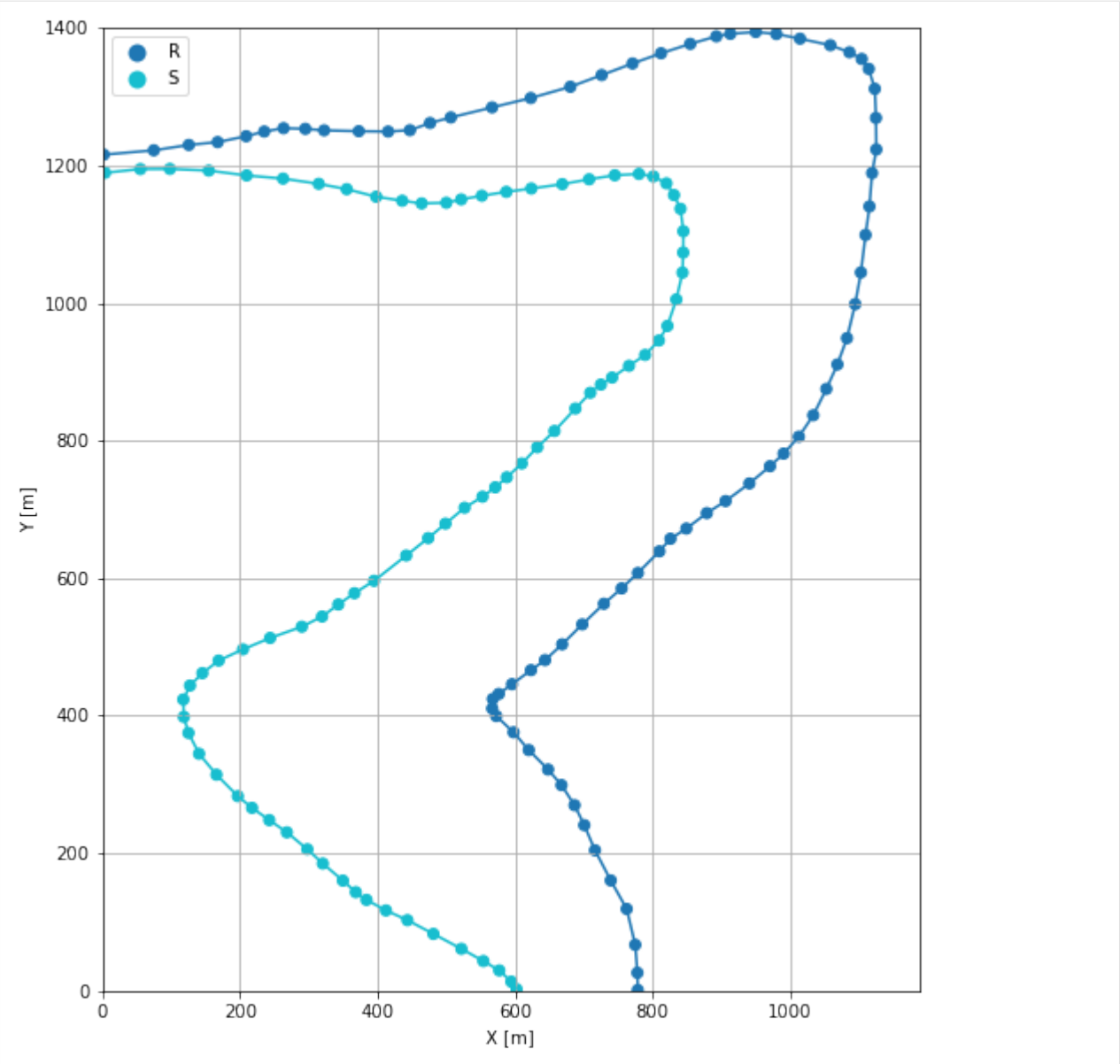
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords
```

[12]:
```
    formation                  geometry      X       Y      Z
0           S    POINT (4.008 1189.324)   4.01 1189.32 312.04
1           S   POINT (55.403 1194.683)  55.40 1194.68 324.30
2           S   POINT (98.273 1194.926)  98.27 1194.93 339.27
3           S  POINT (154.784 1192.491) 154.78 1192.49 359.42
4           S  POINT (209.833 1185.427) 209.83 1185.43 377.76
..        ...                        ...    ...     ...    ...
148         R   POINT (738.890 160.683) 738.89  160.68 302.06
149         R   POINT (762.031 119.275) 762.03  119.27 324.07
150         R    POINT (774.453 67.148) 774.45   67.15 343.90
151         R    POINT (777.620 26.470) 777.62   26.47 358.34
152         R     POINT (778.107 1.382) 778.11    1.38 361.61

[153 rows x 5 columns]
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1187)
plt.ylim(0, 1400)
```

[13]:
```
(0.0, 1400.0)
```



7.27.6 Orientations from Strike Lines

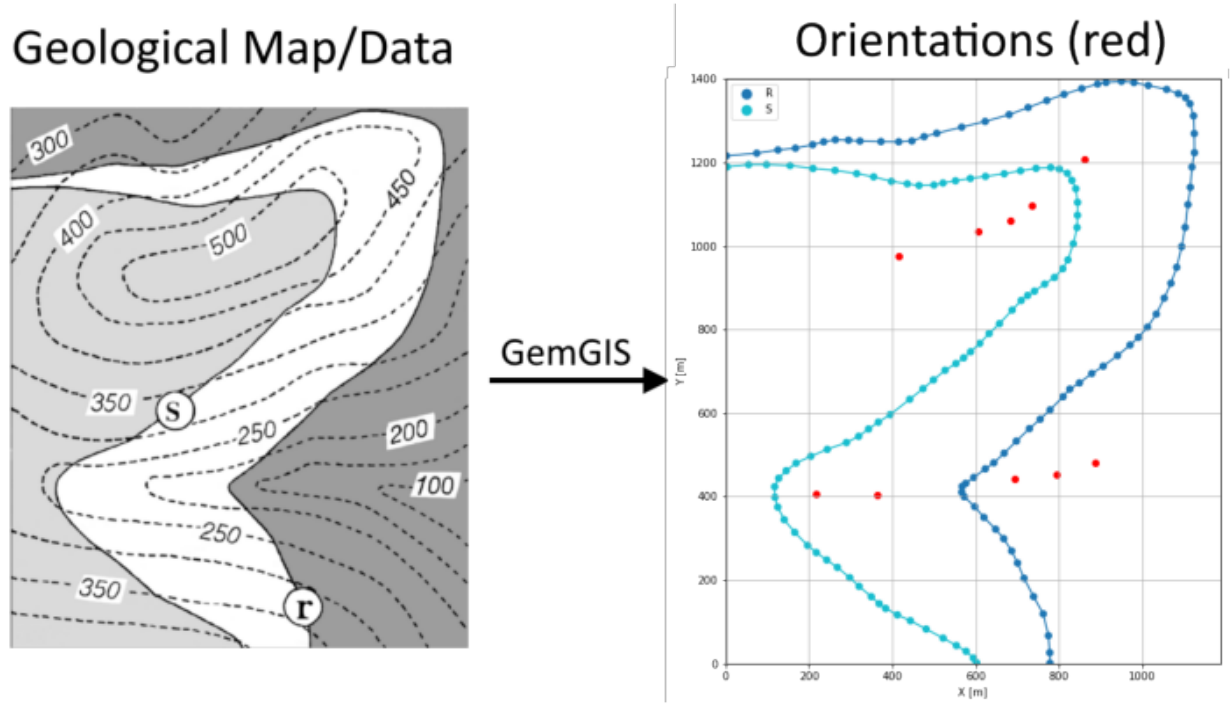
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
img = mpimg.imread('../images/orientations_example27.png')
plt.figure(figsize=(10, 10))
```

(continues on next page)

(continued from previous page)

```
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes27.shp')
strikes
```

[15]:

	id	formation	Z	geometry
0	1	S1	250	LINESTRING (163.553 475.146, 133.349 358.227)
1	2	S1	300	LINESTRING (320.906 542.374, 246.858 248.129)
2	3	S1	350	LINESTRING (512.848 689.497, 378.391 133.159)
3	1	R1	200	LINESTRING (663.380 498.529, 613.203 359.201)
4	2	R1	250	LINESTRING (810.503 639.319, 687.738 272.000)
5	3	R1	300	LINESTRING (938.139 735.777, 738.403 167.747)
6	4	R1	350	LINESTRING (1089.159 979.358, 777.376 39.137)
7	4	S2	500	LINESTRING (707.225 1180.068, 842.656 1093.353)
8	3	S2	450	LINESTRING (576.666 1159.607, 816.349 957.922)
9	2	S2	400	LINESTRING (338.931 1169.350, 687.738 852.696)
10	1	S2	350	LINESTRING (122.632 1193.708, 511.386 689.009)
11	3	R2	400	LINESTRING (705.276 1323.293, 1127.158 1232.681)
12	2	R2	350	LINESTRING (522.104 1273.603, 1094.031 995.921)
13	1	R2	300	LINESTRING (174.271 1230.733, 941.062 738.700)

Calculating Orientations for each formation

[16]:
```
orientations_r1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'R1'].sort_values(by='id', ascending=True).reset_index())
orientations_r1
```

[16]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 27.78   288.65 225.00  POINT (693.706 442.262)      1.00        R1 693.71
1 31.26   289.11 275.00  POINT (793.696 453.711)      1.00        R1 793.70
2 36.87   288.62 325.00  POINT (885.769 480.505)      1.00        R1 885.77

       Y
0 442.26
1 453.71
2 480.50
```

[17]:
```
orientations_r2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'R2'].sort_values(by='id', ascending=True).reset_index())
orientations_r2
```

[17]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 12.59   210.56 325.00  POINT (682.867 1059.739)      1.00        R2 682.87
1 21.85   201.78 375.00  POINT (862.142 1206.375)      1.00        R2 862.14

        Y
0 1059.74
1 1206.37
```

[18]:
```
orientations_s1 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'S1'].sort_values(by='id', ascending=True).reset_index())
orientations_s1
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 20.16   284.17 275.00  POINT (216.166 405.969)      1.00        S1 216.17
1 18.21   283.70 325.00  POINT (364.751 403.289)      1.00        S1 364.75

       Y
0 405.97
1 403.29
```

[19]:
```
orientations_s2 = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'S2'].sort_values(by='id', ascending=True).reset_index())
orientations_s2
```

[19]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 17.72   228.97 375.00   POINT (415.172 976.191)      1.00        S2 415.17
1 18.14   221.58 425.00  POINT (604.921 1034.894)      1.00        S2 604.92
2 26.63   218.55 475.00  POINT (735.724 1097.738)      1.00        S2 735.72

        Y
0  976.19
1 1034.89
2 1097.74
```

Merging Orientations

[20]:
```
import pandas as pd
orientations = pd.concat([orientations_r1, orientations_r2, orientations_s1,␣
↪orientations_s2])
orientations['formation'] = ['R', 'R', 'R', 'R', 'R', 'S', 'S', 'S', 'S', 'S', ]
orientations = orientations[orientations['formation'].isin(['R', 'S'])].reset_index()
orientations
```

[20]:

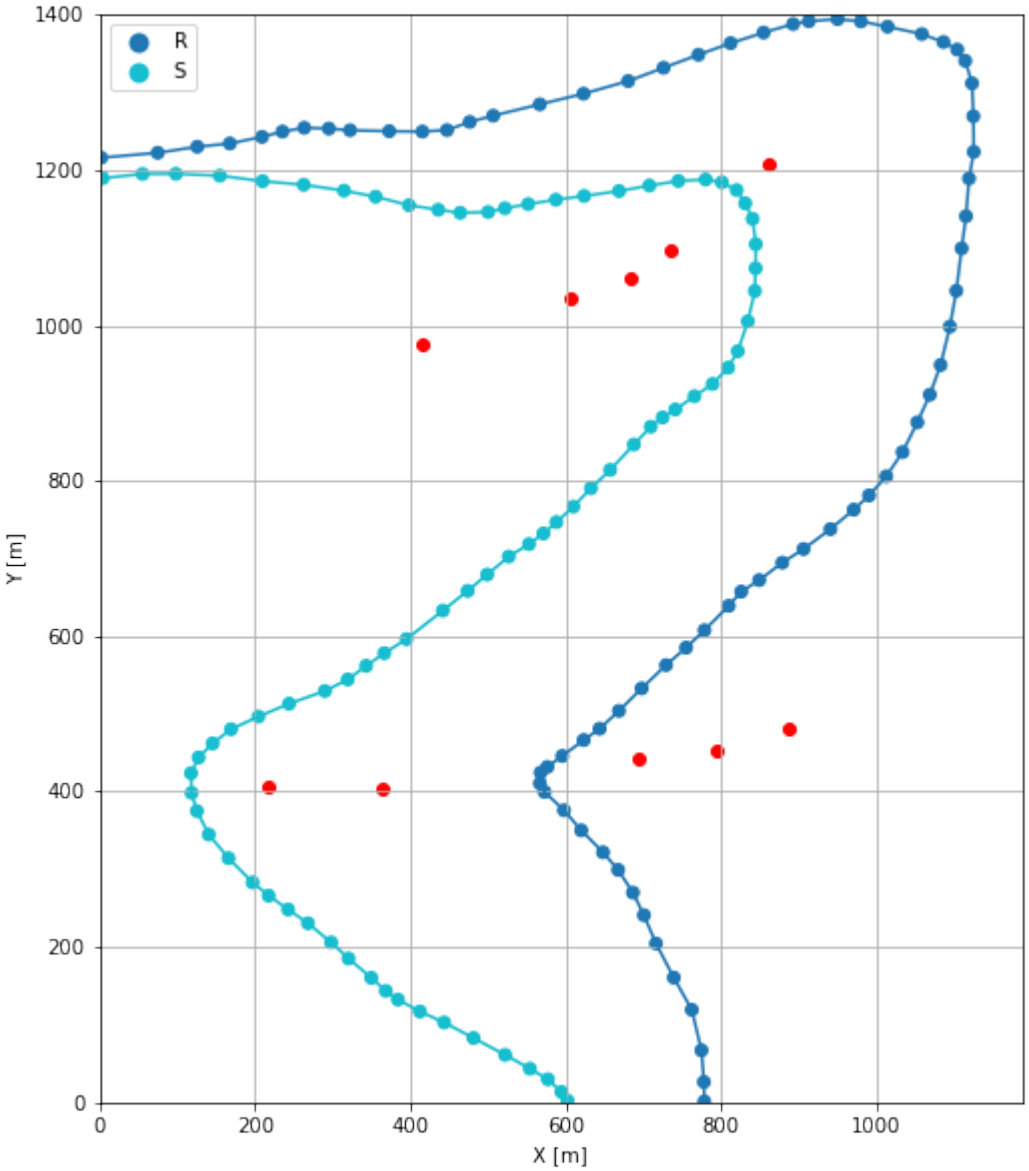
	index	dip	azimuth	Z	geometry	polarity	formation	\
0	0	27.78	288.65	225.00	POINT (693.706 442.262)	1.00	R	
1	1	31.26	289.11	275.00	POINT (793.696 453.711)	1.00	R	
2	2	36.87	288.62	325.00	POINT (885.769 480.505)	1.00	R	
3	0	12.59	210.56	325.00	POINT (682.867 1059.739)	1.00	R	
4	1	21.85	201.78	375.00	POINT (862.142 1206.375)	1.00	R	
5	0	20.16	284.17	275.00	POINT (216.166 405.969)	1.00	S	
6	1	18.21	283.70	325.00	POINT (364.751 403.289)	1.00	S	
7	0	17.72	228.97	375.00	POINT (415.172 976.191)	1.00	S	
8	1	18.14	221.58	425.00	POINT (604.921 1034.894)	1.00	S	
9	2	26.63	218.55	475.00	POINT (735.724 1097.738)	1.00	S	

	X	Y
0	693.71	442.26
1	793.70	453.71
2	885.77	480.50
3	682.87	1059.74
4	862.14	1206.37
5	216.17	405.97
6	364.75	403.29
7	415.17	976.19
8	604.92	1034.89
9	735.72	1097.74

Plotting the Orientations

[21]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1187)
plt.ylim(0, 1400)
```

[21]:
```
(0.0, 1400.0)
```



7.27.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[22]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[23]:
```
geo_model = gp.create_model('Model27')
geo_model
```

[23]:
```
Model27  2022-04-12 08:07
```

Initiate Data

[24]:
```
gp.init_data(geo_model, [0, 1187, 0, 1400, 0, 600], [100, 100, 100],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[24]:
```
Model27  2022-04-12 08:07
```

Model Surfaces

[25]:
```
geo_model.surfaces
```

[25]:

	surface	series	order_surfaces	color	id
0	S	Default series	1	#015482	1
1	R	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[26]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata1': ('S'),
                          'Strata2': ('R')},
                         remove_unused_series=True)
geo_model.add_surfaces('C')
```

[26]:

	surface	series	order_surfaces	color	id
0	S	Strata1	1	#015482	1
1	R	Strata2	1	#9f0052	2
2	C	Strata2	2	#ffbe00	3

Showing the Number of Data Points

[27]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[27]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	S	Strata1	1	#015482	1	78	5
1	R	Strata2	1	#9f0052	2	75	5
2	C	Strata2	2	#ffbe00	3	0	0

Loading Digital Elevation Model

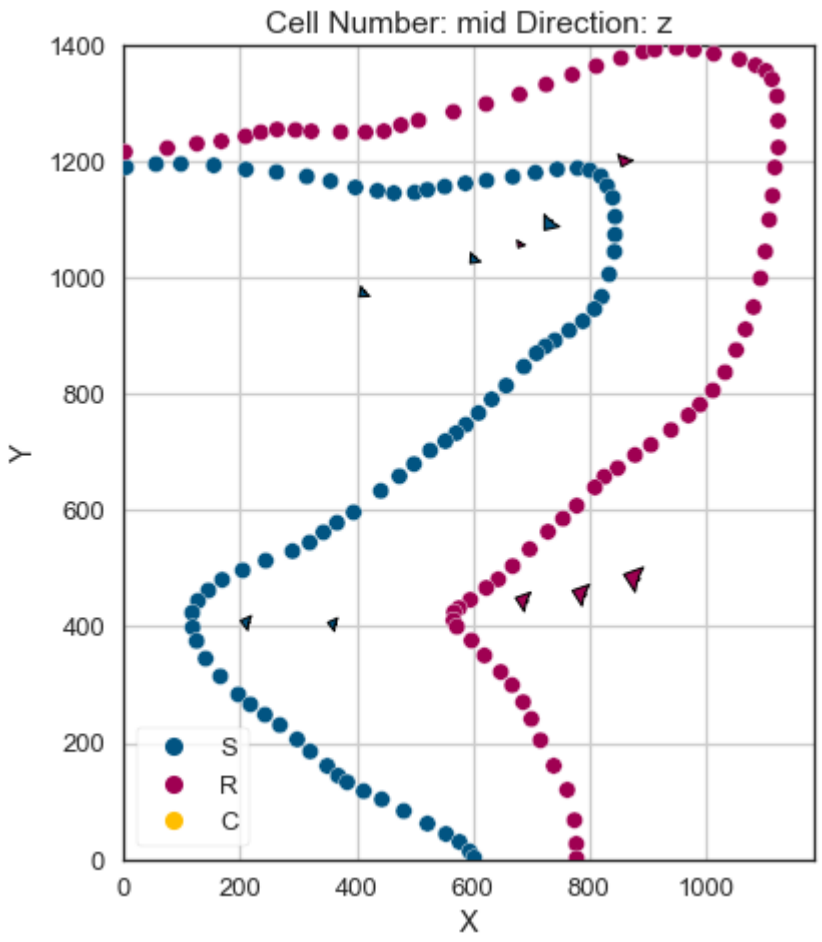
[28]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster27.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

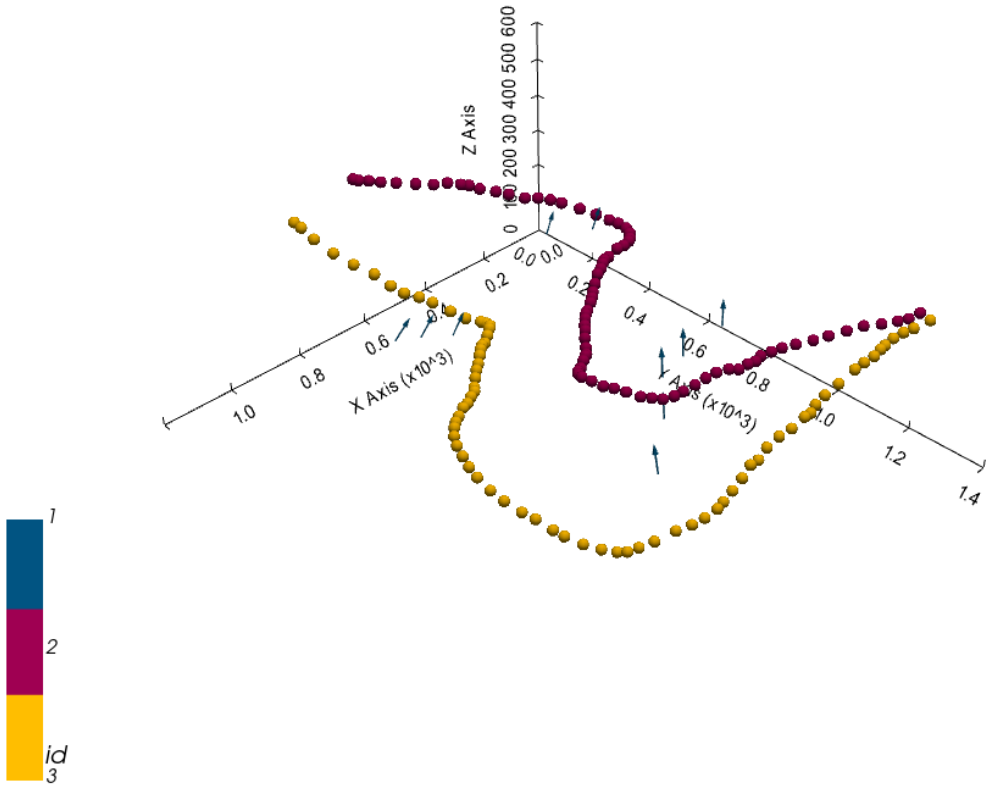
[28]:
```
Grid Object. Values:
array([[   5.935     ,    7.        ,    3.        ],
       [   5.935     ,    7.        ,    9.        ],
       [   5.935     ,    7.        ,   15.        ],
       ...,
       [1182.01260504, 1375.        ,  413.17556763],
       [1182.01260504, 1385.        ,  413.9055481 ],
       [1182.01260504, 1395.        ,  414.4772644 ]])
```

Plotting Input Data

[29]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[30]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[30]:
```
<gempy.plot.vista.GemPyToVista at 0x145e8c08490>
```

Setting the Interpolator

[31]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range            1931.05
```

(continues on next page)

(continued from previous page)

```
$C_o$           88784.98
drift equations   [3, 3]
```

[31]:
```
<gempy.core.interpolator.InterpolatorModel at 0x145e1456c70>
```

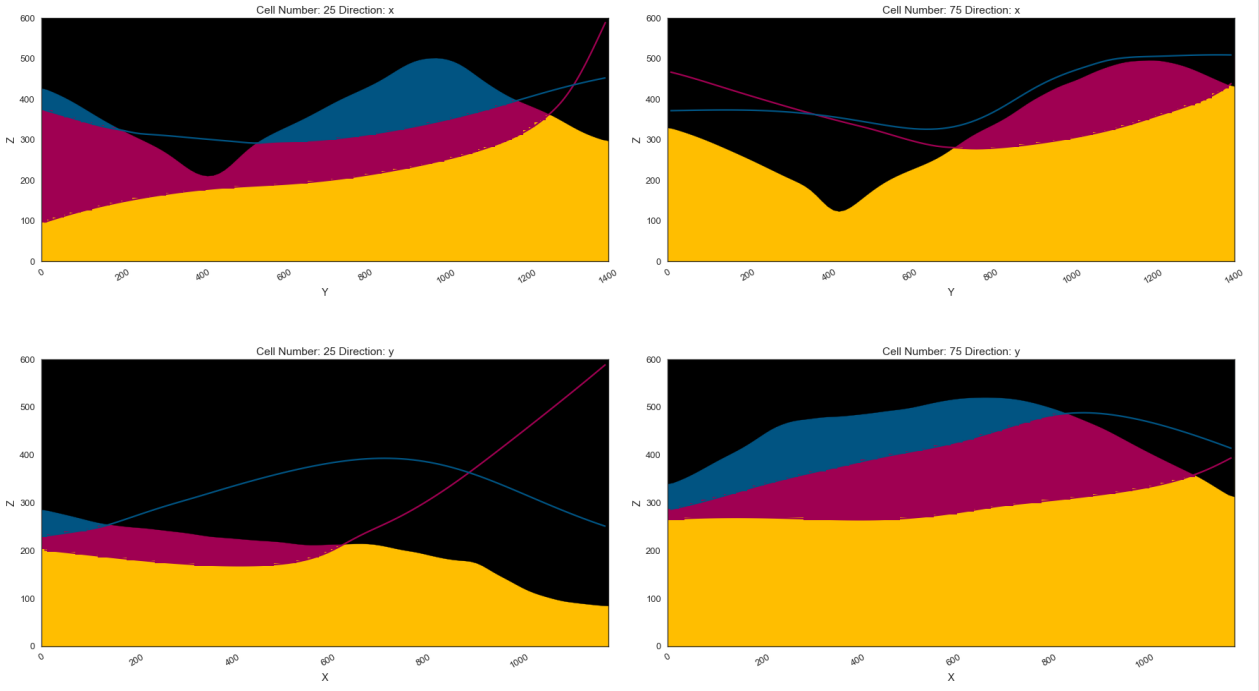
Computing Model

[32]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

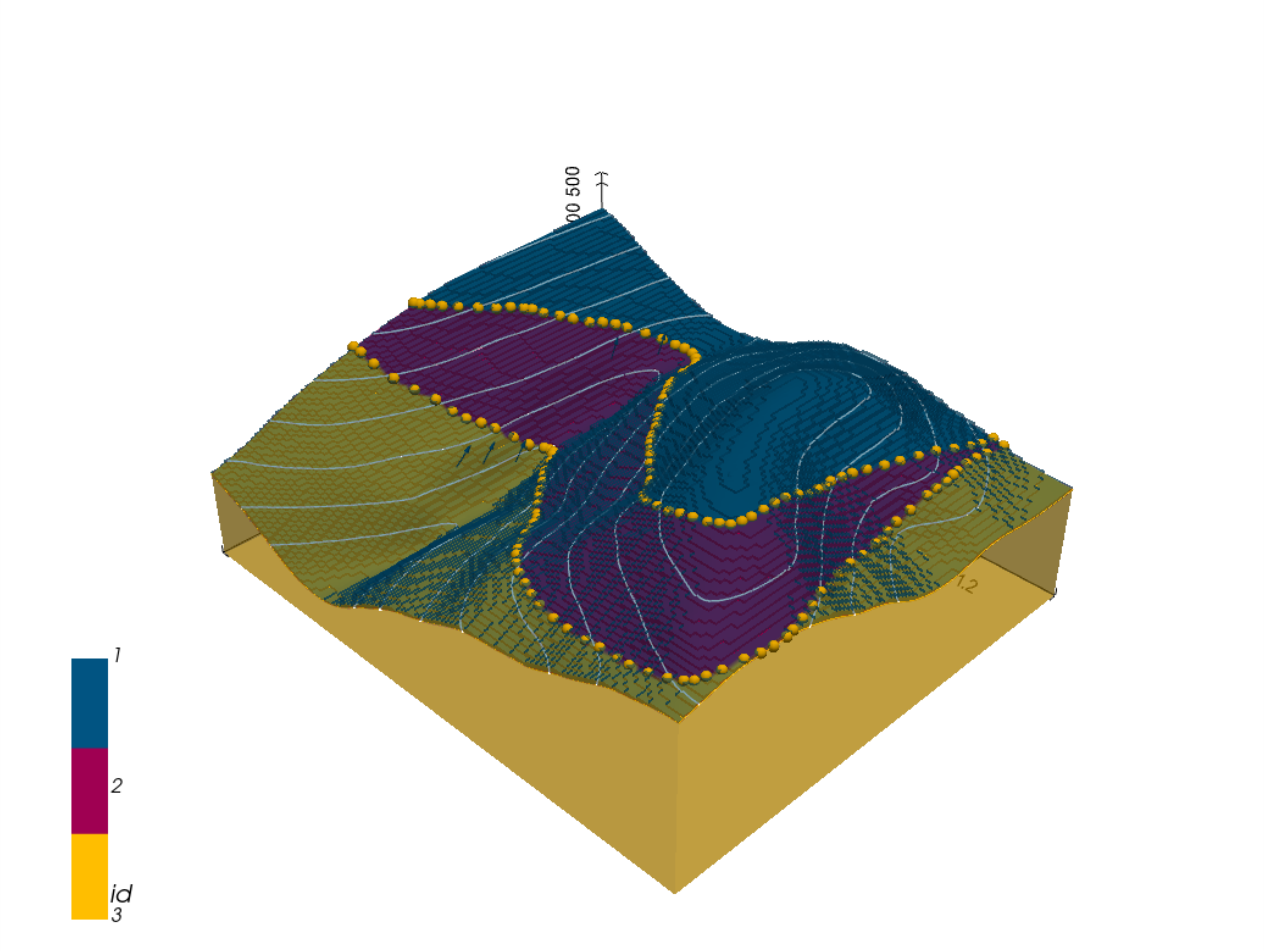
Plotting Cross Sections

[33]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[33]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1458f9bab20>
```



[34]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



[ ]:

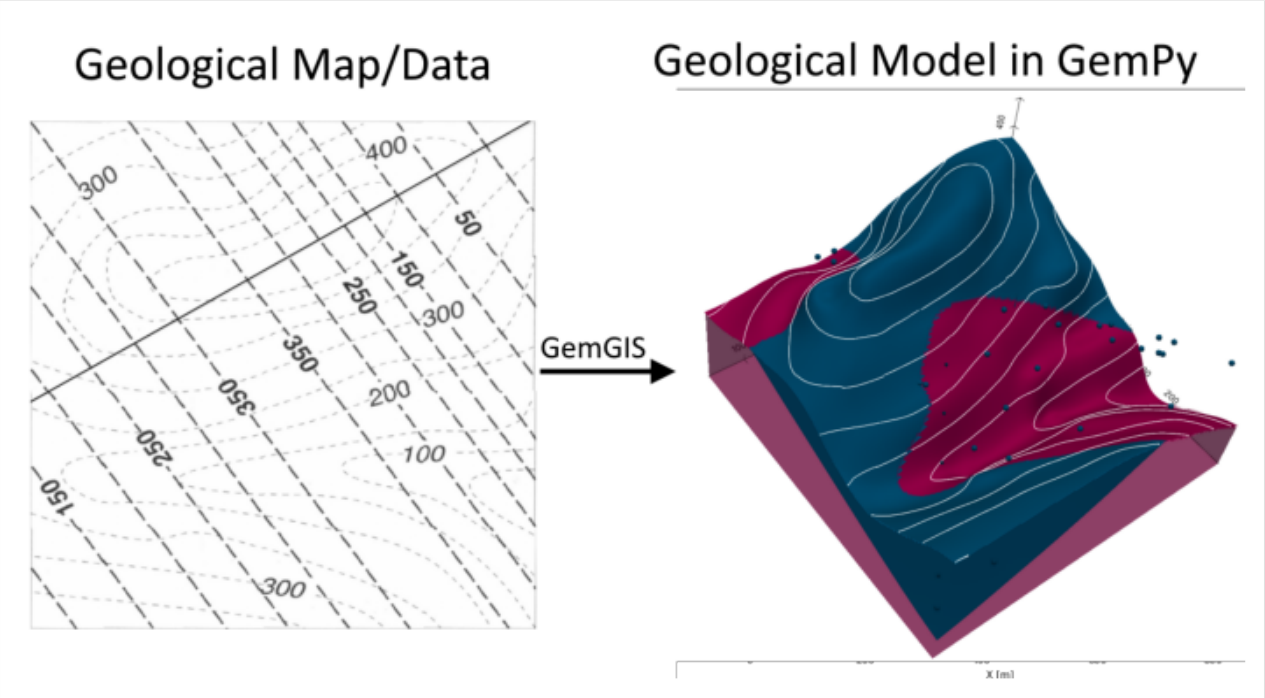
7.28 Example 28 - Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 3954 m wide (W-E extent) and 2738 m high (N-S extent). The model represents folded layers that dip to the northeast or southwest, respectively.

The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and will be used to calculate orientations for the `GemPy` model. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: An Introduction to Geological Structures and Maps by G.M. Bennison

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example28.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.28.1 Licensing

7.28.2 Import GemGIS

If you have installed `GemGIS` via pip, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.28.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded form an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

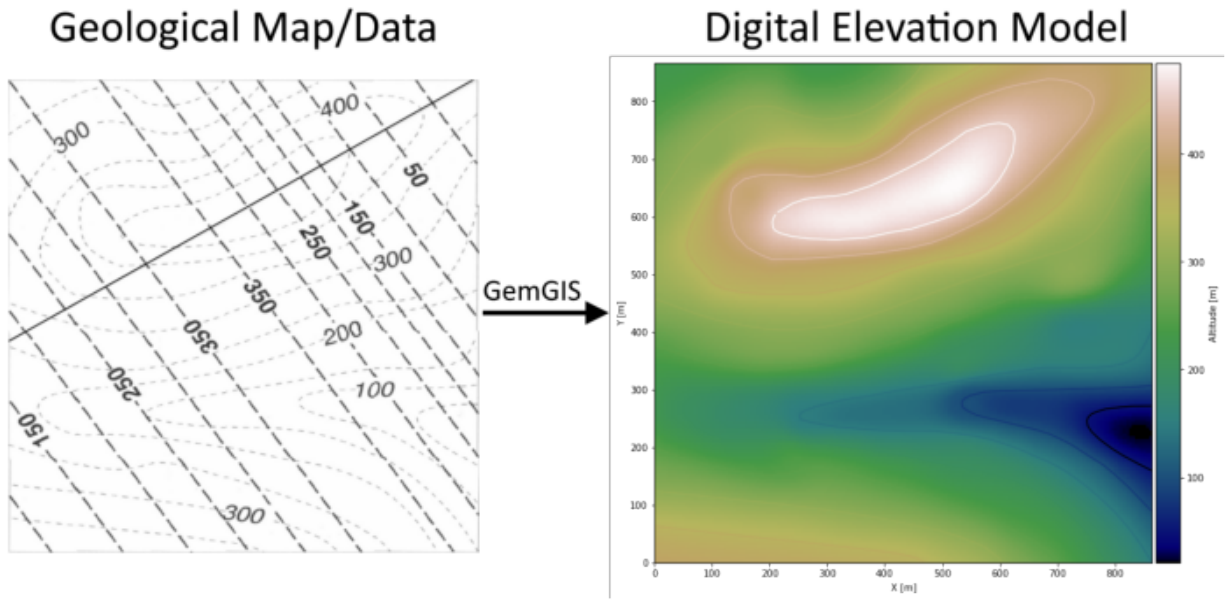
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example28/'
gg.download_gemgis_data.download_tutorial_data(filename="example28_folded_layers.zip",␣
↪dirpath=file_path)
```

7.28.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
img = mpimg.imread('../images/dem_example28.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo28.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	350	LINESTRING (1.385 62.877, 136.138 47.987, 292...
1	None	300	LINESTRING (1.571 124.111, 129.996 110.524, 22...
2	None	250	LINESTRING (0.082 198.374, 59.641 178.273, 118...
3	None	50	LINESTRING (863.690 271.706, 829.815 271.334, ...
4	None	100	LINESTRING (864.062 307.070, 828.326 304.092, ...

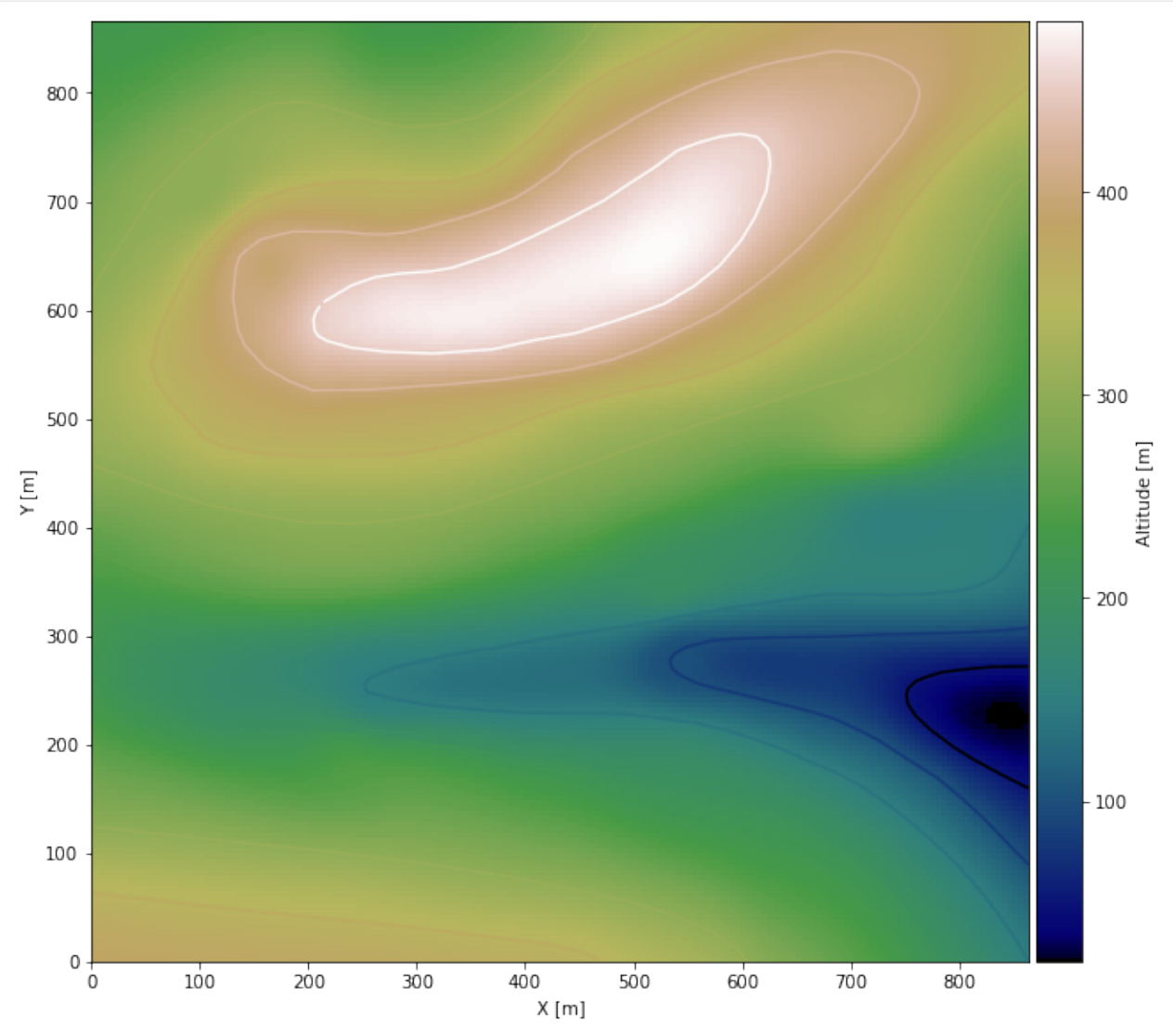
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fix, ax = plt.subplots(1, figsize=(10,10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0,865,0,867], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
```

[8]:
```
Text(67.3456168134622, 0.5, 'Y [m]')
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as as raster is already provided with the example data.

[9]:
```
gg.raster.save_as_tiff(raster=topo_raster, path=file_path + 'raster28.tif', extent=[0,
↪865,0,867], crs='EPSG:4326', overwrite_file=True)
```

```
Raster successfully saved
```

Opening Raster

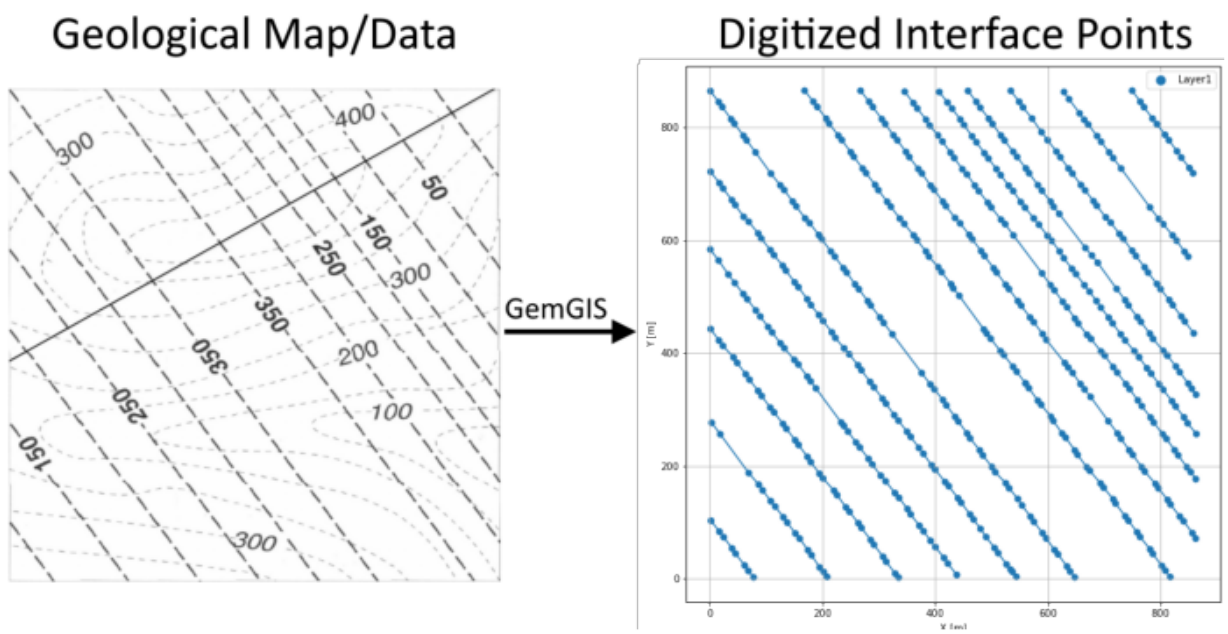
The previously computed and saved raster can now be opened using rasterio.

[10]:
```
topo_raster = rasterio.open(file_path + 'raster28.tif')
```

7.28.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[11]:
```
img = mpimg.imread('../images/interfaces_example28.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[12]:
```
interfaces = gpd.read_file(file_path + 'interfaces28.shp')
interfaces.head()
```

[12]:

	id	formation	Z	geometry
0	None	Layer1	0	LINESTRING (750.000 865.441, 764.514 845.629, ...
1	None	Layer1	50	LINESTRING (628.941 863.195, 637.925 850.294, ...
2	None	Layer1	100	LINESTRING (534.604 865.153, 549.463 845.687, ...
3	None	Layer1	150	LINESTRING (458.590 864.955, 473.370 845.597, ...
4	None	Layer1	100	LINESTRING (2.577 102.527, 17.435 83.097, 25.0...

Extracting XY coordinates from Digital Elevation Model

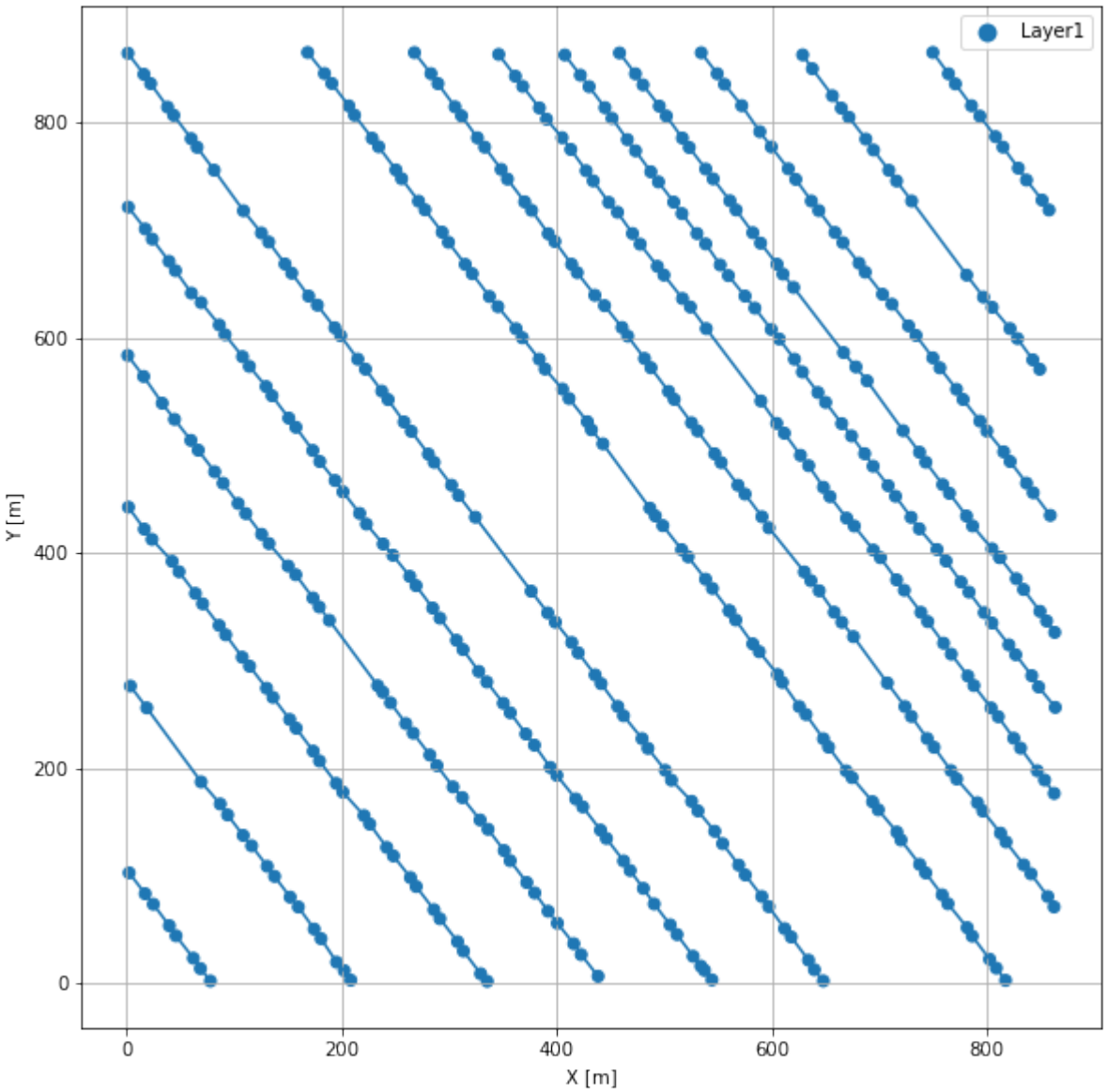
[13]:
```
interfaces_coords = gg.vector.extract_xy(gdf=interfaces)
interfaces_coords = interfaces_coords.sort_values(by='formation', ascending=False)
interfaces_coords.head()
```

[13]:
```
    formation      Z                 geometry      X      Y
0      Layer1   0.00  POINT (750.000 865.441) 750.00 865.44
304    Layer1 350.00  POINT (345.483 629.092) 345.48 629.09
332    Layer1 350.00  POINT (669.435 197.325) 669.43 197.33
331    Layer1 350.00  POINT (653.053 219.421) 653.05 219.42
330    Layer1 350.00  POINT (648.227 227.168) 648.23 227.17
```

Plotting the Interface Points

[14]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
```

[14]:
```
Text(67.45106220501688, 0.5, 'Y [m]')
```



7.28.6 Orientations from Strike Lines

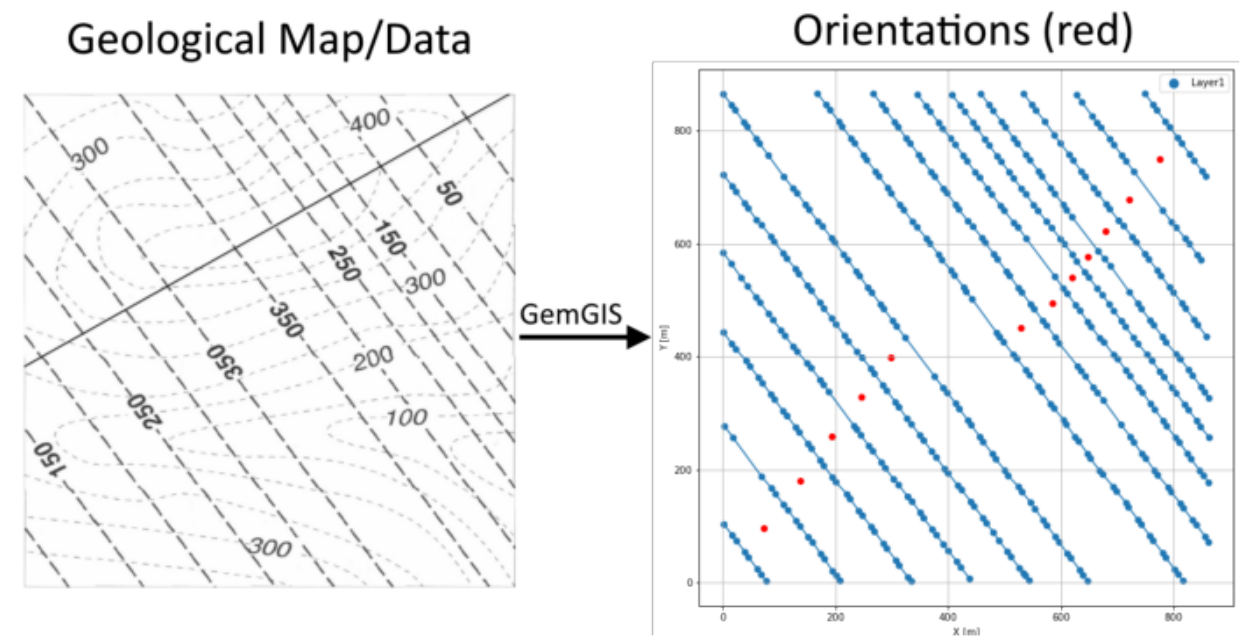
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[15]:
```
img = mpimg.imread('../images/orientations_example28.png')
plt.figure(figsize=(10, 10))
```

(continues on next page)

(continued from previous page)

```
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[16]:
```
strikes = gpd.read_file(file_path + 'strikes28.shp')
strikes
```

[16]:

	id	Z	formation	geometry
0	1	100	Layer1a	LINESTRING (2.577 102.527, 78.137 1.951)
1	2	150	Layer1a	LINESTRING (3.720 276.250, 208.683 2.713)
2	3	200	Layer1a	LINESTRING (1.942 442.607, 335.546 1.697)
3	4	250	Layer1a	LINESTRING (1.307 583.693, 438.662 6.395)
4	5	300	Layer1a	LINESTRING (1.815 721.604, 544.572 2.967)
5	6	350	Layer1a	LINESTRING (1.466 864.659, 648.100 2.014)
6	8	350	Layer1b	LINESTRING (168.585 865.421, 817.632 2.522)
7	7	300	Layer1b	LINESTRING (267.892 865.294, 862.714 70.843)
8	6	250	Layer1b	LINESTRING (346.245 863.643, 862.841 176.372)
9	5	200	Layer1b	LINESTRING (407.708 863.135, 863.857 256.376)
10	4	150	Layer1b	LINESTRING (458.590 864.955, 863.317 326.157)
11	3	100	Layer1b	LINESTRING (534.604 865.153, 859.196 434.936)
12	2	50	Layer1b	LINESTRING (628.941 863.195, 862.767 552.331)
13	1	0	Layer1b	LINESTRING (750.000 865.441, 858.044 718.925)

Calculate Orientations for each formation

[17]:
```
orientations_layer1a = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Layer1a'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_layer1a
```

[17]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 25.48   233.15 125.00    POINT (73.279 95.860)      1.00   Layer1a  73.28
1 26.81   232.96 175.00  POINT (137.473 180.816)      1.00   Layer1a 137.47
2 30.56   232.87 225.00  POINT (194.364 258.598)      1.00   Layer1a 194.36
3 31.23   232.90 275.00  POINT (246.589 328.665)      1.00   Layer1a 246.59
4 31.29   233.06 325.00  POINT (298.988 397.811)      1.00   Layer1a 298.99

       Y
0  95.86
1 180.82
2 258.60
3 328.66
4 397.81
```

[18]:
```
orientations_layer1b = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation']=='Layer1b'].sort_values(by='Z', ascending=True).
↪reset_index())
orientations_layer1b
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 27.42    53.15  25.00  POINT (774.938 749.973)      1.00   Layer1b 774.94
1 34.21    52.99  75.00  POINT (721.377 678.904)      1.00   Layer1b 721.38
2 39.39    53.04 125.00  POINT (678.927 622.800)      1.00   Layer1b 678.93
3 50.31    53.07 175.00  POINT (648.368 577.656)      1.00   Layer1b 648.37
4 45.68    53.07 225.00  POINT (620.163 539.881)      1.00   Layer1b 620.16
5 39.01    53.13 275.00  POINT (584.923 494.038)      1.00   Layer1b 584.92
6 32.97    53.11 325.00  POINT (529.206 451.020)      1.00   Layer1b 529.21

       Y
0 749.97
1 678.90
2 622.80
3 577.66
4 539.88
5 494.04
6 451.02
```

Merging Orientations

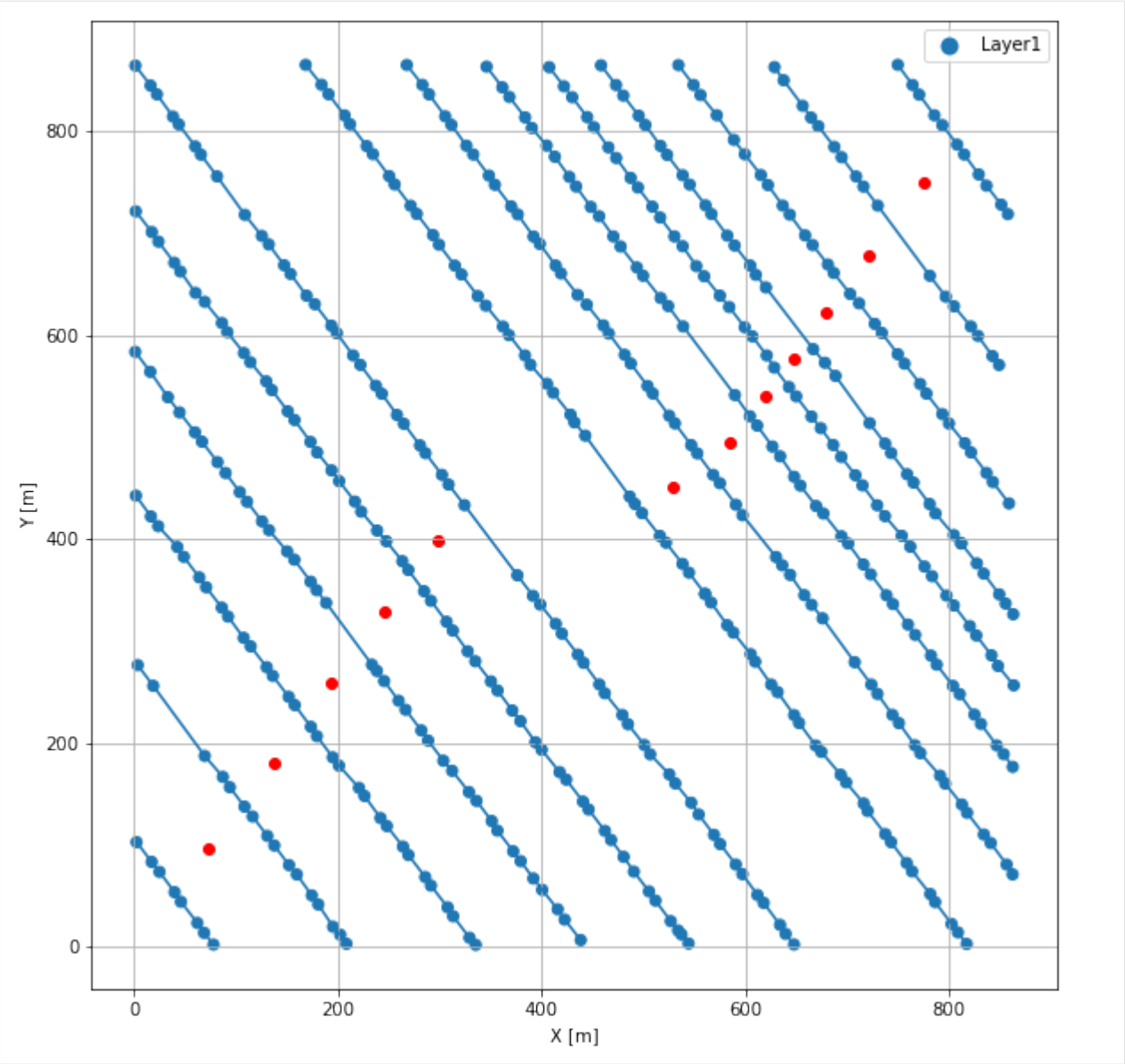
[19]:
```
import pandas as pd
orientations = pd.concat([orientations_layer1a, orientations_layer1b]).reset_index()
orientations['formation'] = 'Layer1'
orientations
```

[19]:
```
    index   dip  azimuth      Z                 geometry  polarity formation  \
0       0 25.48   233.15 125.00    POINT (73.279 95.860)      1.00    Layer1
1       1 26.81   232.96 175.00  POINT (137.473 180.816)      1.00    Layer1
2       2 30.56   232.87 225.00  POINT (194.364 258.598)      1.00    Layer1
3       3 31.23   232.90 275.00  POINT (246.589 328.665)      1.00    Layer1
4       4 31.29   233.06 325.00  POINT (298.988 397.811)      1.00    Layer1
5       0 27.42    53.15  25.00  POINT (774.938 749.973)      1.00    Layer1
6       1 34.21    52.99  75.00  POINT (721.377 678.904)      1.00    Layer1
7       2 39.39    53.04 125.00  POINT (678.927 622.800)      1.00    Layer1
8       3 50.31    53.07 175.00  POINT (648.368 577.656)      1.00    Layer1
9       4 45.68    53.07 225.00  POINT (620.163 539.881)      1.00    Layer1
10      5 39.01    53.13 275.00  POINT (584.923 494.038)      1.00    Layer1
11      6 32.97    53.11 325.00  POINT (529.206 451.020)      1.00    Layer1

        X      Y
0   73.28  95.86
1  137.47 180.82
2  194.36 258.60
3  246.59 328.66
4  298.99 397.81
5  774.94 749.97
6  721.38 678.90
7  678.93 622.80
8  648.37 577.66
9  620.16 539.88
10 584.92 494.04
11 529.21 451.02
```

Plotting the Orientations

[20]:
```
fig, ax = plt.subplots(1, figsize=(10,10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
```

[20]:
```
Text(67.45106220501688, 0.5, 'Y [m]')
```



7.28.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[21]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[22]:
```
geo_model = gp.create_model('Model28')
geo_model
```

[22]:
```
Model28  2022-04-13 08:25
```

Initiate Data

[23]:
```
gp.init_data(geo_model, [0,865,0,867,0,500], [50,50,50],
             surface_points_df = interfaces_coords[interfaces_coords['Z']!=0].
↪sample(n=75, random_state=1),
             orientations_df = orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[23]:
```
Model28  2022-04-13 08:25
```

Model Surfaces

[24]:
```
geo_model.surfaces
```

[24]:

	surface	series	order_surfaces	color	id
0	Layer1	Default series	1	#015482	1

Mapping the Stack to Surfaces

[25]:
```
gp.map_stack_to_surfaces(geo_model,
                         {
                          'Strata1': ('Layer1'),
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('Basement')
```

[25]:

	surface	series	order_surfaces	color	id
0	Layer1	Strata1	1	#015482	1
1	Basement	Strata1	2	#9f0052	2

Showing the Number of Data Points

[26]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[26]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	Layer1	Strata1	1	#015482	1	75	12
1	Basement	Strata1	2	#9f0052	2	0	0

Loading Digital Elevation Model

[27]:
```
geo_model.set_topography(
    source='gdal', filepath=file_path + 'raster28.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

[27]:
```
Grid Object. Values:
array([[  8.65      ,   8.67      ,   5.        ],
       [  8.65      ,   8.67      ,  15.        ],
       [  8.65      ,   8.67      ,  25.        ],
       ...,
       [862.5       , 854.54310345, 363.79455566],
       [862.5       , 859.52586207, 364.32107544],
       [862.5       , 864.50862069, 364.75622559]])
```

Defining Custom Section

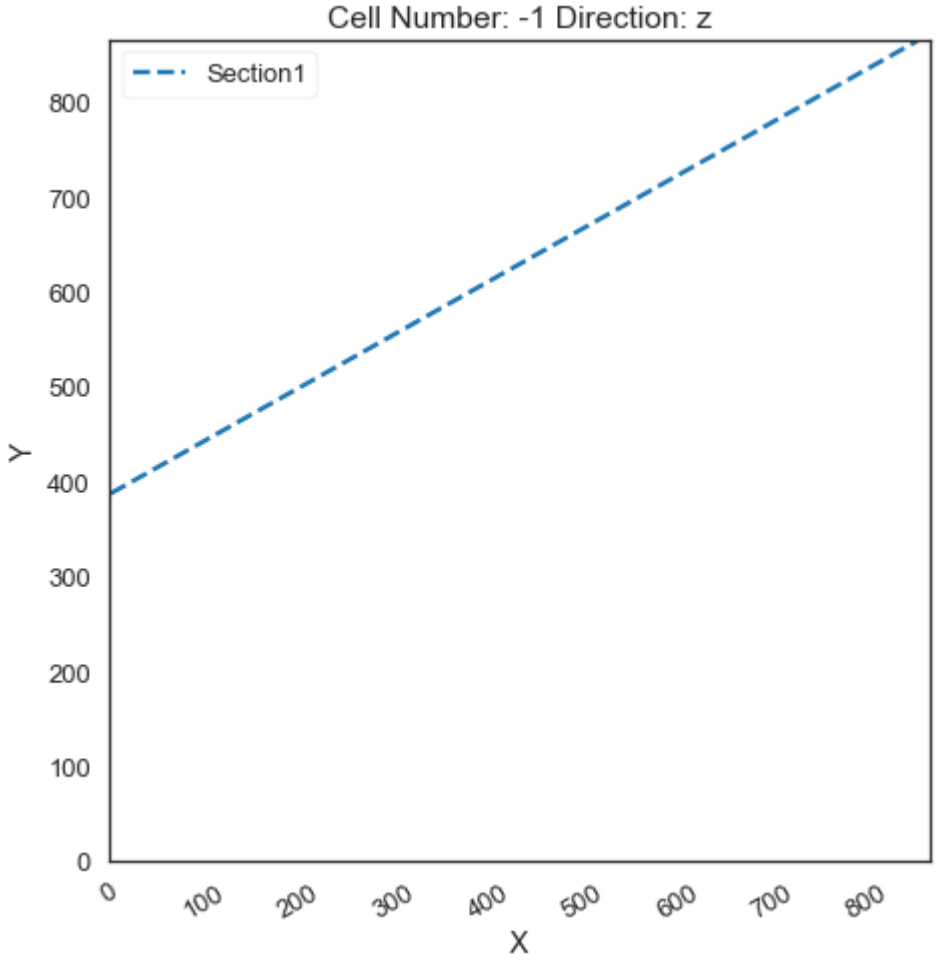
[28]:
```
custom_section = gpd.read_file(file_path + 'customsections28.shp')
custom_section_dict = gg.utils.to_section_dict(custom_section, section_column='section')
geo_model.set_section_grid(custom_section_dict)
```

```
Active grids: ['regular' 'topography' 'sections']
```

[28]:
```
                                          start                                       stop␣
↪resolution   dist
Section1  [1.593210345965275, 388.4453900184363]  [852.4277356922321, 865.9286460336549]␣
↪ [100, 80] 975.66
```

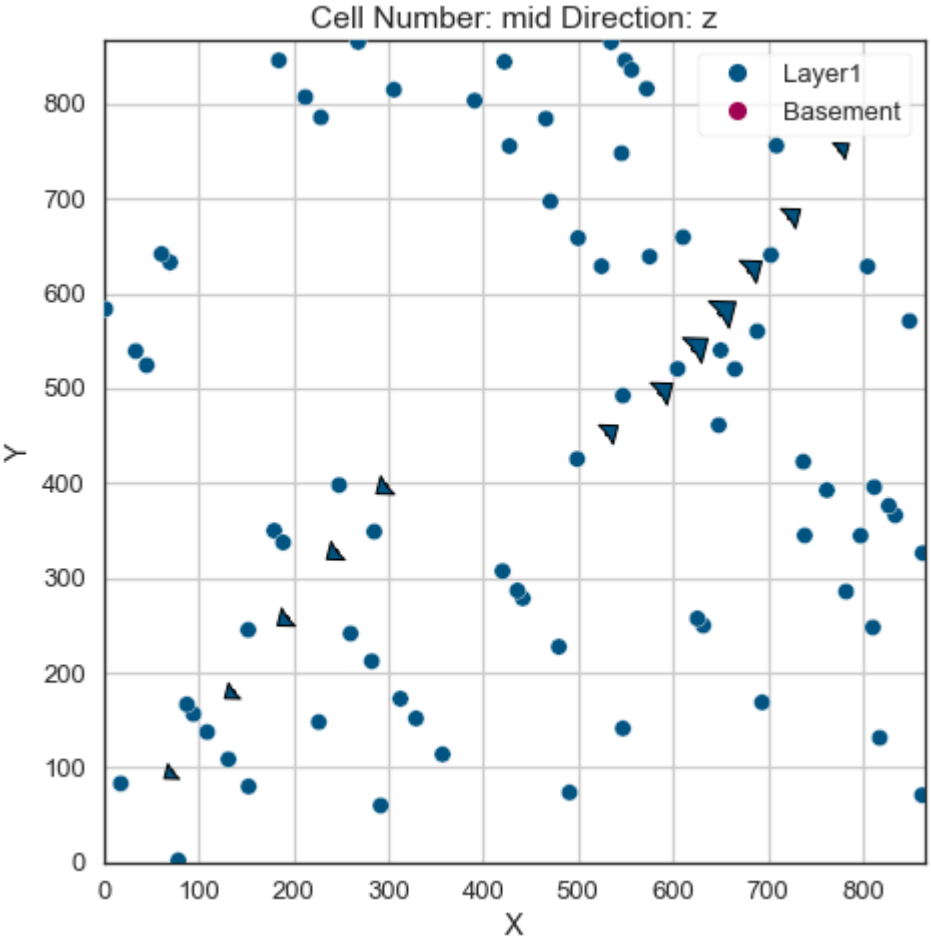
[29]:
```
gp.plot.plot_section_traces(geo_model)
```

[29]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1ebe7a01f10>
```

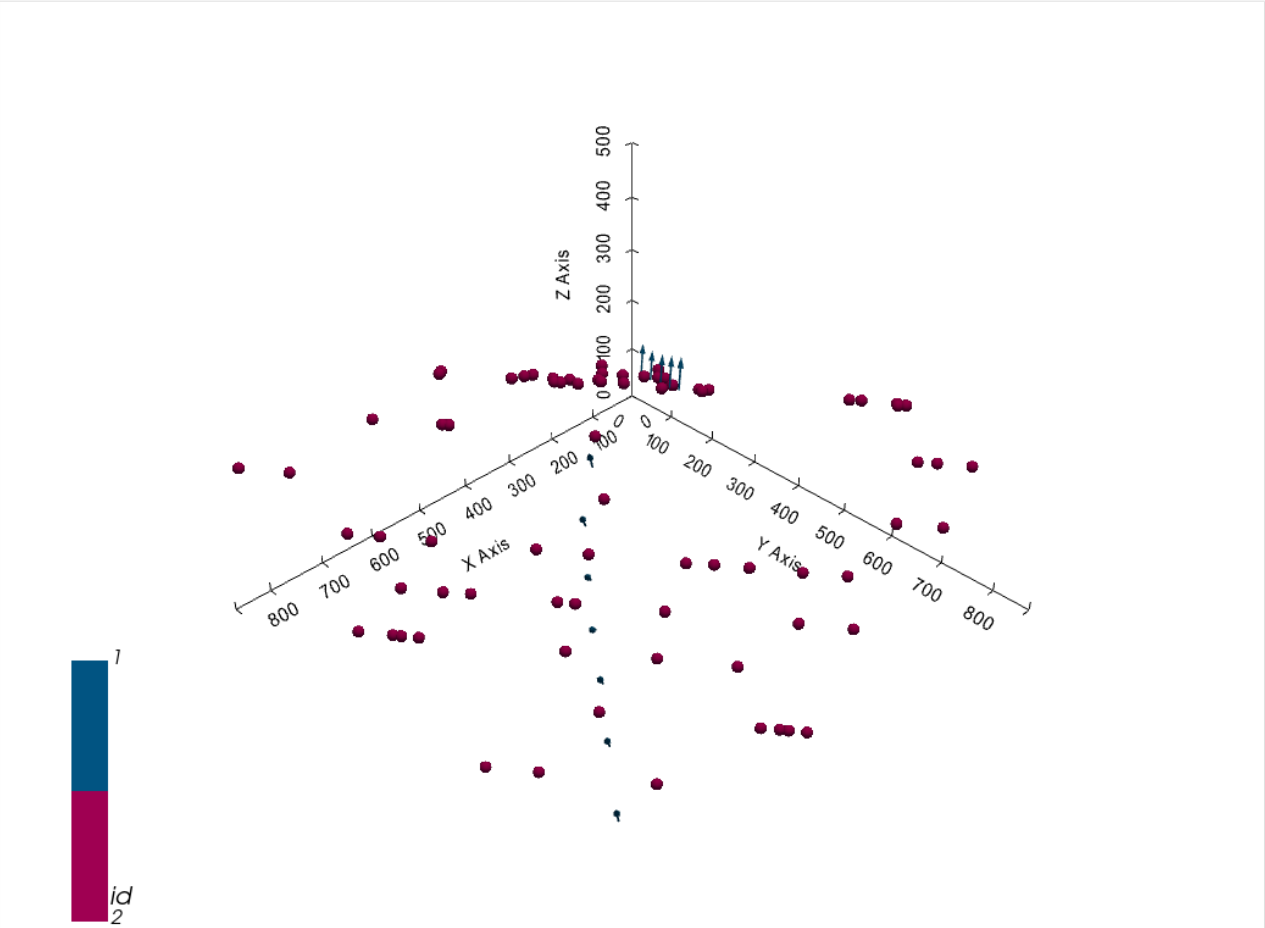


Plotting Input Data

[30]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[31]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[31]:
```
<gempy.plot.vista.GemPyToVista at 0x1ebe70f09d0>
```

Setting the Interpolator

[32]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging = False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range            1322.84
$C_o$           41664.62
drift equations      [3]
```

[32]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1ebe0d78af0>
```

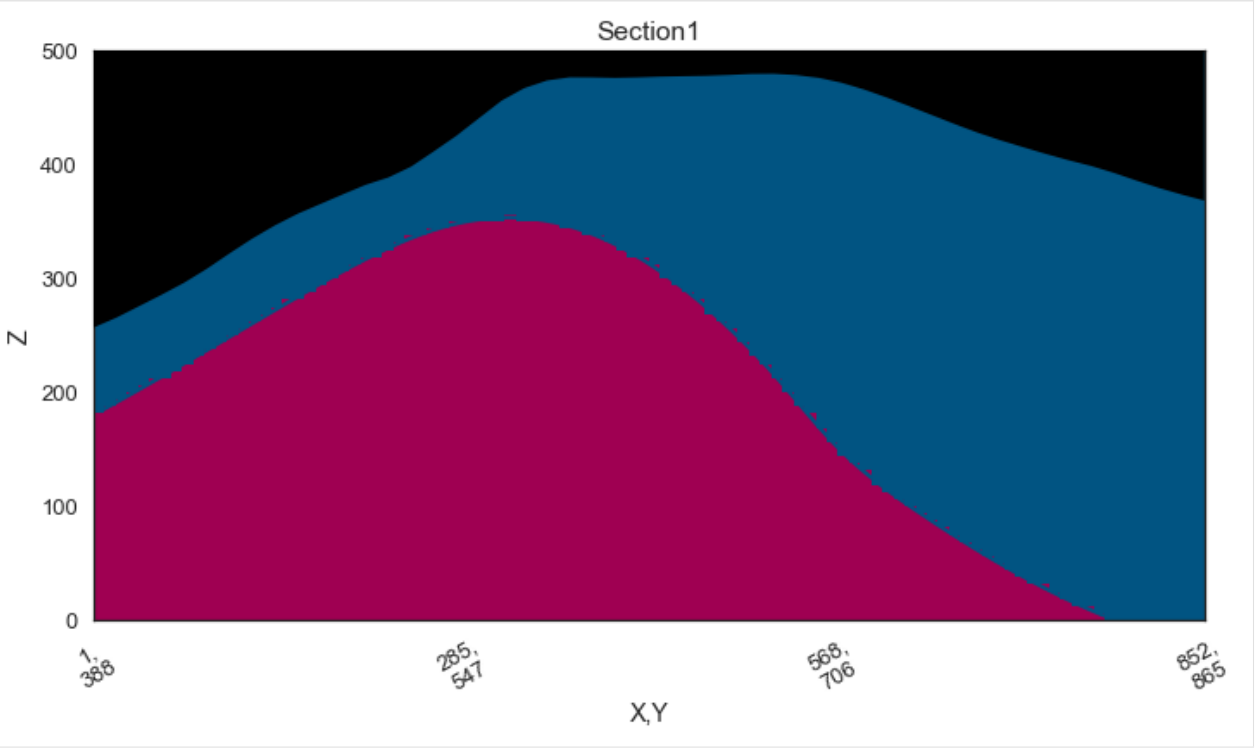
Computing Model

[33]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

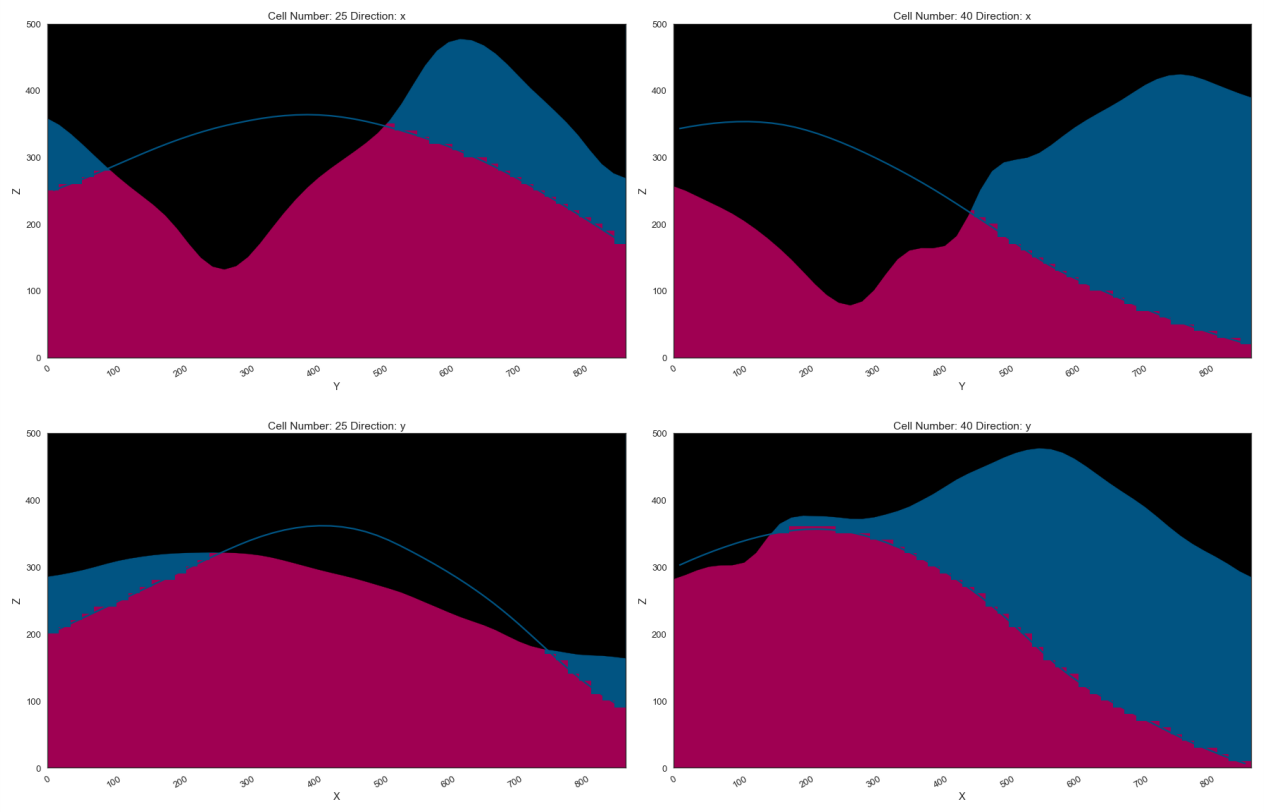
[34]:
```
gp.plot_2d(geo_model, section_names=['Section1'], show_topography=True, show_data=False)
```

[34]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1ebe71ad130>
```



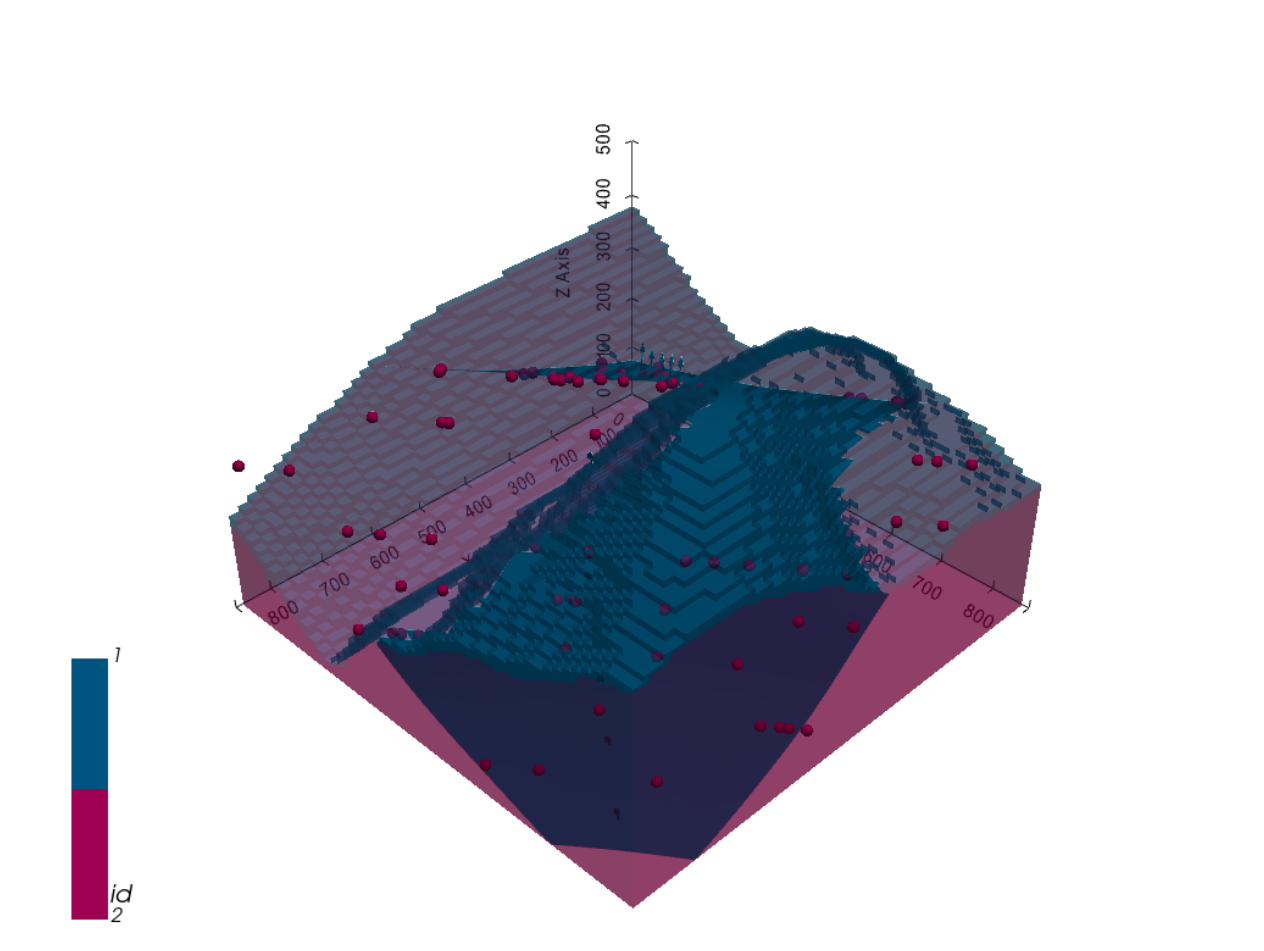
[35]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25,40,25,40], show_
↪topography=True, show_data=False)
```

[35]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1ebe72b9460>
```



Plotting 3D Model

[36]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=False,
                 plotter_type='basic', notebook=True, show_lith=True)
```



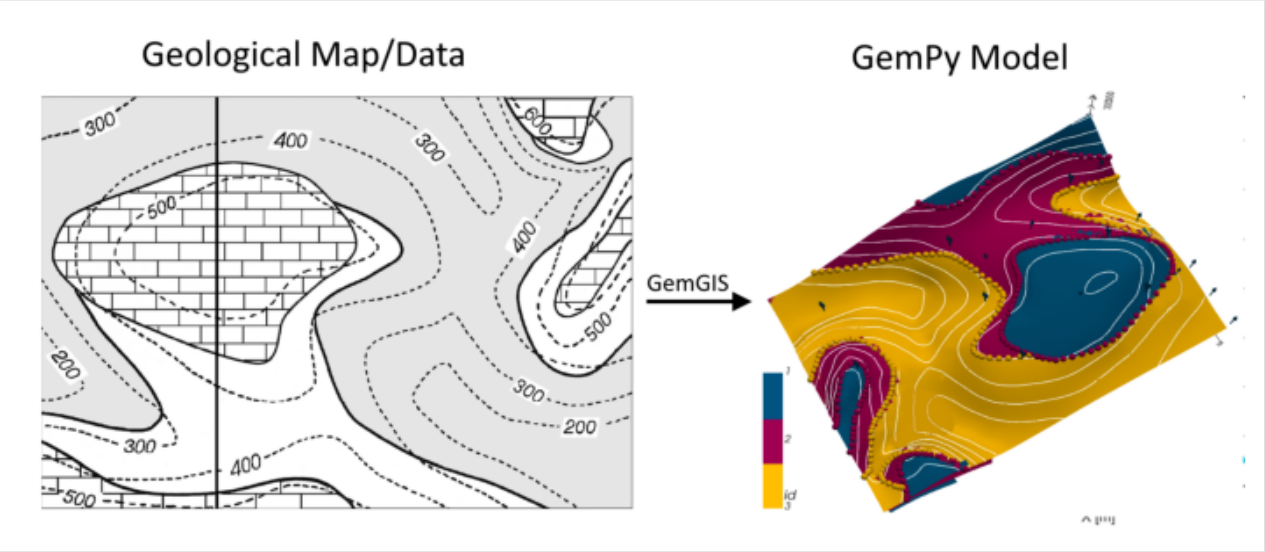
[ ]:

7.29 Example 29 - Unconformable Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 1134 m wide (W-E extent) and 788 m high (N-S extent). The vertical model extents varies between 0 m and 600 m. The model represents two unconformable dipping planar stratigraphic units (blue and red) above an unspecified basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and were used to calculate orientations for the `GemPy` model. These will be loaded into the model directly. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example29.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.29.1 Licensing

7.29.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.29.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

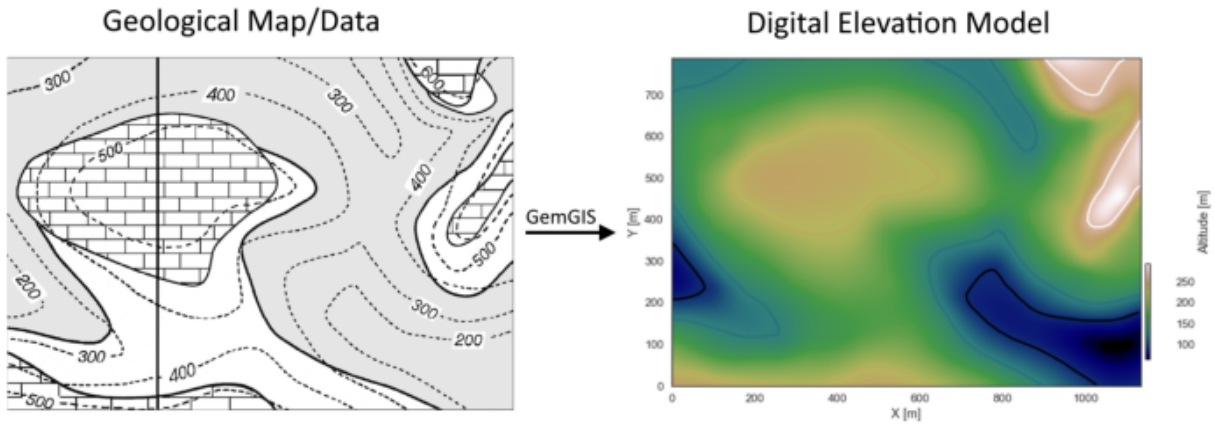
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = '../data/example29/'
gg.download_gemgis_data.download_tutorial_data(filename="example29_unconformable_layers.
↪zip", dirpath=file_path)
```

7.29.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
img = mpimg.imread('../images/dem_example29.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo29.shp')
topo['Z'] = topo['Z']*0.444
topo.head()
```

[6]:
```
     id      Z                                           geometry
0  None 222.00  LINESTRING (2.578 40.099, 33.282 34.251, 75.92...
1  None 177.60  LINESTRING (4.771 101.995, 41.324 85.912, 71.5...
2  None  88.80  LINESTRING (1.603 344.217, 17.199 324.235, 33...
3  None 133.20  LINESTRING (3.065 159.017, 29.140 156.580, 57...
4  None  88.80  LINESTRING (1026.294 2.328, 1000.220 20.117, 9...
```

Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

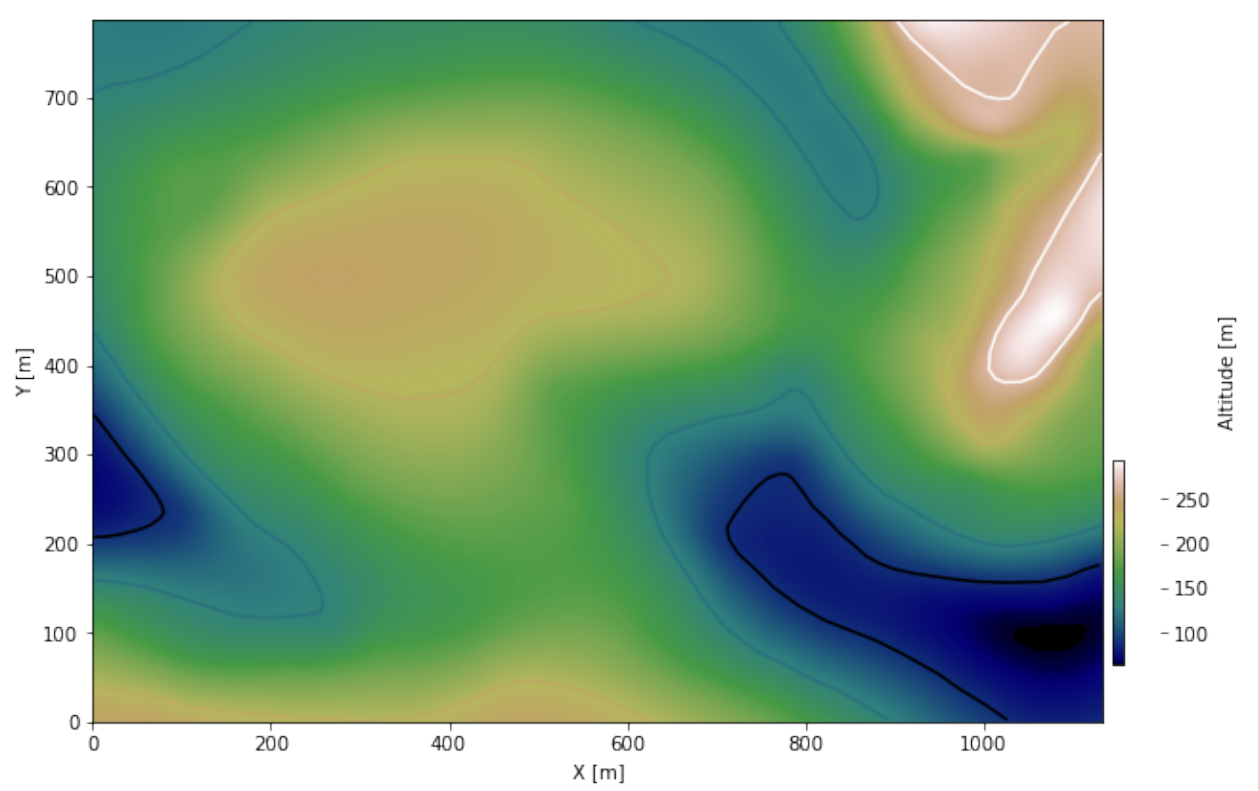
[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
```

(continues on next page)

(continued from previous page)

```
im = ax.imshow(topo_raster, origin='lower', extent=[0, 1134, 0, 788], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
plt.xlim(0, 1134)
plt.ylim(0, 788)
```

[8]: (0.0, 788.0)



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster29.tif', extent = [0, 1134, 0, 788], crs ='EPSG : 4326', overwrite_f ile = True)$

Opening Raster

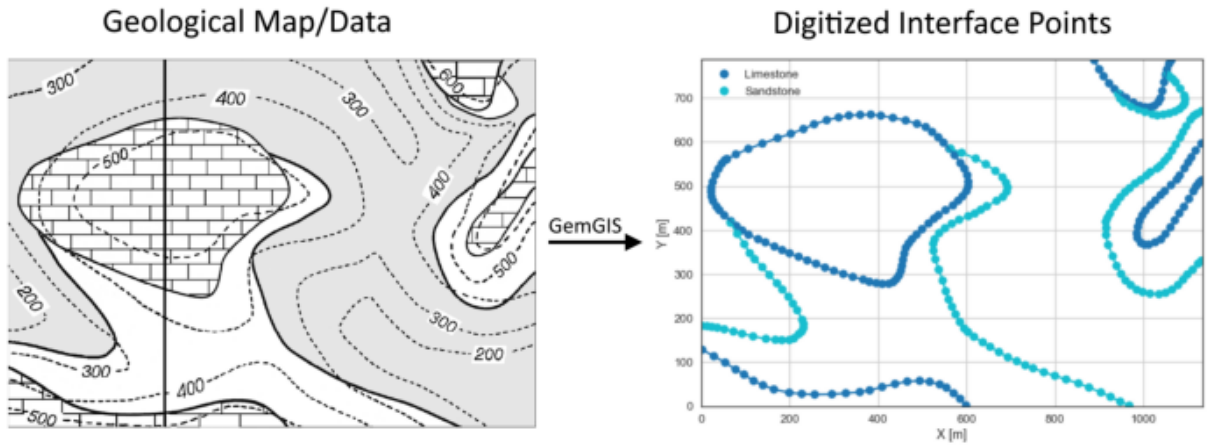
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster29.tif')
```

7.29.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../images/interfaces_example29.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces29.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Limestone	LINESTRING (1.725 128.313, 28.287 113.692, 53...
1	None	Limestone	LINESTRING (58.016 561.339, 78.120 572.061, 10...
2	None	Sandstone	LINESTRING (65.692 416.834, 81.044 391.613, 97...
3	None	Sandstone	LINESTRING (563.660 585.098, 588.760 575.595, ...
4	None	Limestone	LINESTRING (888.978 785.650, 900.188 764.937, ...

Extracting Z coordinate from Digital Elevation Model

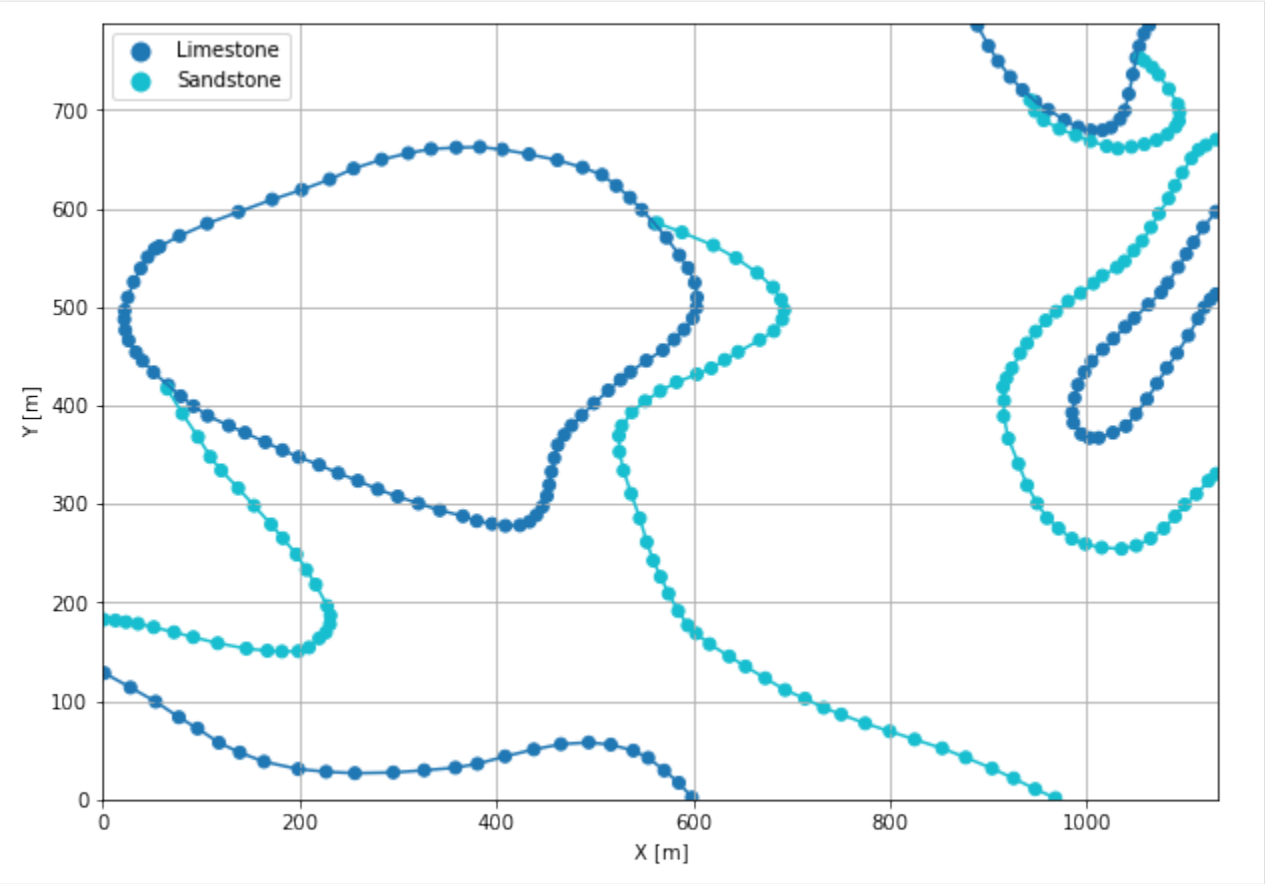
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords
```

[12]:
```
     formation                  geometry       X      Y      Z
0    Limestone     POINT (1.725 128.313)    1.73 128.31 161.10
1    Limestone    POINT (28.287 113.692)   28.29 113.69 166.12
2    Limestone     POINT (53.873 99.314)   53.87  99.31 167.68
3    Limestone     POINT (77.511 83.475)   77.51  83.47 170.08
4    Limestone     POINT (96.518 72.022)   96.52  72.02 171.47
..         ...                       ...     ...    ...    ...
300  Sandstone  POINT (1089.774 287.316) 1089.77 287.32 181.40
301  Sandstone  POINT (1099.521 299.257) 1099.52 299.26 184.35
302  Sandstone  POINT (1111.462 309.979) 1111.46 309.98 185.01
303  Sandstone  POINT (1123.646 323.382) 1123.65 323.38 185.75
304  Sandstone  POINT (1131.444 330.448) 1131.44 330.45 186.40

[305 rows x 5 columns]
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1134)
plt.ylim(0, 788)
```

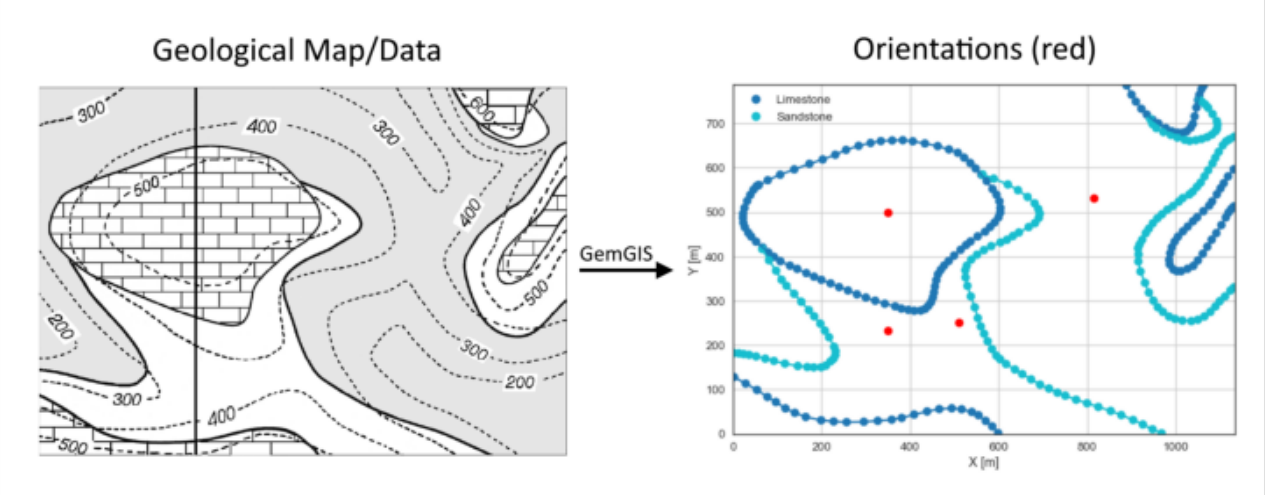
[13]:
```
(0.0, 788.0)
```



7.29.6 Orientations from Strike Lines

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
img = mpimg.imread('../images/orientations_example29.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes29.shp')
strikes['Z'] = strikes['Z']*0.444
strikes
```

[15]:
```
   id   formation      Z                                        geometry
0   2  Limestone1 222.00    LINESTRING (583.886 556.100, 583.642 468.861)
1   1  Limestone1 177.60    LINESTRING (113.820 587.535, 114.063 385.521)
2   1   Sandstone 133.20     LINESTRING (231.032 183.507, 721.080 98.461)
3   2   Sandstone 177.60    LINESTRING (1073.935 271.964, 16.346 453.265)
4   1  Limestone2 177.60     LINESTRING (114.063 385.521, 115.647 58.375)
5   2  Limestone2 222.00     LINESTRING (584.008 468.252, 583.033 21.823)
6   3  Limestone1 266.40  LINESTRING (1041.890 715.835, 1046.764 383.206)
```

Calculating Orientations for each formation

[16]:
```
orientations_limestone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Limestone1'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_limestone1
```

[16]:
```
   dip  azimuth      Z                 geometry  polarity   formation      X  \
0 5.40   269.97 199.80  POINT (348.853 499.504)      1.00  Limestone1 348.85
1 5.51   269.22 244.20  POINT (814.046 531.000)      1.00  Limestone1 814.05

       Y
0 499.50
1 531.00
```

[17]:
```
orientations_limestone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Limestone2'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_limestone2
```

[17]:
```
   dip  azimuth      Z                 geometry  polarity   formation      X  \
0 5.43   269.98 199.80  POINT (349.188 233.493)      1.00  Limestone2 349.19
```

(continues on next page)

(continued from previous page)

```
       Y
0 233.49
```

[18]:
```
orientations_sandstone = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Sandstone'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_sandstone
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity  formation      X  \
0 10.94   189.75 155.40  POINT (510.598 251.799)      1.00  Sandstone 510.60

       Y
0 251.80
```

Merging Orientations

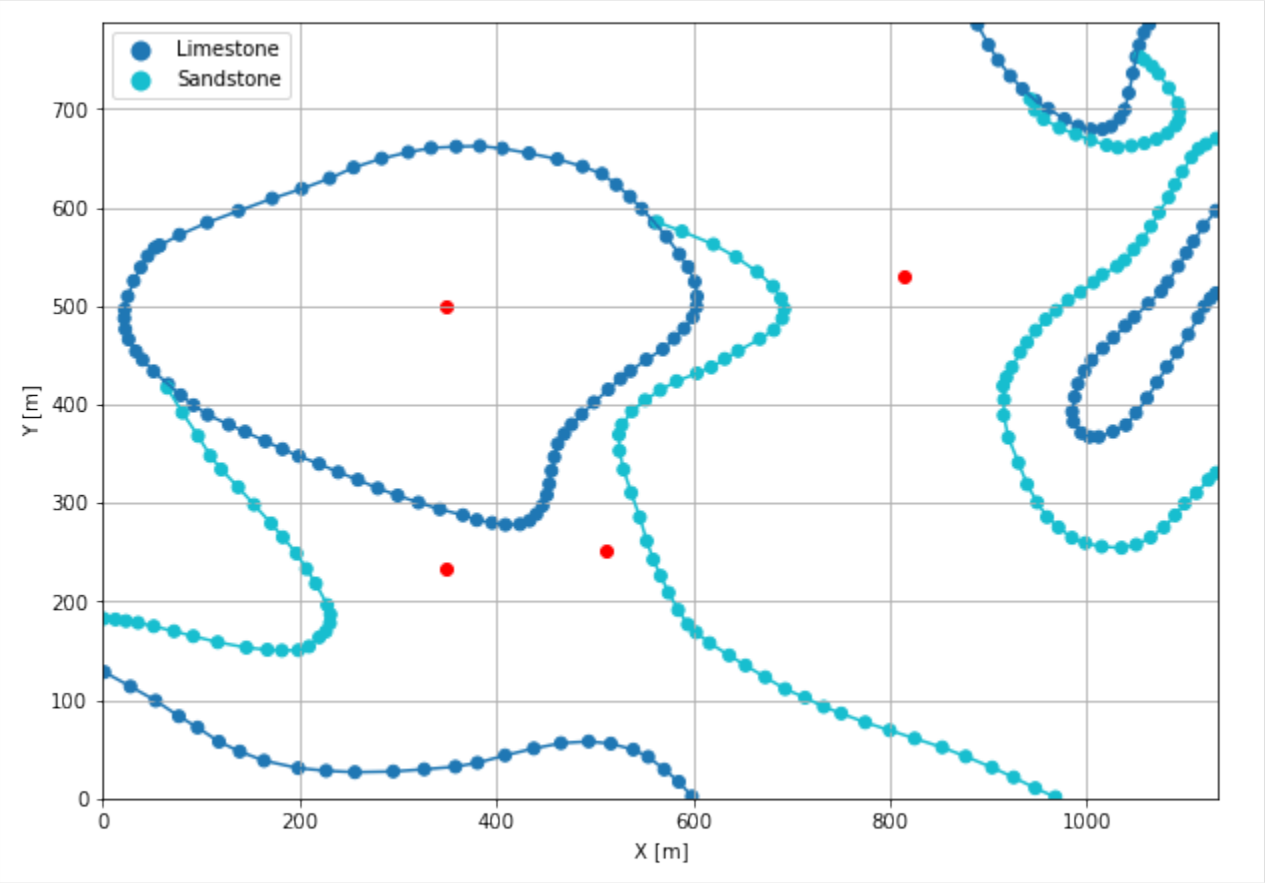
[19]:
```
import pandas as pd
orientations = pd.concat([orientations_limestone1, orientations_limestone2, orientations_
↪sandstone]).reset_index()
orientations['formation'] = ['Limestone', 'Limestone', 'Limestone', 'Sandstone']

orientations
```

[19]:
```
   index   dip  azimuth      Z                 geometry  polarity  formation  \
0      0  5.40   269.97 199.80  POINT (348.853 499.504)      1.00  Limestone
1      1  5.51   269.22 244.20  POINT (814.046 531.000)      1.00  Limestone
2      0  5.43   269.98 199.80  POINT (349.188 233.493)      1.00  Limestone
3      0 10.94   189.75 155.40  POINT (510.598 251.799)      1.00  Sandstone

       X      Y
0 348.85 499.50
1 814.05 531.00
2 349.19 233.49
3 510.60 251.80
```

Plotting the Orientations

[20]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1134)
plt.ylim(0, 788)
```

[20]:
```
(0.0, 788.0)
```



[21]:
```
orientations.is_valid
```

[21]:
```
0    True
1    True
2    True
3    True
dtype: bool
```

7.29.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[22]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[23]:
```
geo_model = gp.create_model('Model29')
geo_model
```

[23]:
```
Model29  2022-04-16 16:39
```

Initiate Data

[24]:
```
gp.init_data(geo_model, [0, 1134, 0, 788, 0, 600], [50,50,50],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[24]:
```
Model29  2022-04-16 16:39
```

Model Surfaces

[25]:
```
geo_model.surfaces
```

[25]:

	surface	series	order_surfaces	color	id
0	Limestone	Default series	1	#015482	1
1	Sandstone	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[26]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata1': ('Limestone'),
                          'Strata2': ('Sandstone')
                          },
                         remove_unused_series=True)
geo_model.add_surfaces('Claystone')
```

[26]:

	surface	series	order_surfaces	color	id
0	Limestone	Strata1	1	#015482	1
1	Sandstone	Strata2	1	#9f0052	2
2	Claystone	Strata2	2	#ffbe00	3

Adding additional Orientations

[27]:
```
geo_model.add_orientations(X=50, Y=600, Z=300, surface='Limestone', orientation = [270,5.
↪5,1])
geo_model.add_orientations(X=50, Y=700, Z=300, surface='Limestone', orientation = [270,5.
↪5,1])
geo_model.add_orientations(X=50, Y=800, Z=300, surface='Limestone', orientation = [270,5.
↪5,1])
```

(continues on next page)

(continued from previous page)

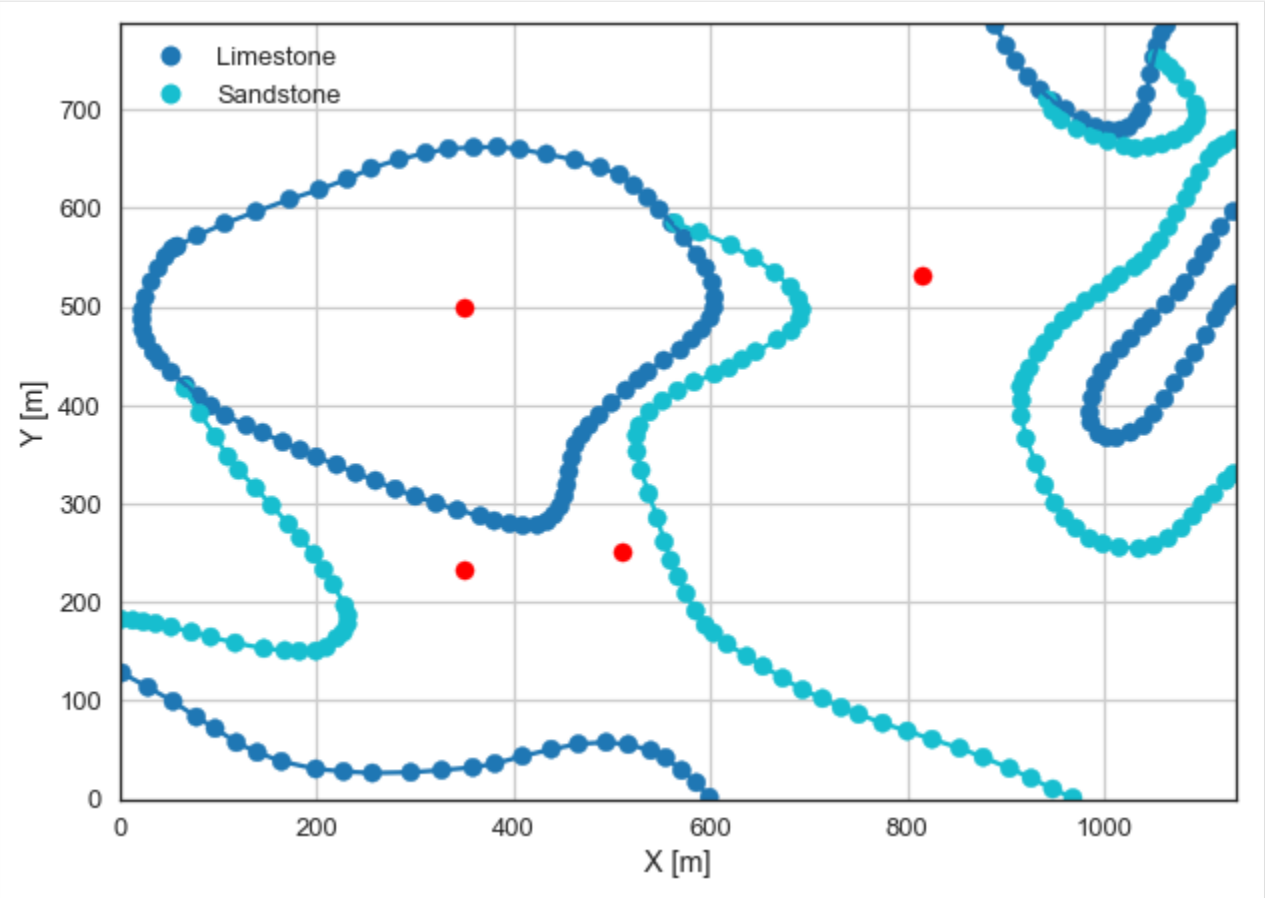
```
geo_model.add_orientations(X=100, Y=200, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=100, Y=400, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=100, Y=600, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])

geo_model.add_orientations(X=200, Y=200, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=200, Y=400, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=200, Y=600, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])

geo_model.add_orientations(X=600, Y=200, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=600, Y=400, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=600, Y=600, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])

geo_model.add_orientations(X=1000, Y=200, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=1000, Y=400, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])
geo_model.add_orientations(X=1000, Y=600, Z=300, surface='Limestone', orientation = [270,
↪5.5,1])

geo_model.add_orientations(X=200, Y=200, Z=300, surface='Sandstone', orientation = [190,
↪11,1])
geo_model.add_orientations(X=200, Y=400, Z=300, surface='Sandstone', orientation = [190,
↪11,1])
geo_model.add_orientations(X=200, Y=600, Z=300, surface='Sandstone', orientation = [190,
↪11,1])

geo_model.add_orientations(X=600, Y=200, Z=300, surface='Sandstone', orientation = [190,
↪11,1])
geo_model.add_orientations(X=600, Y=400, Z=300, surface='Sandstone', orientation = [190,
↪11,1])
geo_model.add_orientations(X=600, Y=600, Z=300, surface='Sandstone', orientation = [190,
↪11,1])

geo_model.add_orientations(X=1000, Y=200, Z=300, surface='Sandstone', orientation = [190,
↪11,1])
geo_model.add_orientations(X=1000, Y=400, Z=300, surface='Sandstone', orientation = [190,
↪11,1])
geo_model.add_orientations(X=1000, Y=600, Z=300, surface='Sandstone', orientation = [190,
↪11,1])
```

[27]:

	X	Y	Z	G_x	G_y	G_z	smooth	surface
0	348.85	499.50	199.80	-0.09	-0.00	1.00	0.01	Limestone
1	814.05	531.00	244.20	-0.10	-0.00	1.00	0.01	Limestone
2	349.19	233.49	199.80	-0.09	-0.00	1.00	0.01	Limestone
4	50.00	600.00	300.00	-0.10	0.00	1.00	0.01	Limestone
5	50.00	700.00	300.00	-0.10	0.00	1.00	0.01	Limestone
6	50.00	800.00	300.00	-0.10	0.00	1.00	0.01	Limestone
7	100.00	200.00	300.00	-0.10	0.00	1.00	0.01	Limestone
8	100.00	400.00	300.00	-0.10	0.00	1.00	0.01	Limestone
9	100.00	600.00	300.00	-0.10	0.00	1.00	0.01	Limestone
10	200.00	200.00	300.00	-0.10	0.00	1.00	0.01	Limestone
11	200.00	400.00	300.00	-0.10	0.00	1.00	0.01	Limestone
12	200.00	600.00	300.00	-0.10	0.00	1.00	0.01	Limestone
13	600.00	200.00	300.00	-0.10	0.00	1.00	0.01	Limestone
14	600.00	400.00	300.00	-0.10	0.00	1.00	0.01	Limestone
15	600.00	600.00	300.00	-0.10	0.00	1.00	0.01	Limestone
16	1000.00	200.00	300.00	-0.10	0.00	1.00	0.01	Limestone
17	1000.00	400.00	300.00	-0.10	0.00	1.00	0.01	Limestone
18	1000.00	600.00	300.00	-0.10	0.00	1.00	0.01	Limestone
3	510.60	251.80	155.40	-0.03	-0.19	0.98	0.01	Sandstone
19	200.00	200.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
20	200.00	400.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
21	200.00	600.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
22	600.00	200.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
23	600.00	400.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
24	600.00	600.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
25	1000.00	200.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
26	1000.00	400.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone
27	1000.00	600.00	300.00	-0.03	-0.19	0.98	0.01	Sandstone

[28]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1134)
plt.ylim(0, 788)
```

[28]:
```
(0.0, 788.0)
```



Showing the Number of Data Points

[29]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[29]:
```
     surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0  Limestone  Strata1               1  #015482   1                160          ␣
↪18
1  Sandstone  Strata2               1  #9f0052   2                145          ␣
↪10
2  Claystone  Strata2               2  #ffbe00   3                  0          ␣
↪0
```

Loading Digital Elevation Model

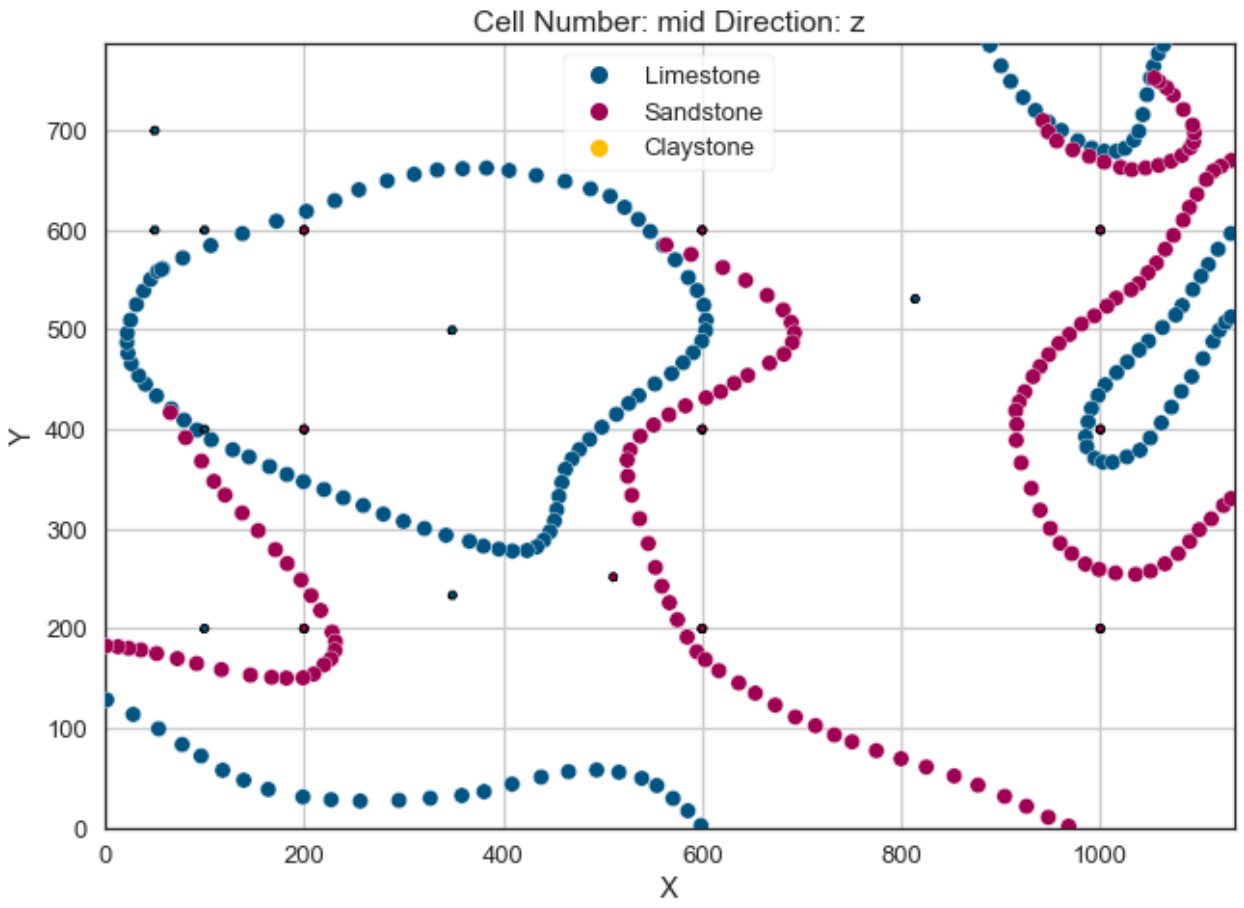
[30]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster29.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

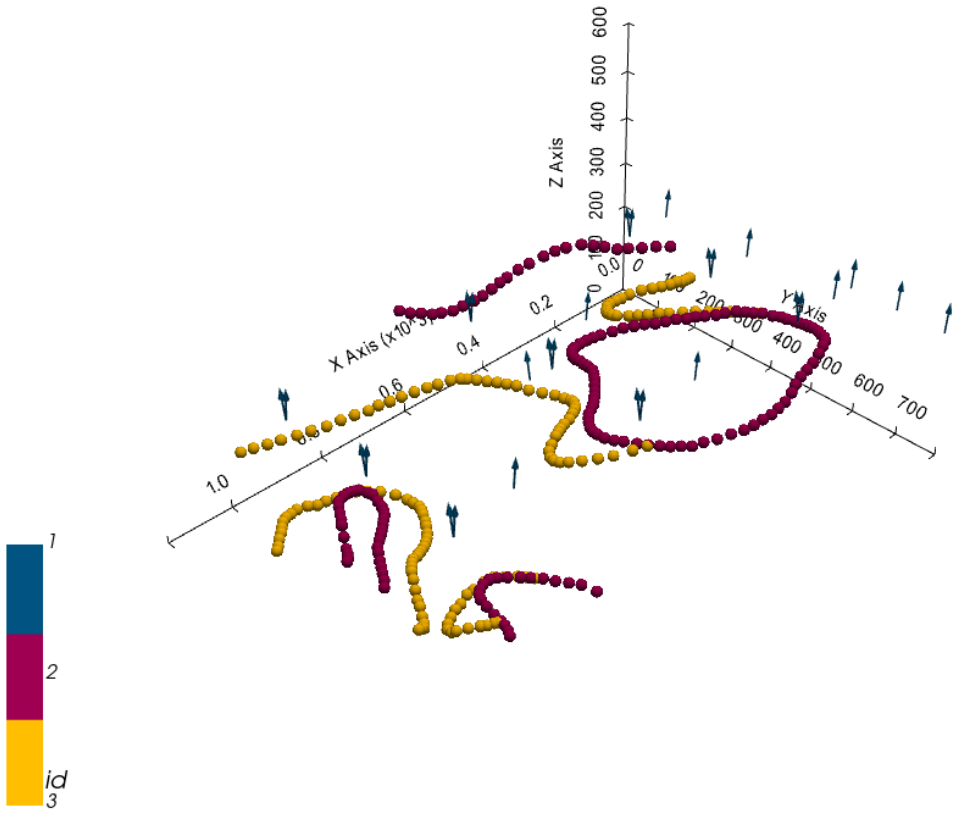
[30]:
```
Grid Object. Values:
array([[  11.34      ,    7.88      ,    6.        ],
       [  11.34      ,    7.88      ,   18.        ],
       [  11.34      ,    7.88      ,   30.        ],
       ...,
       [1131.50220264,  775.53164557,  266.35290527],
       [1131.50220264,  780.51898734,  267.26382446],
       [1131.50220264,  785.50632911,  268.18728638]])
```

Plotting Input Data

[31]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[32]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[32]:
```
<gempy.plot.vista.GemPyToVista at 0x19409249340>
```

Setting the Interpolator

[33]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range            1505.62
```

(continues on next page)

(continued from previous page)

```
$C_o$           53973.81
drift equations   [3, 3]
```

[33]:
```
<gempy.core.interpolator.InterpolatorModel at 0x19406fc6c40>
```

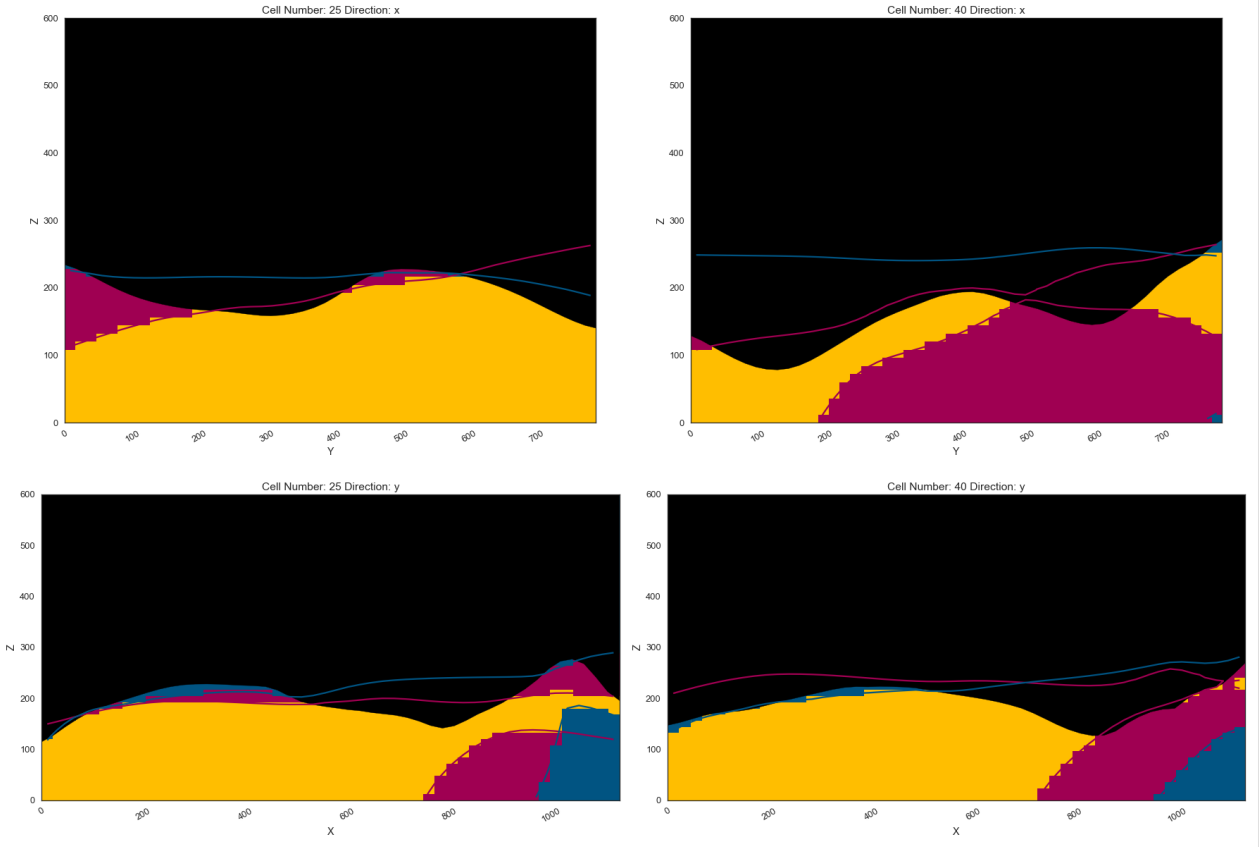
Computing Model

[34]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

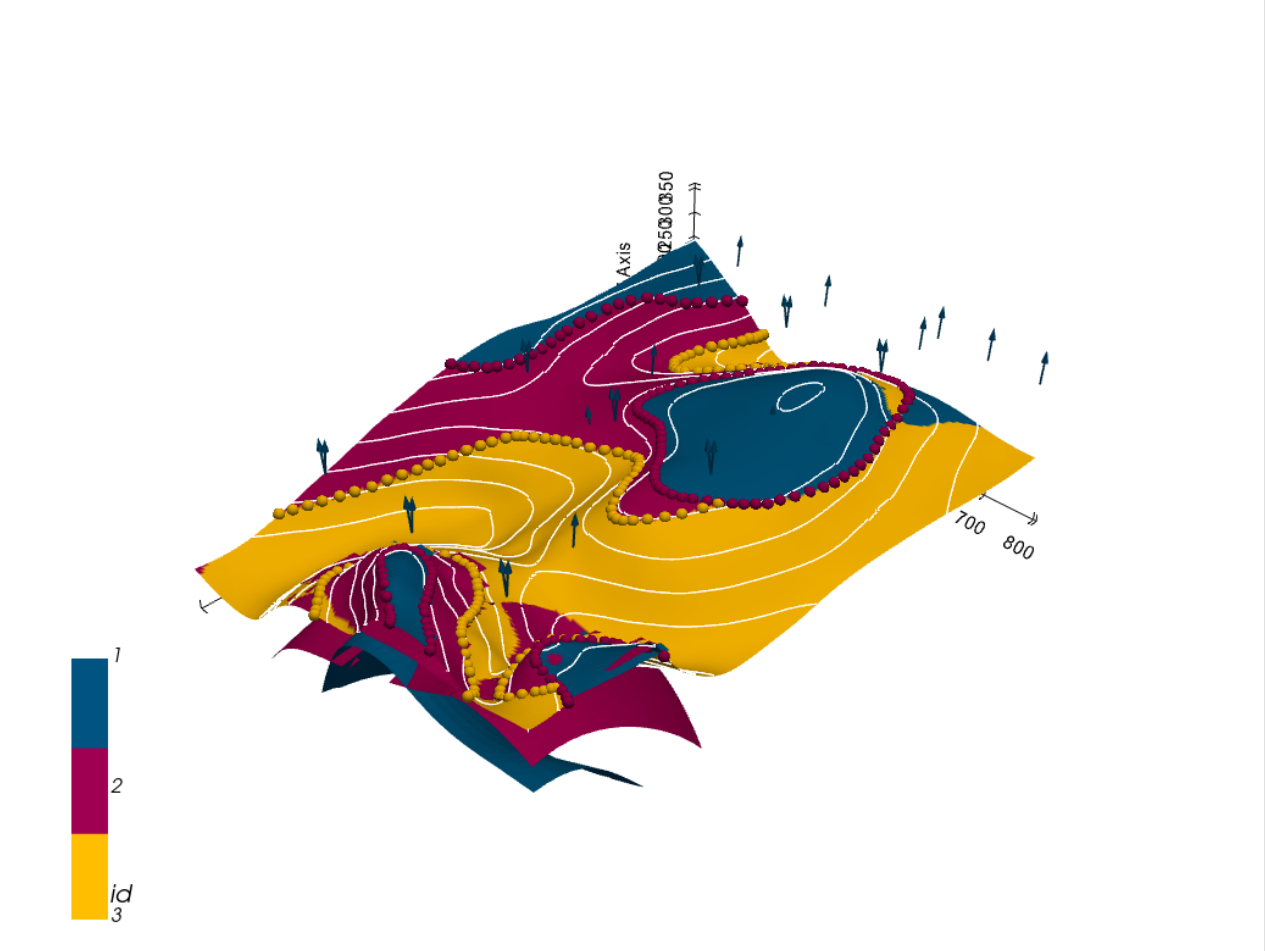
Plotting Cross Sections

[35]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 40, 25, 40], show_
↪topography=True, show_data=False)
```

[35]:
```
<gempy.plot.visualization_2d.Plot2D at 0x1940fcec700>
```



[36]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=False)
```



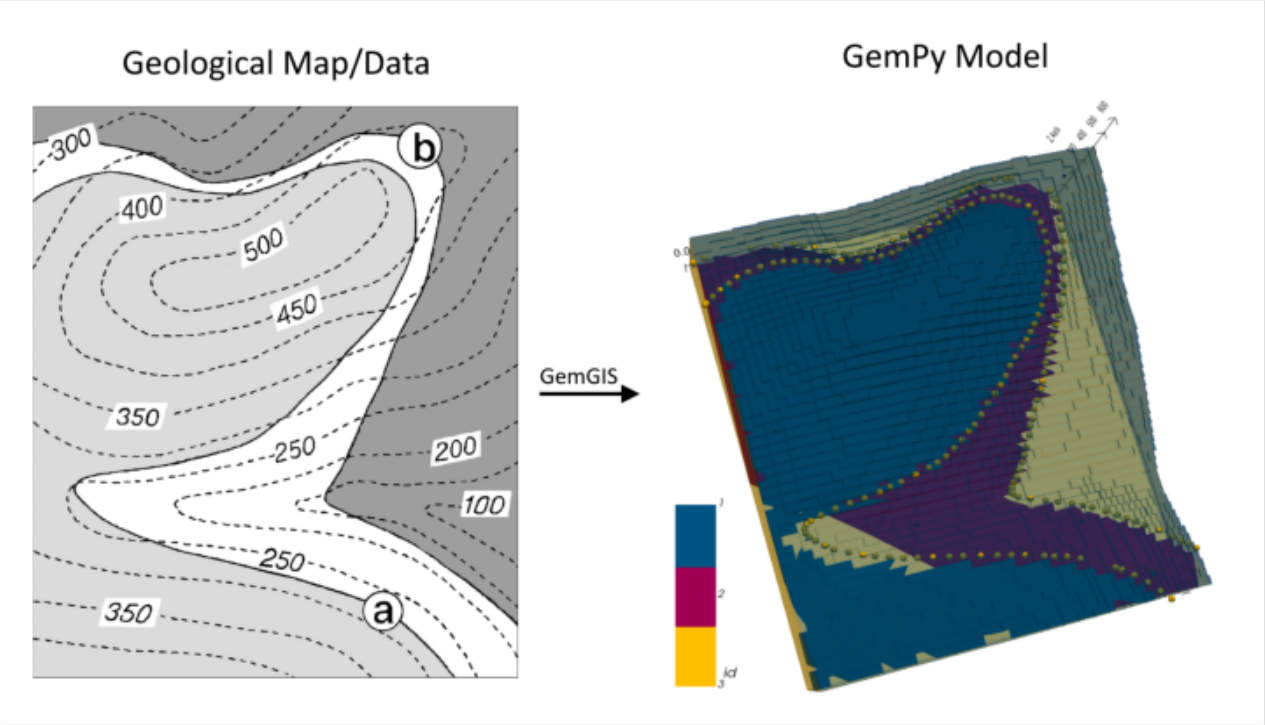
[ ]:

7.30 Example 30 - Planar Dipping Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 1157 m wide (W-E extent) and 1361 m high (N-S extent). The vertical model extents varies between 0 m and 600 m. The model represents two planar stratigraphic units (blue and red) dipping towards the northeast above an unspecified basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and were used to calculate orientations for the `GemPy` model. These will be loaded into the model directly. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example30.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.30.1 Licensing

7.30.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.30.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

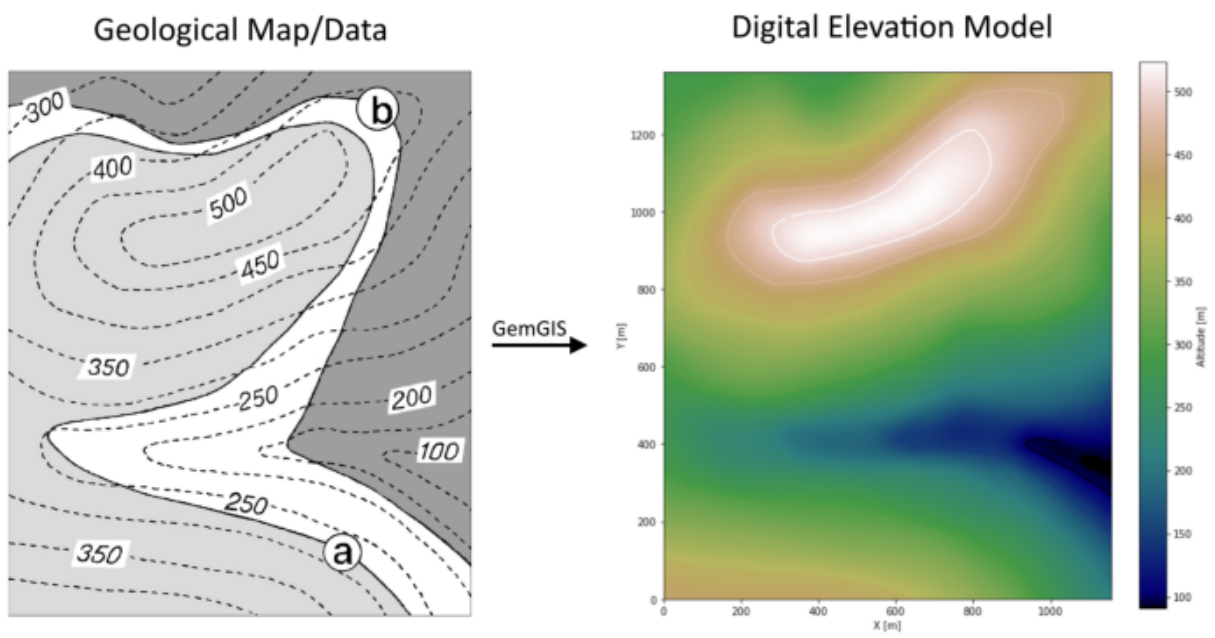
[4]:
```
file_path = 'data/example30/'
gg.download_gemgis_data.download_tutorial_data(filename="example30_planar_dipping_layers.
↪zip", dirpath=file_path)
```

```
Downloading file 'example30_planar_dipping_layers.zip' from 'https://rwth-aachen.sciebo.
↪de/s/AfXRsZywYDbUF34/download?path=%2Fexample30_planar_dipping_layers.zip' to 'C:\
↪Users\ale93371\Documents\gemgis\docs\getting_started\example\data\example30'.
```

7.30.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/dem_example30.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo30.shp')
topo.head()
```

[6]:

	id	Z	geometry
0	None	400	LINESTRING (4.295 91.552, 43.424 86.924, 83.39...
1	None	350	LINESTRING (3.874 182.012, 64.462 176.122, 125...
2	None	300	LINESTRING (4.716 300.242, 37.534 286.357, 86...
3	None	100	LINESTRING (1156.295 265.320, 1138.203 282.570...
4	None	300	LINESTRING (1.771 1206.526, 24.912 1231.771, 4...

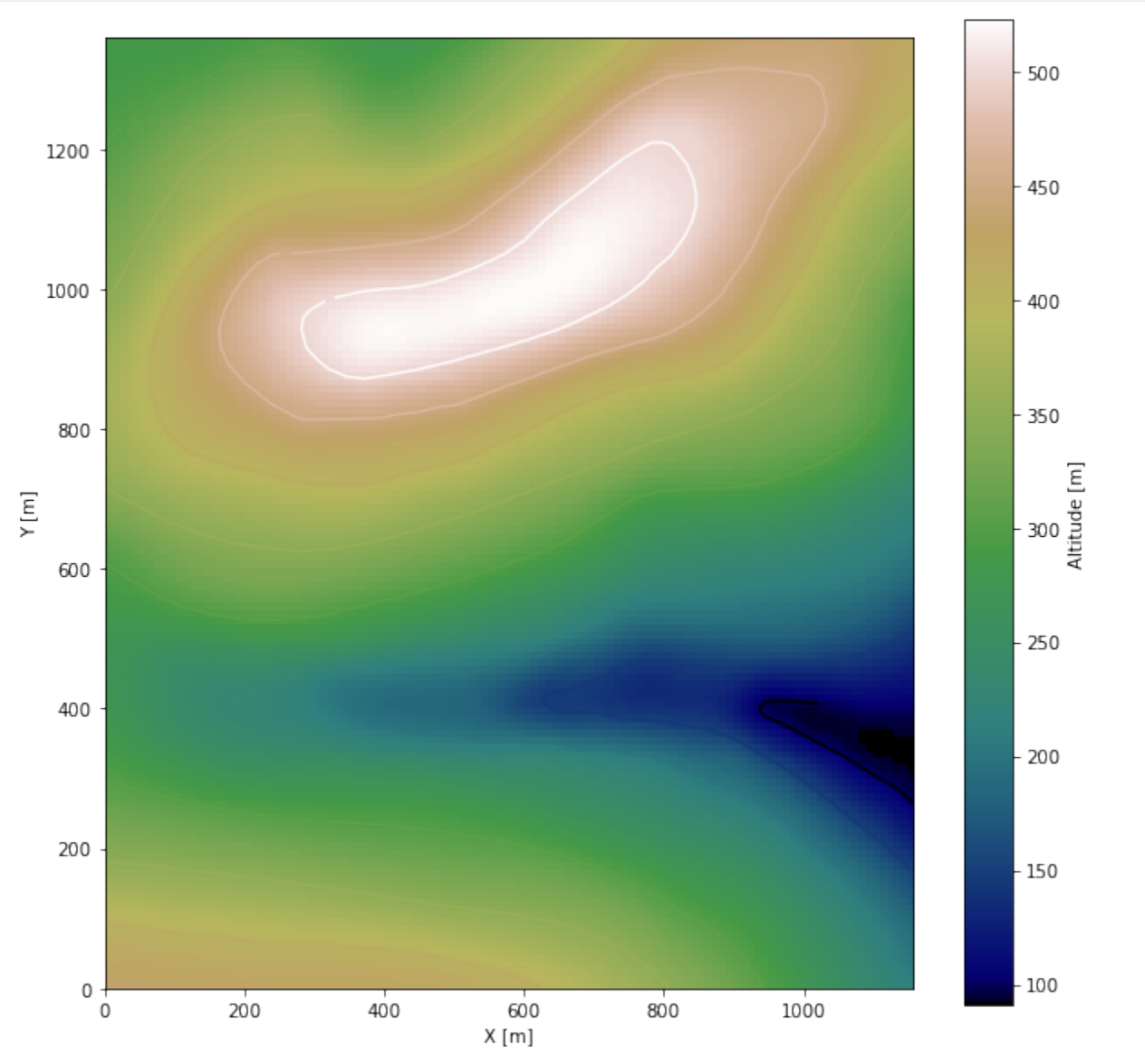
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=10)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

fix, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = plt.imshow(topo_raster, origin='lower', extent=[0, 1157, 0, 1361], cmap='gist_earth
↪')
cbar = plt.colorbar(im)
cbar.set_label('Altitude [m]')
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1157)
plt.ylim(0, 1361)
```

[8]:
```
(0.0, 1361.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster30.tif', extent = [0, 1157, 0, 1361], crs =' EPSG : 4326', overwrite_f ile = True)$

Opening Raster

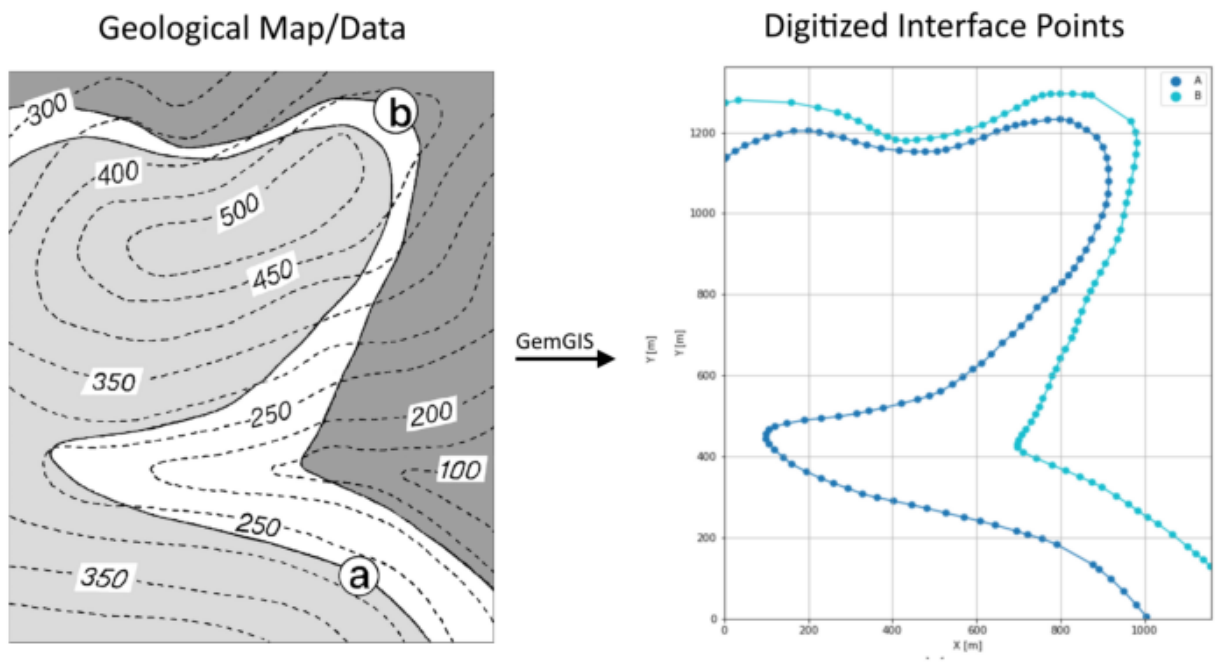
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster30.tif')
```

7.30.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../images/interfaces_example30.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces30.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	A	LINESTRING (1003.775 4.248, 979.792 33.489, 94...
1	None	B	LINESTRING (2.612 1272.583, 33.747 1279.105, 1...

Extracting Z coordinate from Digital Elevation Model

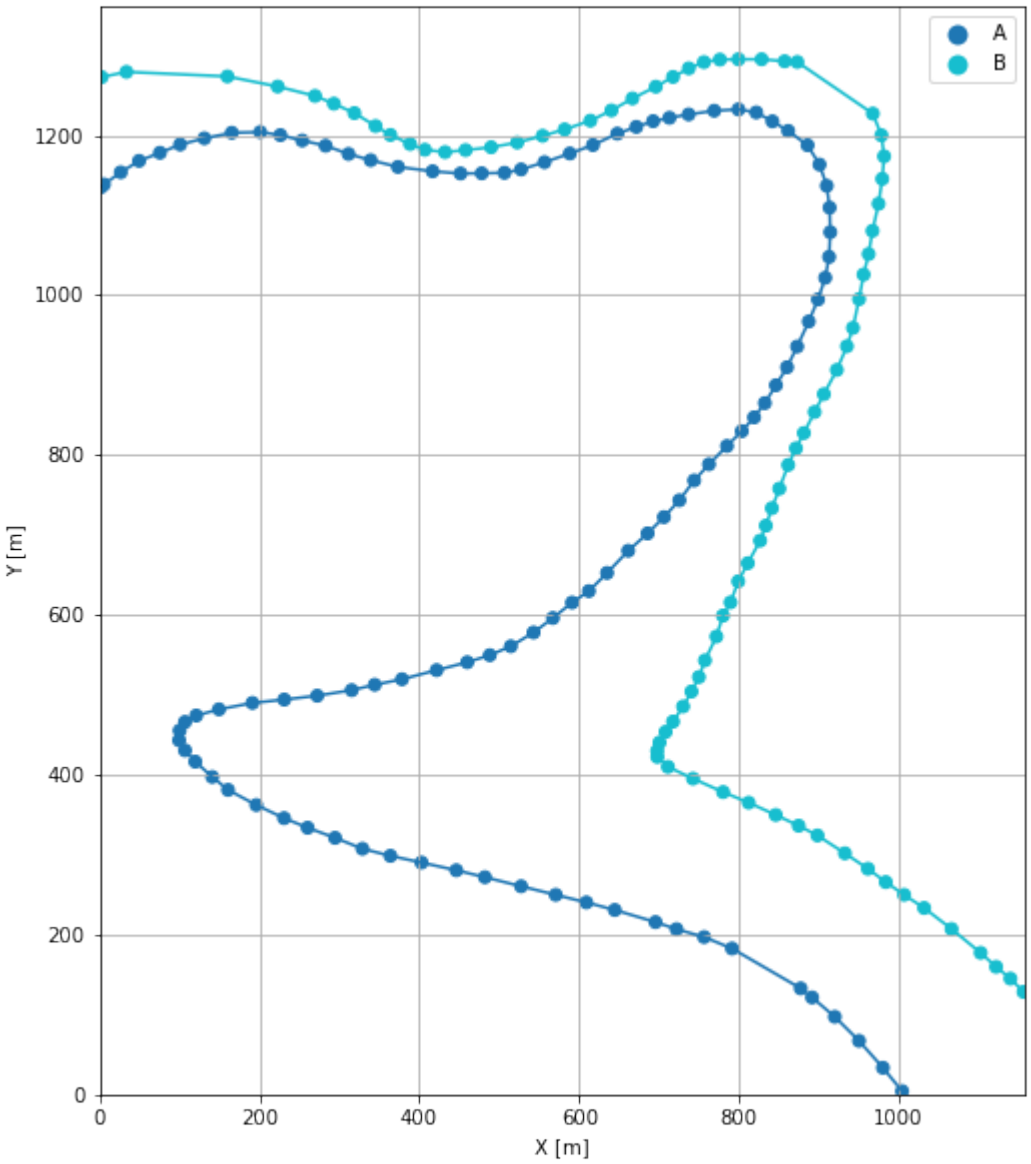
[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['A', 'B'])]
interfaces_coords
```

[12]:
```
    formation                  geometry       X      Y      Z
0           A    POINT (1003.775 4.248) 1003.77   4.25 274.58
1           A    POINT (979.792 33.489)  979.79  33.49 276.06
2           A    POINT (949.919 67.149)  949.92  67.15 280.72
3           A    POINT (919.626 97.232)  919.63  97.23 282.11
4           A   POINT (891.226 121.425)  891.23 121.43 279.32
..        ...                       ...     ...    ...    ...
178         B  POINT (1065.624 207.047) 1065.62 207.05 178.74
179         B  POINT (1102.019 177.174) 1102.02 177.17 176.01
180         B  POINT (1121.373 159.713) 1121.37 159.71 175.79
181         B  POINT (1139.255 144.987) 1139.25 144.99 170.98
182         B  POINT (1155.664 128.367) 1155.66 128.37 176.47

[183 rows x 5 columns]
```

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1157)
plt.ylim(0, 1361)
```

[13]:
```
(0.0, 1361.0)
```



7.30.6 Orientations from Strike Lines

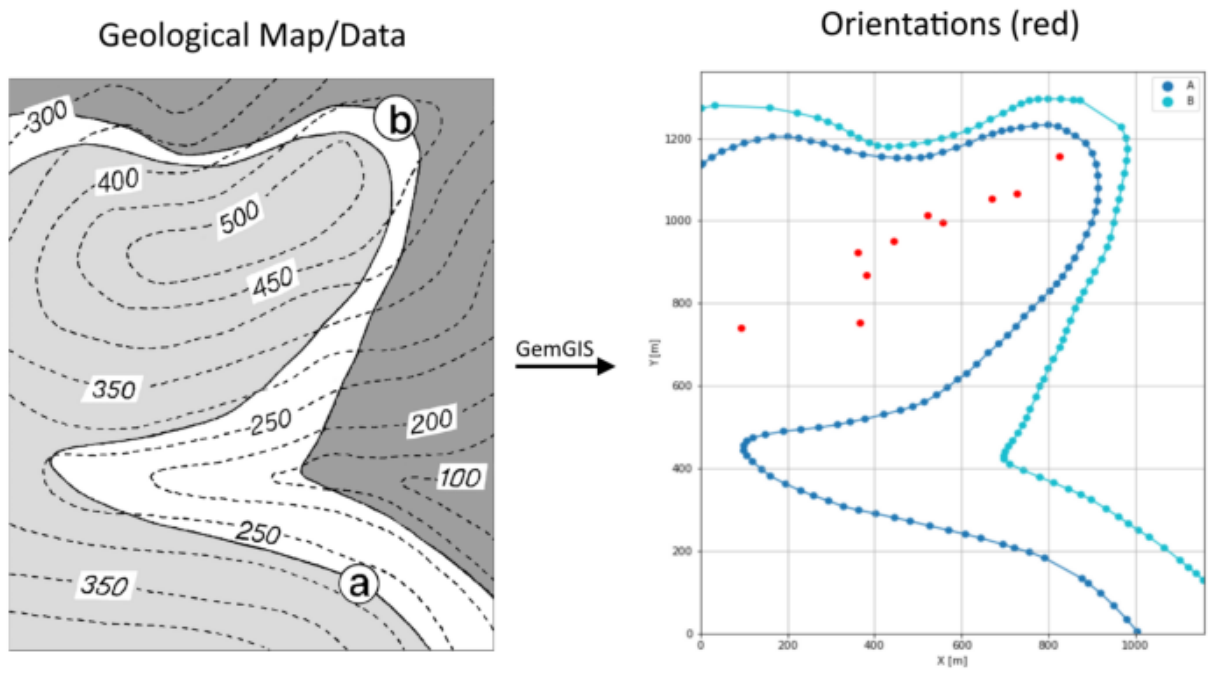
Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

[14]:
```
img = mpimg.imread('../images/orientations_example30.png')
plt.figure(figsize=(10, 10))
```

(continues on next page)

(continued from previous page)

```
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[15]:
```
strikes = gpd.read_file(file_path + 'strikes30.shp')
strikes
```

[15]:

	id	formation	Z	geometry
0	5	A	450	LINESTRING (672.648 1212.207, 908.266 1026.237)
1	4	A	400	LINESTRING (488.362 1152.461, 840.105 873.928)
2	3	A	350	LINESTRING (155.132 1200.426, 738.285 759.485)
3	7	B	450	LINESTRING (981.475 1168.449, 790.457 1296.356)
4	6	B	400	LINESTRING (936.876 944.613, 586.816 1212.207)
5	5	B	350	LINESTRING (872.923 809.133, 275.465 1250.915)
6	4	B	300	LINESTRING (837.581 713.203, 103.801 1281.209)
7	2	A	300	LINESTRING (595.231 618.115, -48.508 1111.228)
8	1	A	250	LINESTRING (104.643 462.439, -276.552 771.266)
9	3	B	250	LINESTRING (790.457 620.639, 47.421 1186.120)
10	2	B	200	LINESTRING (749.224 524.709, -58.606 1140.680)
11	1	B	150	LINESTRING (765.212 382.497, 12.079 963.125)

Calculating Orientations for each formation

[16]:
```
orientations_a = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'A'].sort_values(by='id', ascending=True).reset_index())
orientations_a
```

[16]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0  6.98   217.87 275.00    POINT (93.703 740.762)      1.00         A  93.70
1 14.41   217.29 325.00   POINT (360.035 922.313)      1.00         A 360.03
2 18.11   217.45 375.00   POINT (555.471 996.575)      1.00         A 555.47
3 17.23   218.35 425.00  POINT (727.345 1066.208)      1.00         A 727.35

        Y
0  740.76
1  922.31
2  996.57
3 1066.21
```

[17]:
```
orientations_b = gg.vector.calculate_orientations_from_strike_lines(gdf=strikes[strikes[
↪'formation'] == 'B'].sort_values(by='id', ascending=True).reset_index())
orientations_b
```

[17]:
```
    dip  azimuth      Z                  geometry  polarity formation      X  \
0 26.95   217.47 175.00   POINT (366.977 752.753)      1.00         B 366.98
1 26.47   217.30 225.00   POINT (382.124 868.037)      1.00         B 382.12
2 26.07   217.51 275.00   POINT (444.815 950.293)      1.00         B 444.82
3 31.65   217.25 325.00  POINT (522.443 1013.615)      1.00         B 522.44
4 18.79   216.72 375.00  POINT (668.020 1054.217)      1.00         B 668.02
5 14.70   216.63 425.00  POINT (823.906 1155.406)      1.00         B 823.91

        Y
0  752.75
1  868.04
2  950.29
3 1013.61
4 1054.22
5 1155.41
```

Merging Orientations

[18]:
```
import pandas as pd
orientations = pd.concat([orientations_a, orientations_b])
orientations = orientations[orientations['formation'].isin(['A', 'B'])].reset_index()
orientations
```

[18]:
```
   index   dip  azimuth      Z                  geometry  polarity formation  \
0      0  6.98   217.87 275.00    POINT (93.703 740.762)      1.00         A
1      1 14.41   217.29 325.00   POINT (360.035 922.313)      1.00         A
2      2 18.11   217.45 375.00   POINT (555.471 996.575)      1.00         A
3      3 17.23   218.35 425.00  POINT (727.345 1066.208)      1.00         A
4      0 26.95   217.47 175.00   POINT (366.977 752.753)      1.00         B
5      1 26.47   217.30 225.00   POINT (382.124 868.037)      1.00         B
6      2 26.07   217.51 275.00   POINT (444.815 950.293)      1.00         B
```

(continues on next page)

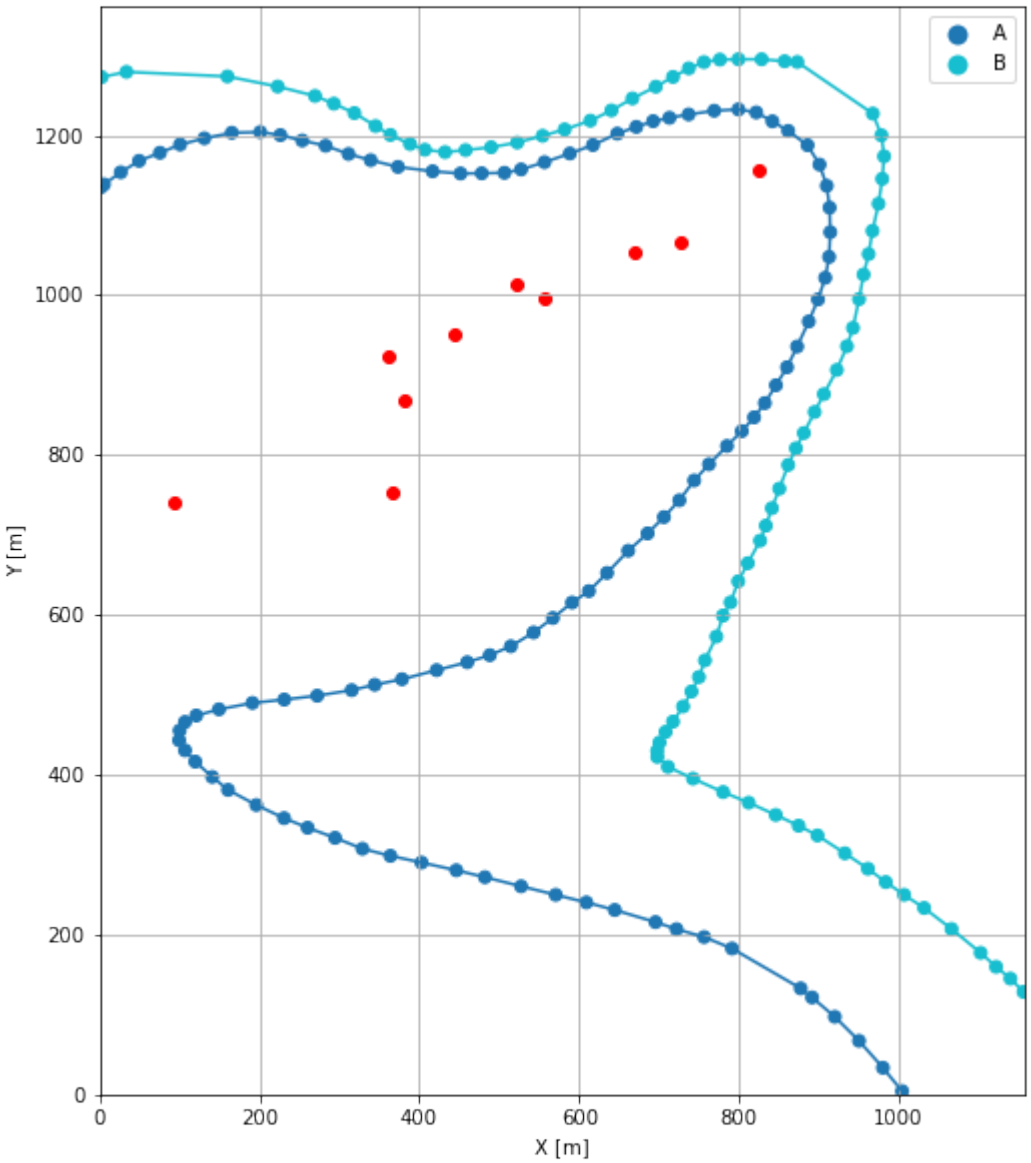
(continued from previous page)

```
7      3 31.65   217.25 325.00  POINT (522.443 1013.615)      1.00          B
8      4 18.79   216.72 375.00  POINT (668.020 1054.217)      1.00          B
9      5 14.70   216.63 425.00  POINT (823.906 1155.406)      1.00          B

       X       Y
0  93.70  740.76
1 360.03  922.31
2 555.47  996.57
3 727.35 1066.21
4 366.98  752.75
5 382.12  868.04
6 444.82  950.29
7 522.44 1013.61
8 668.02 1054.22
9 823.91 1155.41
```

Plotting the Orientations

[19]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 1157)
plt.ylim(0, 1361)
```

[19]:
```
(0.0, 1361.0)
```



7.30.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[20]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[21]:
```
geo_model = gp.create_model('Model30')
geo_model
```

[21]:
```
Model30  2022-04-17 08:51
```

Initiate Data

[22]:
```
gp.init_data(geo_model, [0, 1157, 0, 1361, 0, 600], [50,50,50],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[22]:
```
Model30  2022-04-17 08:51
```

Model Surfaces

[23]:
```
geo_model.surfaces
```

[23]:
```
  surface          series  order_surfaces    color  id
0       A  Default series               1  #015482   1
1       B  Default series               2  #9f0052   2
```

Mapping the Stack to Surfaces

[24]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata1': ('A'),
                          'Strata2': ('B')
                         },
                         remove_unused_series=True)
geo_model.add_surfaces('C')
```

[24]:
```
  surface   series  order_surfaces    color  id
0       A  Strata1               1  #015482   1
1       B  Strata2               1  #9f0052   2
2       C  Strata2               2  #ffbe00   3
```

7.30.8 Adding additional orientations

[25]:
```
geo_model.add_orientations(X=1000, Y=1250, Z=500, surface='A', orientation = [217.5,18,
↪1])
geo_model.add_orientations(X=100, Y=200, Z=300, surface='A', orientation = [217.5,7,1])
geo_model.add_orientations(X=400, Y=400, Z=200, surface='A', orientation = [217.5,14.5,
↪1])

geo_model.add_orientations(X=850, Y=300, Z=200, surface='B', orientation = [217.5,26,1])
```

(continues on next page)

(continued from previous page)

```
geo_model.add_orientations(X=1000, Y=100, Z=200, surface='B', orientation = [217.5,26,1])
```

[25]:

	X	Y	Z	G_x	G_y	G_z	smooth	surface
0	93.70	740.76	275.00	-0.07	-0.10	0.99	0.01	A
1	360.03	922.31	325.00	-0.15	-0.20	0.97	0.01	A
2	555.47	996.57	375.00	-0.19	-0.25	0.95	0.01	A
3	727.35	1066.21	425.00	-0.18	-0.23	0.96	0.01	A
10	1000.00	1250.00	500.00	-0.19	-0.25	0.95	0.01	A
11	100.00	200.00	300.00	-0.07	-0.10	0.99	0.01	A
12	400.00	400.00	200.00	-0.15	-0.20	0.97	0.01	A
4	366.98	752.75	175.00	-0.28	-0.36	0.89	0.01	B
5	382.12	868.04	225.00	-0.27	-0.35	0.90	0.01	B
6	444.82	950.29	275.00	-0.27	-0.35	0.90	0.01	B
7	522.44	1013.61	325.00	-0.32	-0.42	0.85	0.01	B
8	668.02	1054.22	375.00	-0.19	-0.26	0.95	0.01	B
9	823.91	1155.41	425.00	-0.15	-0.20	0.97	0.01	B
13	850.00	300.00	200.00	-0.27	-0.35	0.90	0.01	B
14	1000.00	100.00	200.00	-0.27	-0.35	0.90	0.01	B

Showing the Number of Data Points

[26]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[26]:

	surface	series	order_surfaces	color	id	No. of Interfaces	No. of Orientations
0	A	Strata1	1	#015482	1	102	7
1	B	Strata2	1	#9f0052	2	81	8
2	C	Strata2	2	#ffbe00	3	0	0

Loading Digital Elevation Model

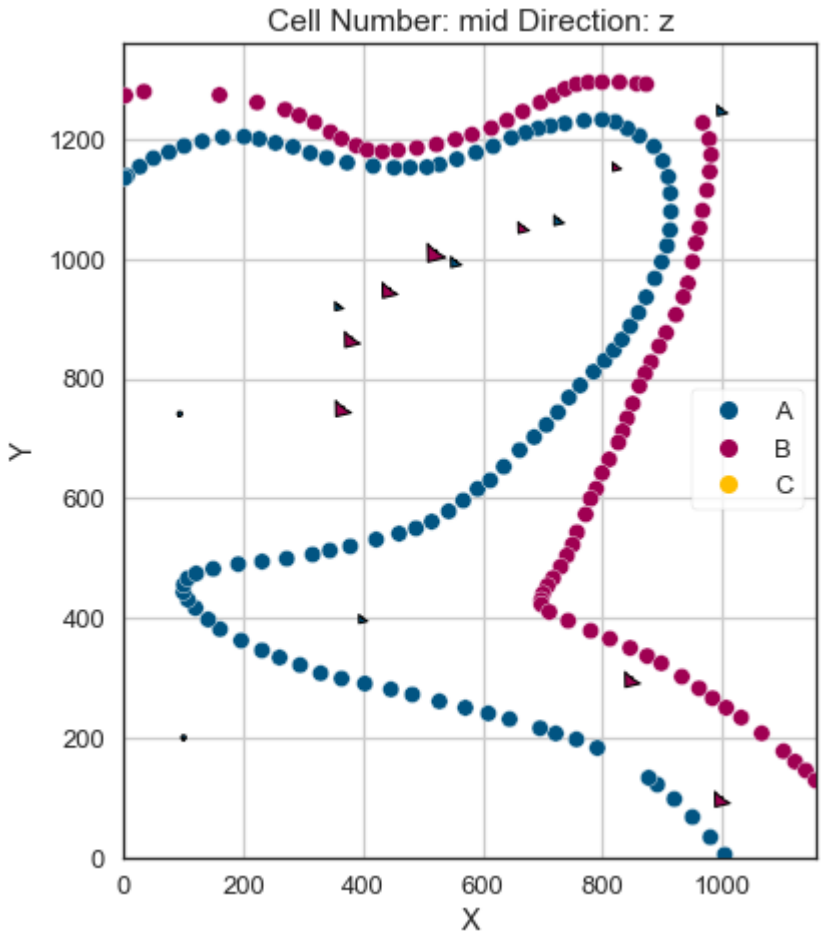
[27]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster30.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

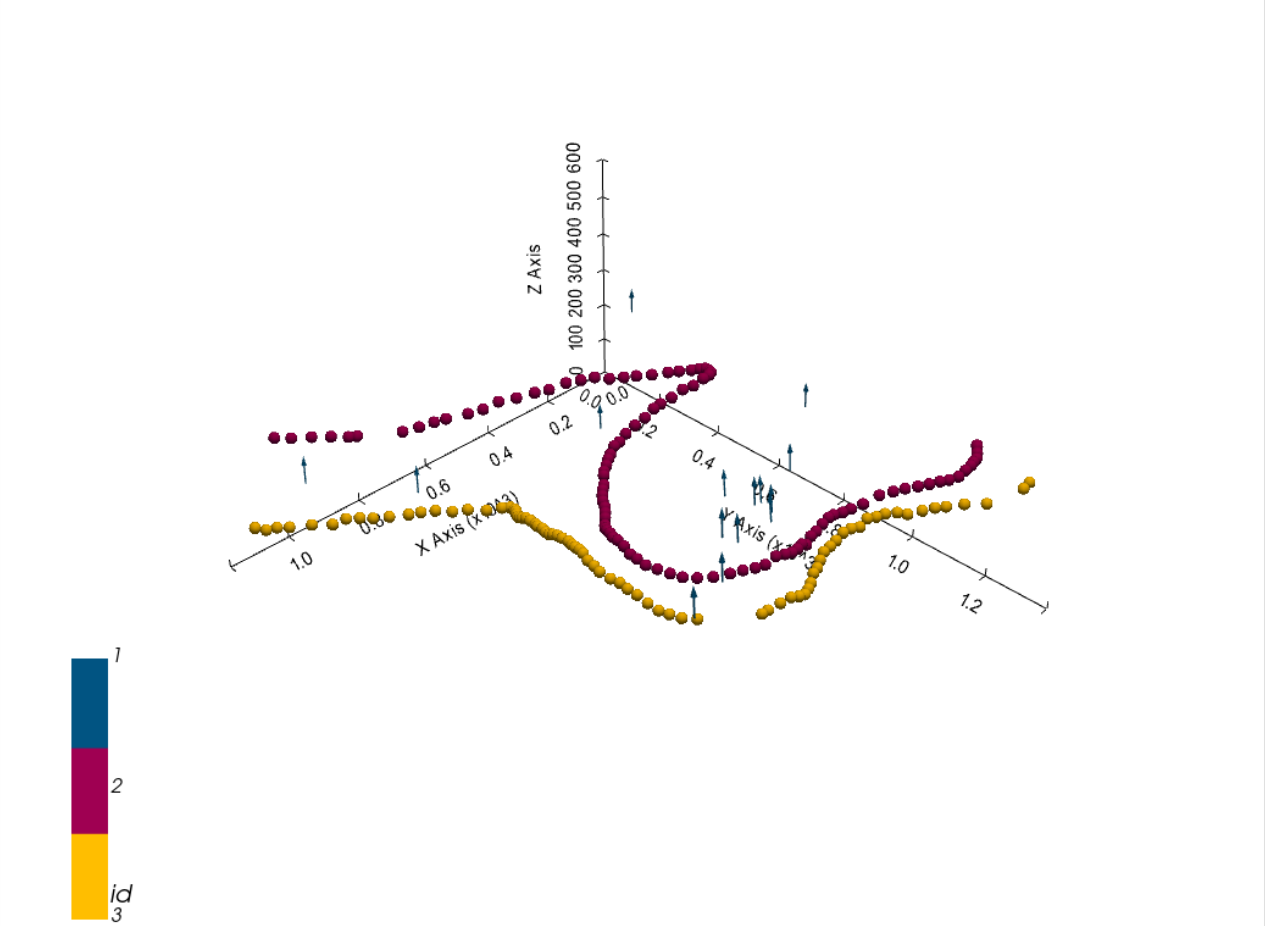
[27]:
```
Grid Object. Values:
array([[  11.57      ,   13.61      ,    6.        ],
       [  11.57      ,   13.61      ,   18.        ],
       [  11.57      ,   13.61      ,   30.        ],
       ...,
       [1152.01293103, 1335.98161765,  415.3067627 ],
       [1152.01293103, 1345.98897059,  416.07495117],
       [1152.01293103, 1355.99632353,  416.66900635]])
```

Plotting Input Data

[28]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[29]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[29]:
```
<gempy.plot.vista.GemPyToVista at 0x26e9ff0c730>
```

Setting the Interpolator

[30]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range             1884.4
$C_o$            84546.9
drift equations  [3, 3]
```

[30]:
```
<gempy.core.interpolator.InterpolatorModel at 0x26ea597b7f0>
```

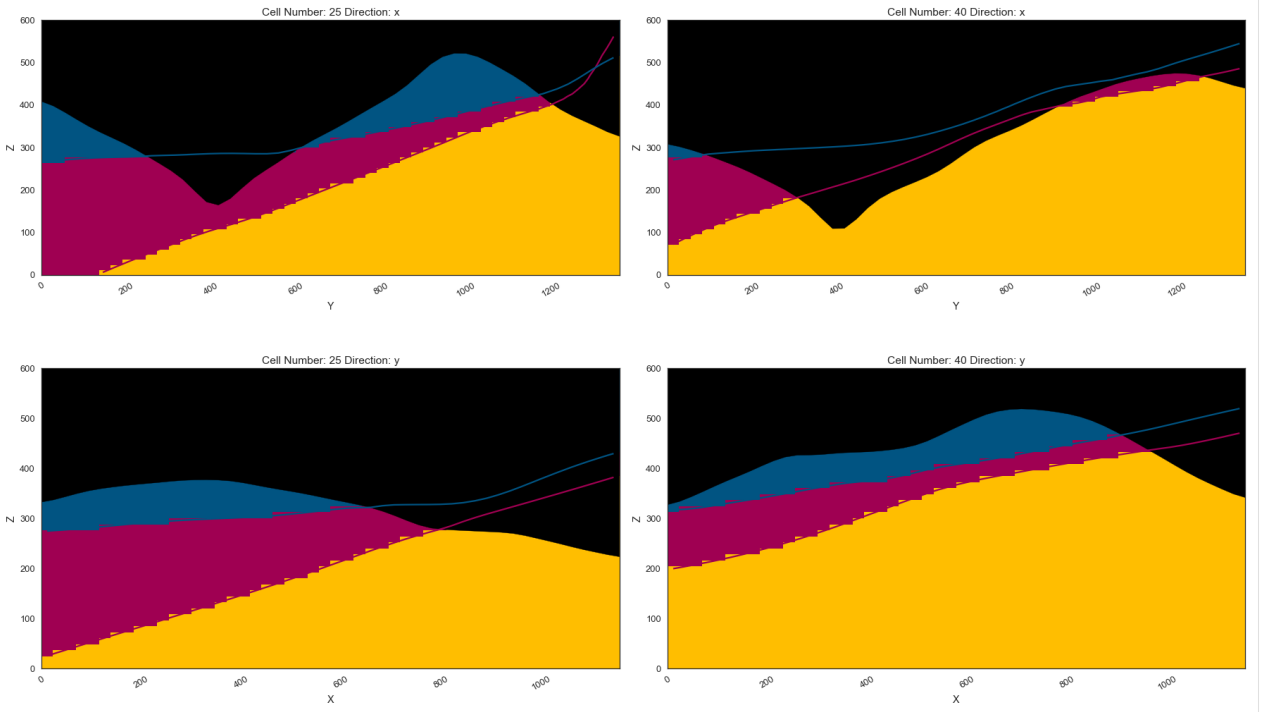
Computing Model

[31]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

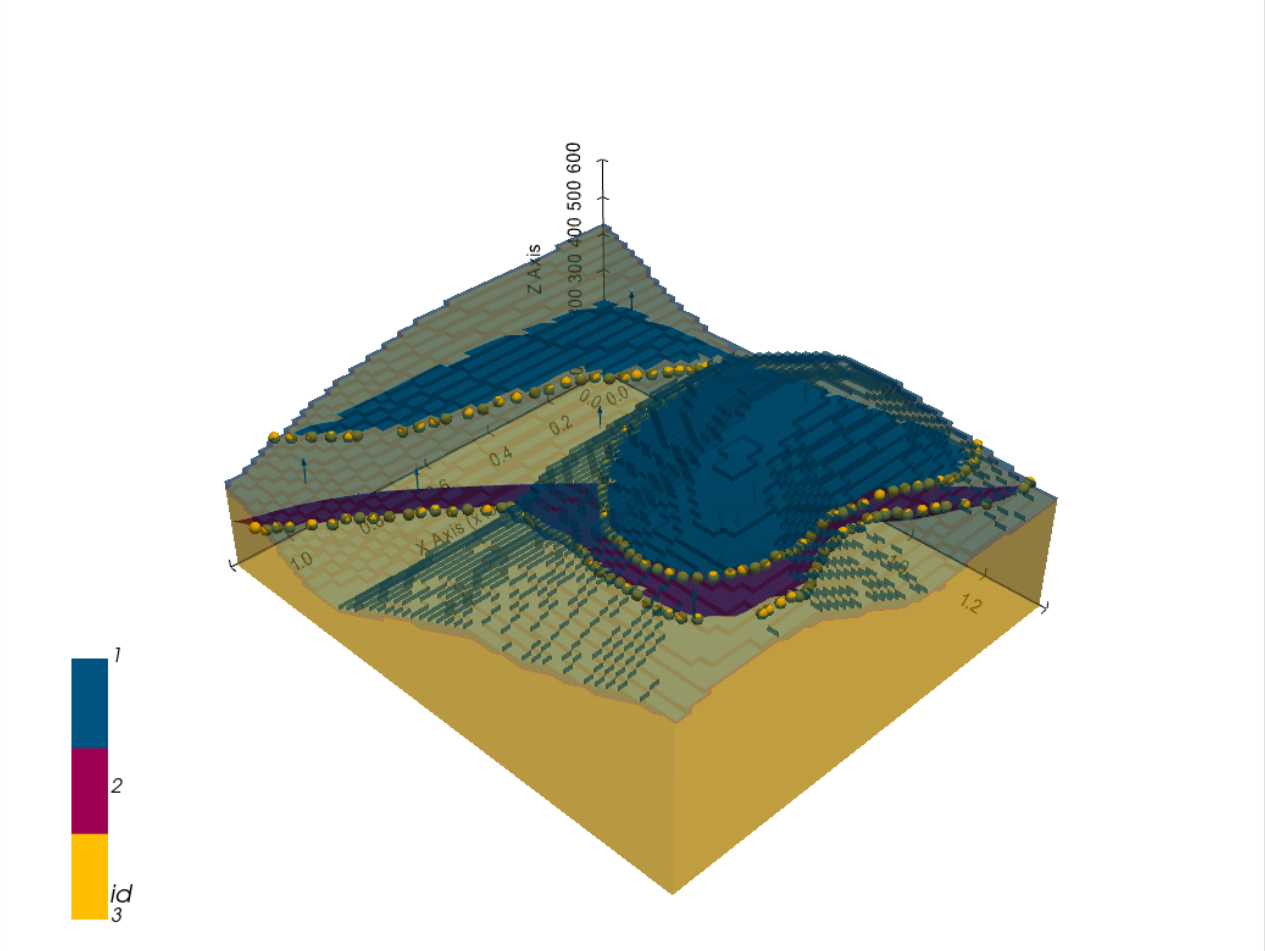
Plotting Cross Sections

[32]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 40, 25, 40], show_
↪topography=True, show_data=False)
```

[32]:
```
<gempy.plot.visualization_2d.Plot2D at 0x26eaf241040>
```



[33]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=False,
                 plotter_type='basic', notebook=True, show_lith=True)
```



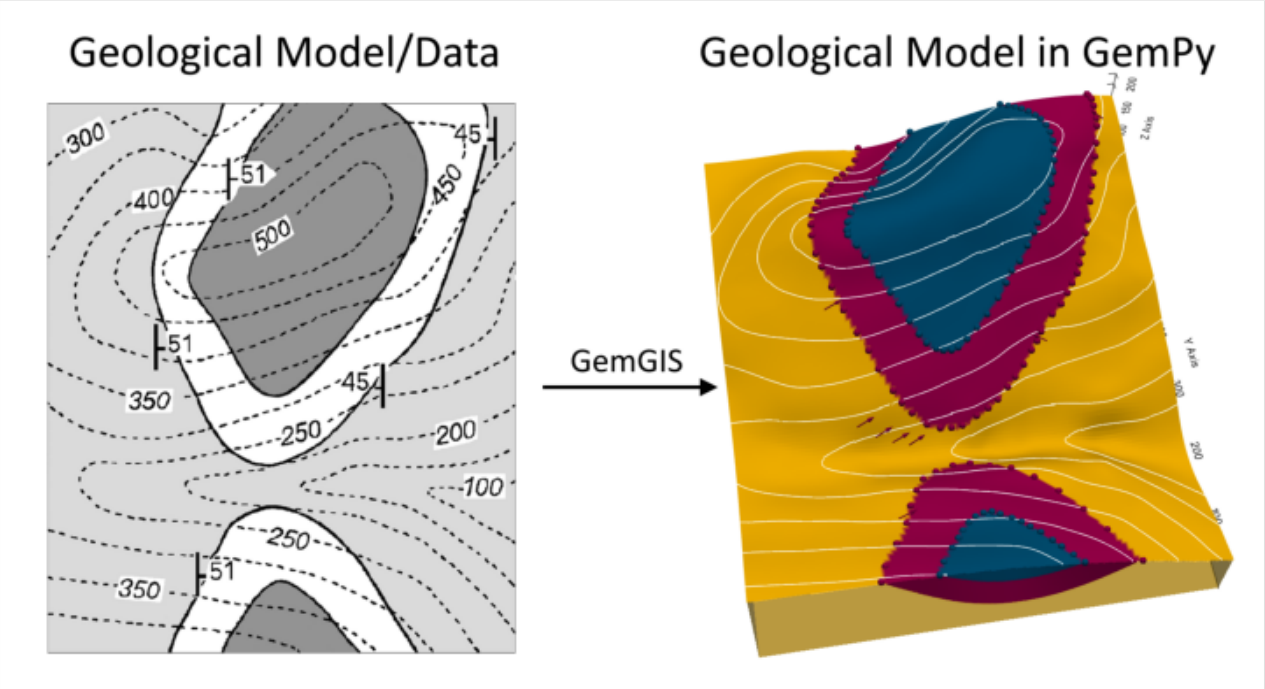
[ ]:

7.31 Example 31 - Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 601 m wide (W-E extent) and 705 m high (N-S extent). The vertical model extents varies between 0 m and 300 m. The model represents two folded stratigraphic units (blue and red) above an unspecified basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and were used to calculate orientations for the `GemPy` model. These will be loaded into the model directly. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example31.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.31.1 Licensing

7.31.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.31.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

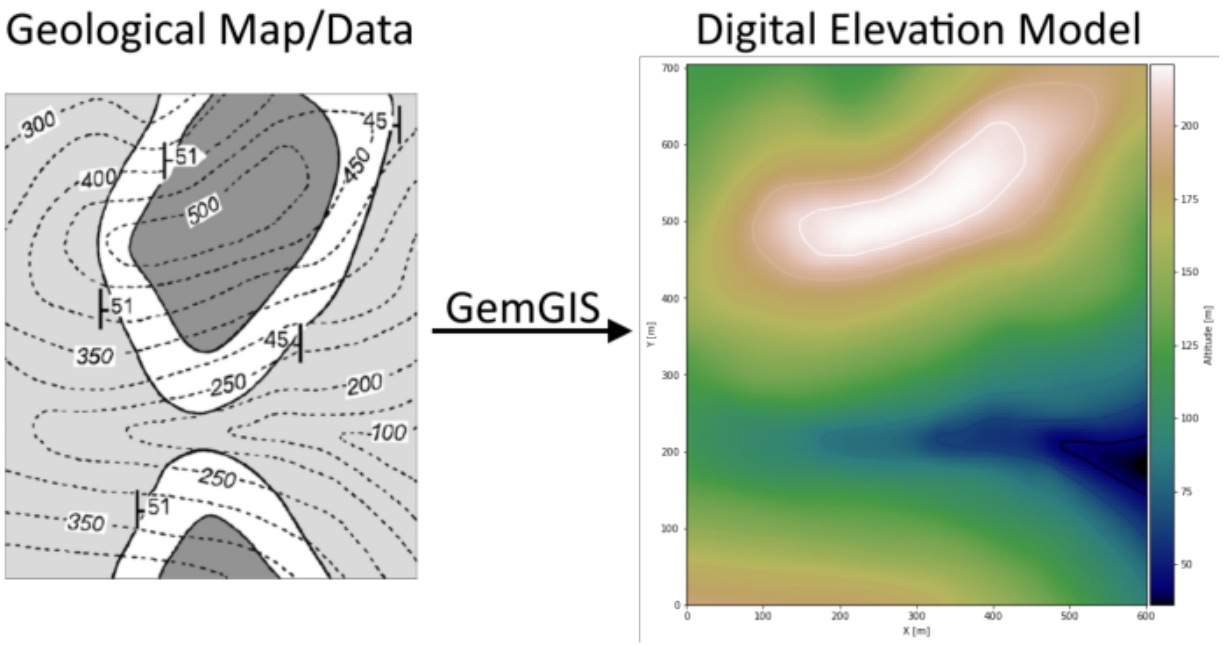
[3]:
```
import geopandas as gpd
import rasterio
```

[4]:
```
file_path = 'data/example31/'
gg.download_gemgis_data.download_tutorial_data(filename="example31_folded_layers.zip",␣
↪dirpath=file_path)
```

7.31.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
img = mpimg.imread('../images/dem_example31.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo31.shp')
topo['Z'] = topo['Z']*0.425
topo.head()
```

[6]:

	id	Z	geometry
0	None	170.00	LINESTRING (0.994 52.851, 36.954 48.266, 68.32...
1	None	148.75	LINESTRING (1.235 98.706, 42.022 94.603, 80.39...
2	None	127.50	LINESTRING (1.477 157.955, 44.435 147.818, 81...
3	None	127.50	LINESTRING (1.597 320.738, 11.492 310.722, 22...
4	None	106.25	LINESTRING (553.299 3.497, 534.233 30.527, 513...

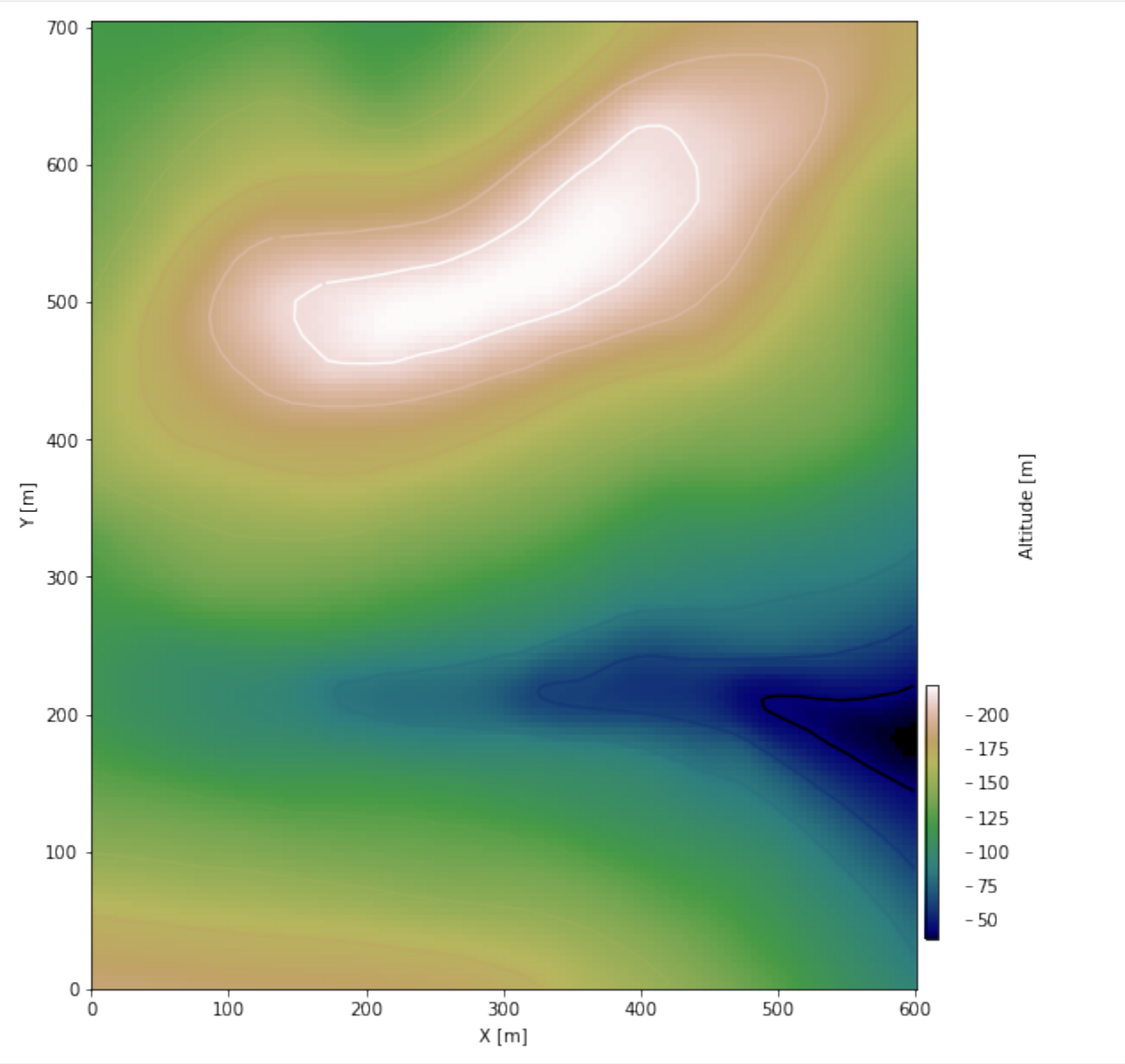
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0, 601, 0, 705], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[8]:
```
(0.0, 705.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t iff(raster = topo_r aster, path = file_p ath +' raster31.tif', extent = [0, 601, 0, 705], crs ='EPSG : 4326', overwrite_f ile = True)$

Opening Raster

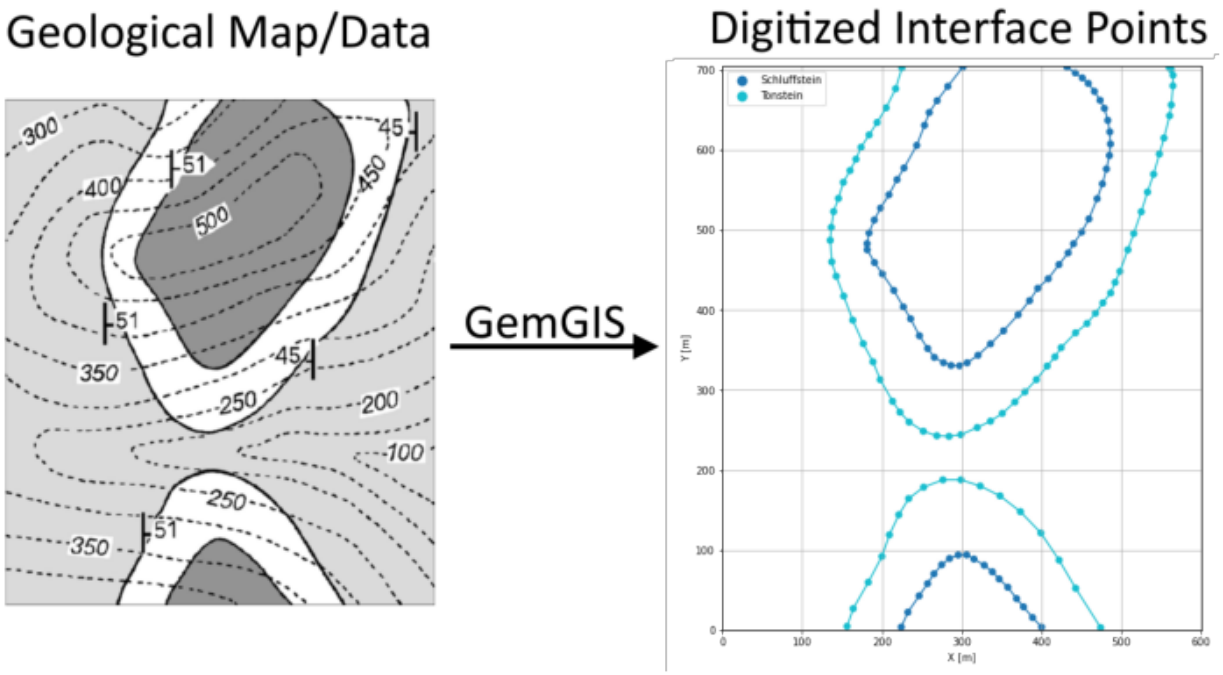
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster31.tif')
```

7.31.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../images/interfaces_example31.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces31.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Schluffstein	LINESTRING (224.233 3.618, 232.559 22.201, 246...
1	None	Tonstein	LINESTRING (156.296 4.583, 164.019 26.787, 182...
2	None	Tonstein	LINESTRING (225.078 702.899, 217.234 676.834, ...
3	None	Schluffstein	LINESTRING (301.462 703.864, 282.637 679.730, ...

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Tonstein',
↪'Schluffstein'])]
interfaces_coords
```

[12]:

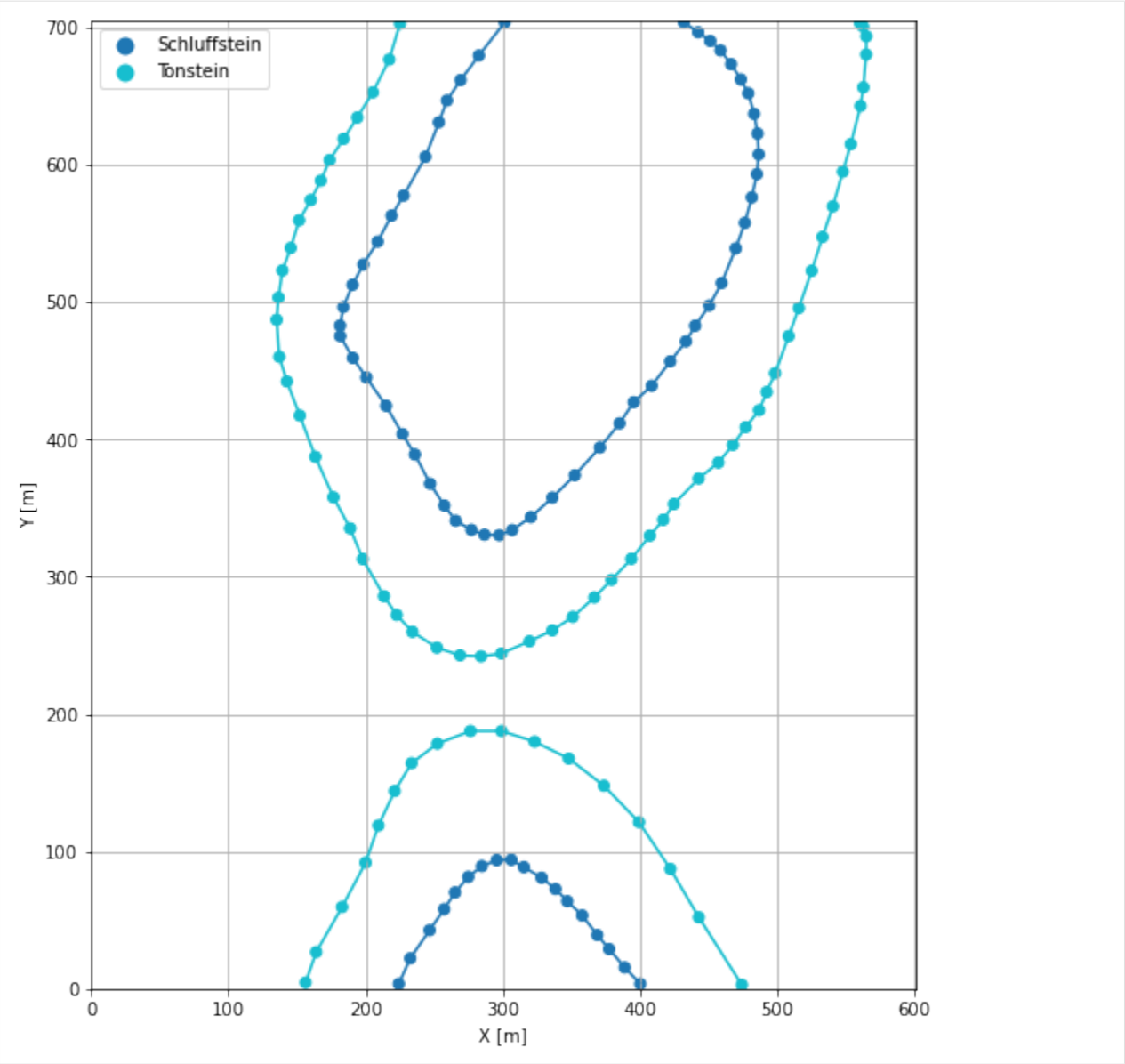
	formation	geometry	X	Y	Z
0	Schluffstein	POINT (224.233 3.618)	224.23	3.62	180.40
1	Schluffstein	POINT (232.559 22.201)	232.56	22.20	172.51
2	Schluffstein	POINT (246.677 42.715)	246.68	42.72	161.16
3	Schluffstein	POINT (257.296 58.040)	257.30	58.04	151.99
4	Schluffstein	POINT (265.261 70.228)	265.26	70.23	144.44
..	...	...	...	...	...
138	Schluffstein	POINT (466.538 673.456)	466.54	673.46	193.52
139	Schluffstein	POINT (458.815 683.471)	458.81	683.47	189.84
140	Schluffstein	POINT (451.333 690.349)	451.33	690.35	186.16
141	Schluffstein	POINT (442.524 696.624)	442.52	696.62	183.49
142	Schluffstein	POINT (432.026 704.106)	432.03	704.11	180.41

[143 rows x 5 columns]

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[13]:
```
(0.0, 705.0)
```

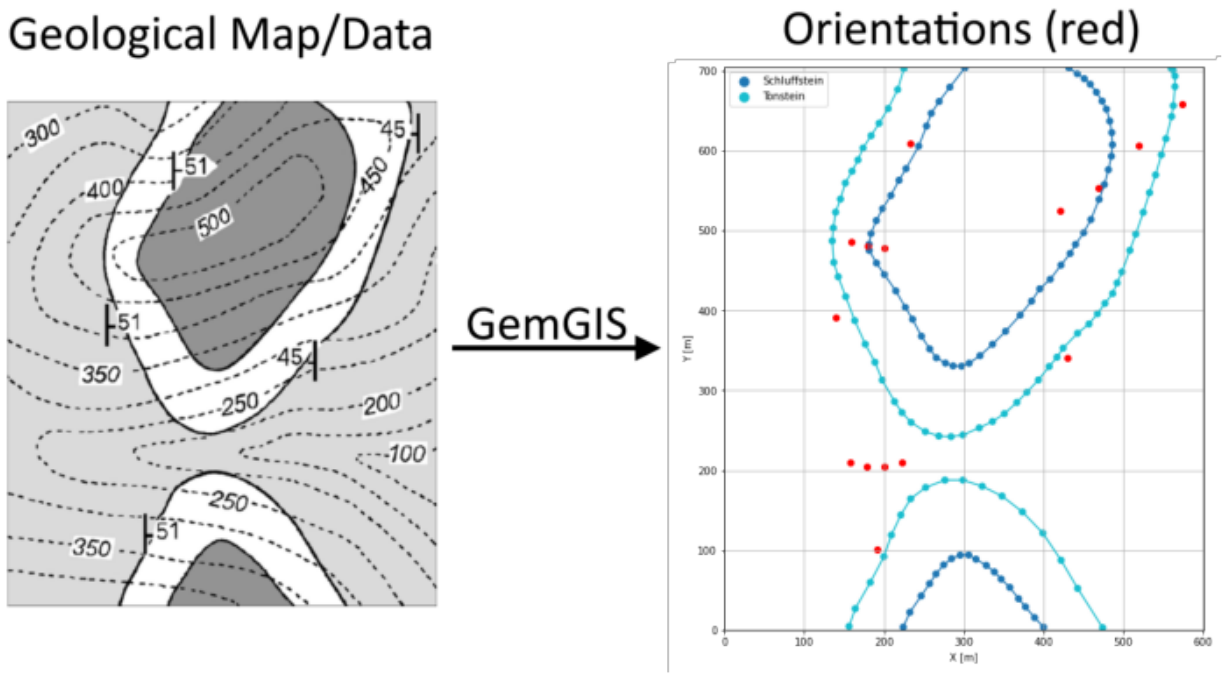


7.31.6 Orientations from Strike Lines and Map

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

In addition, orientations were provided on the map which were digitized as points and can be used right away.

[14]:
```
img = mpimg.imread('../images/orientations_example31.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



Orientations from Map

[15]:
```
orientations_points = gpd.read_file(file_path + 'orientations31.shp')
orientations_points = gg.vector.extract_xyz(gdf=orientations_points, dem=topo_raster)
orientations_points
```

[15]:

	formation	dip	azimuth	polarity	geometry	X	Y	\
0	Tonstein	51.00	90.00	1.00	POINT (139.161 391.089)	139.16	391.09	
1	Tonstein	45.00	270.00	1.00	POINT (430.216 340.890)	430.22	340.89	
2	Tonstein	51.00	90.00	1.00	POINT (192.014 101.481)	192.01	101.48	
3	Tonstein	51.00	90.00	1.00	POINT (232.559 608.777)	232.56	608.78	
4	Tonstein	45.00	270.00	1.00	POINT (574.296 658.493)	574.30	658.49	

	Z
0	175.58
1	112.43
2	136.37
3	160.57
4	179.60

Orientations from Strike Lines

[16]:
```
strikes = gpd.read_file(file_path + 'strikes31.shp')
strikes['Z'] = strikes['Z']*0.425
strikes
```

[16]:

	id	formation	Z	geometry
0	1	Tonstein	106.25	LINESTRING (234.450 260.524, 234.691 164.713)
1	2	Tonstein	127.50	LINESTRING (210.799 289.485, 210.799 124.168)
2	3	Tonstein	148.75	LINESTRING (190.044 331.719, 190.044 74.934)
3	4	Tonstein	170.00	LINESTRING (168.564 374.919, 168.082 36.561)
4	5	Tonstein	191.25	LINESTRING (148.533 425.842, 143.465 3.498)
5	4	Tonstein1	191.25	LINESTRING (148.775 425.600, 149.499 550.132)
6	3	Tonstein1	170.00	LINESTRING (169.288 592.849, 168.806 374.437)
7	2	Tonstein1	148.75	LINESTRING (190.285 628.326, 190.285 330.513)
8	1	Tonstein1	127.50	LINESTRING (210.558 663.561, 211.040 289.968)
9	1	Schluffstein5	148.75	LINESTRING (261.239 62.867, 257.860 350.544)
10	2	Schluffstein5	170.00	LINESTRING (237.105 28.356, 235.898 386.504)
11	1	Schluffstein4	148.75	LINESTRING (364.291 42.836, 365.739 390.365)
12	3	Schluffstein5	191.25	LINESTRING (213.695 425.600, 215.143 2.291)
13	4	Schluffstein5	212.50	LINESTRING (191.974 456.009, 193.422 4.705)
14	2	Schluffstein4	170.00	LINESTRING (421.247 453.113, 423.178 2.774)
15	3	Schluffstein4	191.25	LINESTRING (463.240 520.206, 463.240 0.843)
16	4	Schluffstein1	212.50	LINESTRING (192.457 516.827, 191.974 454.803)
17	3	Schluffstein1	191.25	LINESTRING (214.902 555.200, 213.695 425.600)
18	2	Schluffstein1	170.00	LINESTRING (237.588 595.021, 235.657 387.710)
19	1	Schluffstein1	148.75	LINESTRING (257.136 640.875, 257.136 349.579)
20	3	Schluffstein2	191.25	LINESTRING (461.309 679.972, 463.240 520.447)
21	2	Schluffstein2	170.00	LINESTRING (421.005 454.079, 419.557 701.934)
22	1	Schluffstein2	148.75	LINESTRING (365.739 390.365, 364.773 703.865)
23	1	Schluffstein3	148.75	LINESTRING (257.136 640.151, 258.101 351.027)
24	2	Schluffstein3	148.75	LINESTRING (365.256 389.158, 364.773 703.865)
25	4	Tonstein2	170.00	LINESTRING (542.882 580.058, 544.813 705.072)
26	3	Tonstein2	148.75	LINESTRING (493.649 436.702, 495.579 703.624)
27	2	Tonstein2	127.50	LINESTRING (443.933 371.540, 444.898 703.624)
28	1	Tonstein2	106.25	LINESTRING (399.044 319.411, 397.596 704.589)

Calculating Orientations for each formation

[17]:
```
orientations_claystone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Tonstein1'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone1
```

[17]:

	dip	azimuth	Z	geometry	polarity	formation	X	Y
0	46.28	89.95	138.12	POINT (200.542 478.092)	1.00	Tonstein1	200.54	478.09
1	45.34	90.04	159.38	POINT (179.666 481.531)	1.00	Tonstein1	179.67	481.53
2	47.17	90.18	180.62	POINT (159.092 485.754)	1.00	Tonstein1	159.09	485.75

[18]:
```
orientations_claystone = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Tonstein'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity formation      X  \
0 41.94    89.96 116.88  POINT (222.685 209.722)      1.00  Tonstein 222.68
1 45.67    90.00 138.12  POINT (200.421 205.077)      1.00  Tonstein 200.42
2 44.61    90.05 159.38  POINT (179.183 204.534)      1.00  Tonstein 179.18
3 45.83    90.45 180.62  POINT (157.161 210.205)      1.00  Tonstein 157.16

       Y
0 209.72
1 205.08
2 204.53
3 210.21
```

[19]:
```
orientations_claystone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Tonstein2'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone2
```

[19]:
```
    dip  azimuth      Z                 geometry  polarity  formation      X  \
0 25.24   269.95 116.88  POINT (421.368 524.791)      1.00  Tonstein2 421.37
1 23.22   270.26 138.12  POINT (469.515 553.872)      1.00  Tonstein2 469.51
2 23.79   270.50 159.38  POINT (519.231 606.364)      1.00  Tonstein2 519.23

       Y
0 524.79
1 553.87
2 606.36
```

[20]:
```
orientations_siltstone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Schluffstein1'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_siltstone1
```

[20]:
```
    dip  azimuth      Z                 geometry  polarity      formation  \
0 47.39    90.18 159.38  POINT (246.879 493.296)      1.00  Schluffstein1
1 43.60    90.53 180.62  POINT (225.460 490.883)      1.00  Schluffstein1
2 44.02    90.52 201.88  POINT (203.257 488.107)      1.00  Schluffstein1

       X      Y
0 246.88 493.30
1 225.46 490.88
2 203.26 488.11
```

[21]:
```
orientations_siltstone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Schluffstein2'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_siltstone2
```

[21]:
```
    dip  azimuth      Z                 geometry  polarity      formation  \
0 21.20   269.76 159.38  POINT (392.769 562.561)      1.00  Schluffstein2
```

(continues on next page)

(continued from previous page)

```
1 27.05   269.56 180.62  POINT (441.278 589.108)      1.00  Schluffstein2

       X      Y
0 392.77 562.56
1 441.28 589.11
```

[22]:
```
orientations_siltstone3 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Schluffstein3'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_siltstone3
```

[22]:
```
   dip  azimuth      Z                 geometry  polarity      formation  \
0 0.00     0.00 148.75  POINT (311.317 521.050)      1.00  Schluffstein3

       X      Y
0 311.32 521.05
```

[23]:
```
orientations_siltstone4 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Schluffstein4'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_siltstone4
```

[23]:
```
    dip  azimuth      Z                 geometry  polarity      formation  \
0 20.86   269.93 159.38  POINT (393.613 222.272)      1.00  Schluffstein4
1 27.94   269.89 180.62  POINT (442.726 244.234)      1.00  Schluffstein4

       X      Y
0 393.61 222.27
1 442.73 244.23
```

[24]:
```
orientations_siltstone5 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Schluffstein5'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_siltstone5
```

[24]:
```
    dip  azimuth      Z                 geometry  polarity      formation  \
0 44.21    89.62 159.38  POINT (248.025 207.068)      1.00  Schluffstein5
1 43.94    89.81 180.62  POINT (225.460 210.688)      1.00  Schluffstein5
2 44.50    89.81 201.88  POINT (203.559 222.151)      1.00  Schluffstein5

       X      Y
0 248.03 207.07
1 225.46 210.69
2 203.56 222.15
```

Merging Orientations

[25]:
```
import pandas as pd
orientations = pd.concat([orientations_points, orientations_claystone, orientations_
↪claystone1, orientations_claystone2, orientations_siltstone1, orientations_siltstone2,␣
↪orientations_siltstone3, orientations_siltstone4, orientations_siltstone5])
orientations['formation'] = ['Tonstein', 'Tonstein', 'Tonstein', 'Tonstein', 'Tonstein',
↪'Tonstein', 'Tonstein', 'Tonstein', 'Tonstein', 'Tonstein','Tonstein', 'Tonstein',
↪'Tonstein','Tonstein', 'Tonstein', 'Siltstone', 'Siltstone', 'Siltstone', 'Siltstone',
↪'Siltstone', 'Siltstone', 'Siltstone', 'Siltstone', 'Siltstone', 'Siltstone',
↪'Siltstone', ]
orientations = orientations[orientations['formation'].isin(['Tonstein', 'Schluffstein
↪'])].reset_index()
orientations
```

[25]:

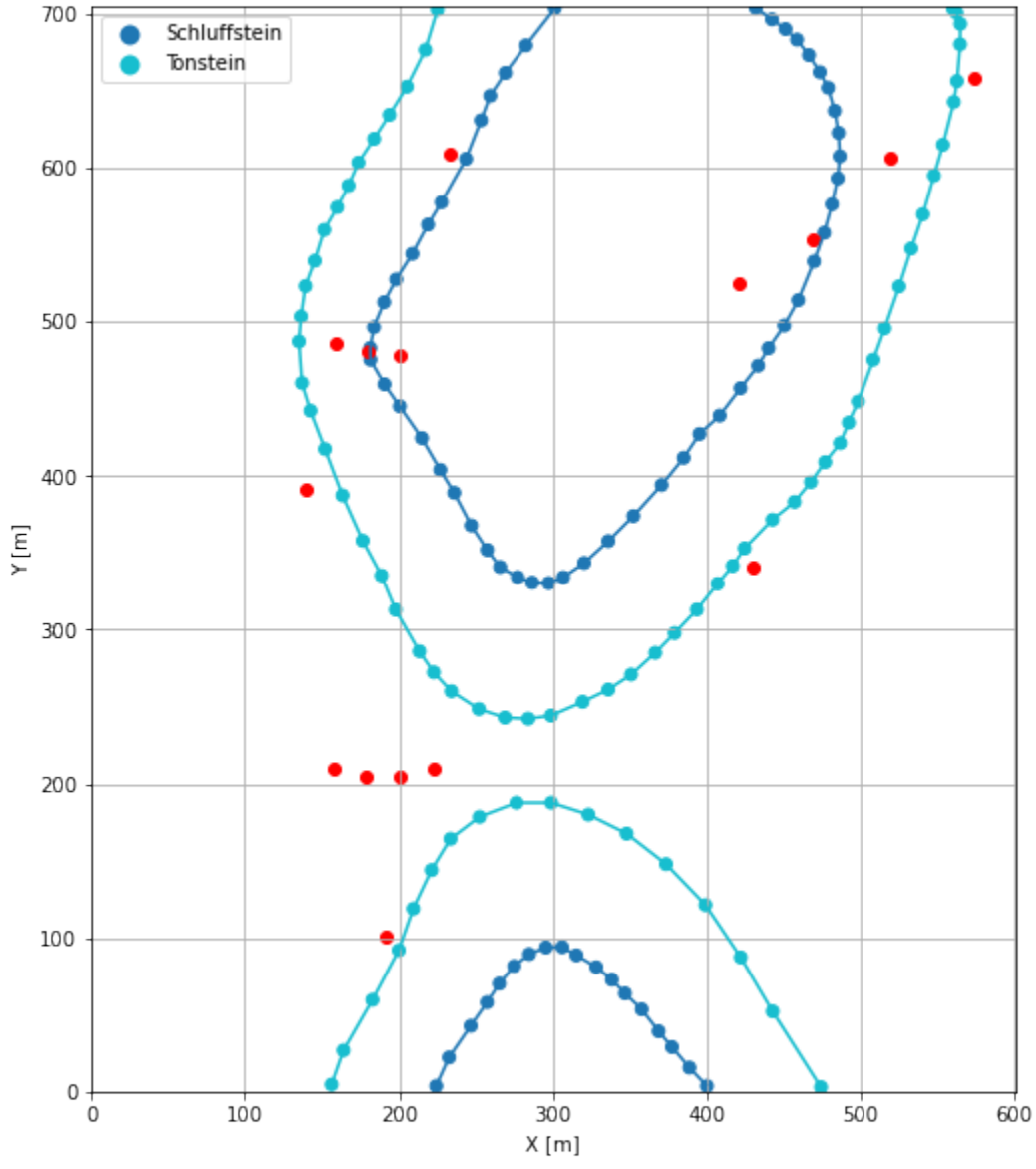
	index	formation	dip	azimuth	polarity	geometry	X	\
0	0	Tonstein	51.00	90.00	1.00	POINT (139.161 391.089)	139.16	
1	1	Tonstein	45.00	270.00	1.00	POINT (430.216 340.890)	430.22	
2	2	Tonstein	51.00	90.00	1.00	POINT (192.014 101.481)	192.01	
3	3	Tonstein	51.00	90.00	1.00	POINT (232.559 608.777)	232.56	
4	4	Tonstein	45.00	270.00	1.00	POINT (574.296 658.493)	574.30	
5	0	Tonstein	41.94	89.96	1.00	POINT (222.685 209.722)	222.68	
6	1	Tonstein	45.67	90.00	1.00	POINT (200.421 205.077)	200.42	
7	2	Tonstein	44.61	90.05	1.00	POINT (179.183 204.534)	179.18	
8	3	Tonstein	45.83	90.45	1.00	POINT (157.161 210.205)	157.16	
9	0	Tonstein	46.28	89.95	1.00	POINT (200.542 478.092)	200.54	
10	1	Tonstein	45.34	90.04	1.00	POINT (179.666 481.531)	179.67	
11	2	Tonstein	47.17	90.18	1.00	POINT (159.092 485.754)	159.09	
12	0	Tonstein	25.24	269.95	1.00	POINT (421.368 524.791)	421.37	
13	1	Tonstein	23.22	270.26	1.00	POINT (469.515 553.872)	469.51	
14	2	Tonstein	23.79	270.50	1.00	POINT (519.231 606.364)	519.23	

	Y	Z
0	391.09	175.58
1	340.89	112.43
2	101.48	136.37
3	608.78	160.57
4	658.49	179.60
5	209.72	116.88
6	205.08	138.12
7	204.53	159.38
8	210.21	180.62
9	478.09	138.12
10	481.53	159.38
11	485.75	180.62
12	524.79	116.88
13	553.87	138.12
14	606.36	159.38

Plotting the Orientations

[26]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[26]:
```
(0.0, 705.0)
```



7.31.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[27]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[28]:
```
geo_model = gp.create_model('Model31')
geo_model
```

[28]:
```
Model31  2022-04-17 14:43
```

Initiate Data

[29]:
```
gp.init_data(geo_model, [0, 601, 0, 705, 0, 300], [100, 100, 100],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[29]:
```
Model31  2022-04-17 14:43
```

Model Surfaces

[30]:
```
geo_model.surfaces
```

[30]:

	surface	series	order_surfaces	color	id
0	Schluffstein	Default series	1	#015482	1
1	Tonstein	Default series	2	#9f0052	2

Mapping the Stack to Surfaces

[31]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata1': ('Schluffstein', 'Tonstein'),
                          },
                         remove_unused_series=True)
geo_model.add_surfaces('Sandstein')
```

[31]:
```
        surface   series  order_surfaces    color  id
0  Schluffstein  Strata1               1  #015482   1
1      Tonstein  Strata1               2  #9f0052   2
2     Sandstein  Strata1               3  #ffbe00   3
```

Showing the Number of Data Points

[32]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[32]:
```
        surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0  Schluffstein  Strata1               1  #015482   1                 70               ␣
↪    0
1      Tonstein  Strata1               2  #9f0052   2                 73               ␣
↪   15
2     Sandstein  Strata1               3  #ffbe00   3                  0               ␣
↪    0
```

Loading Digital Elevation Model

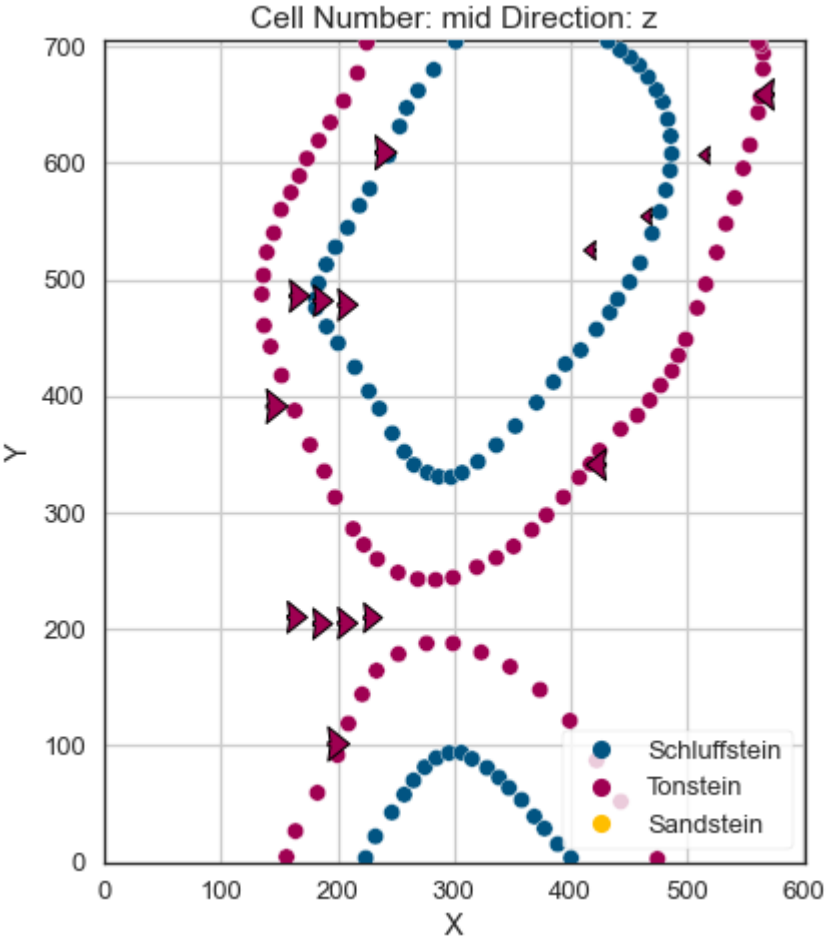
[33]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster31.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

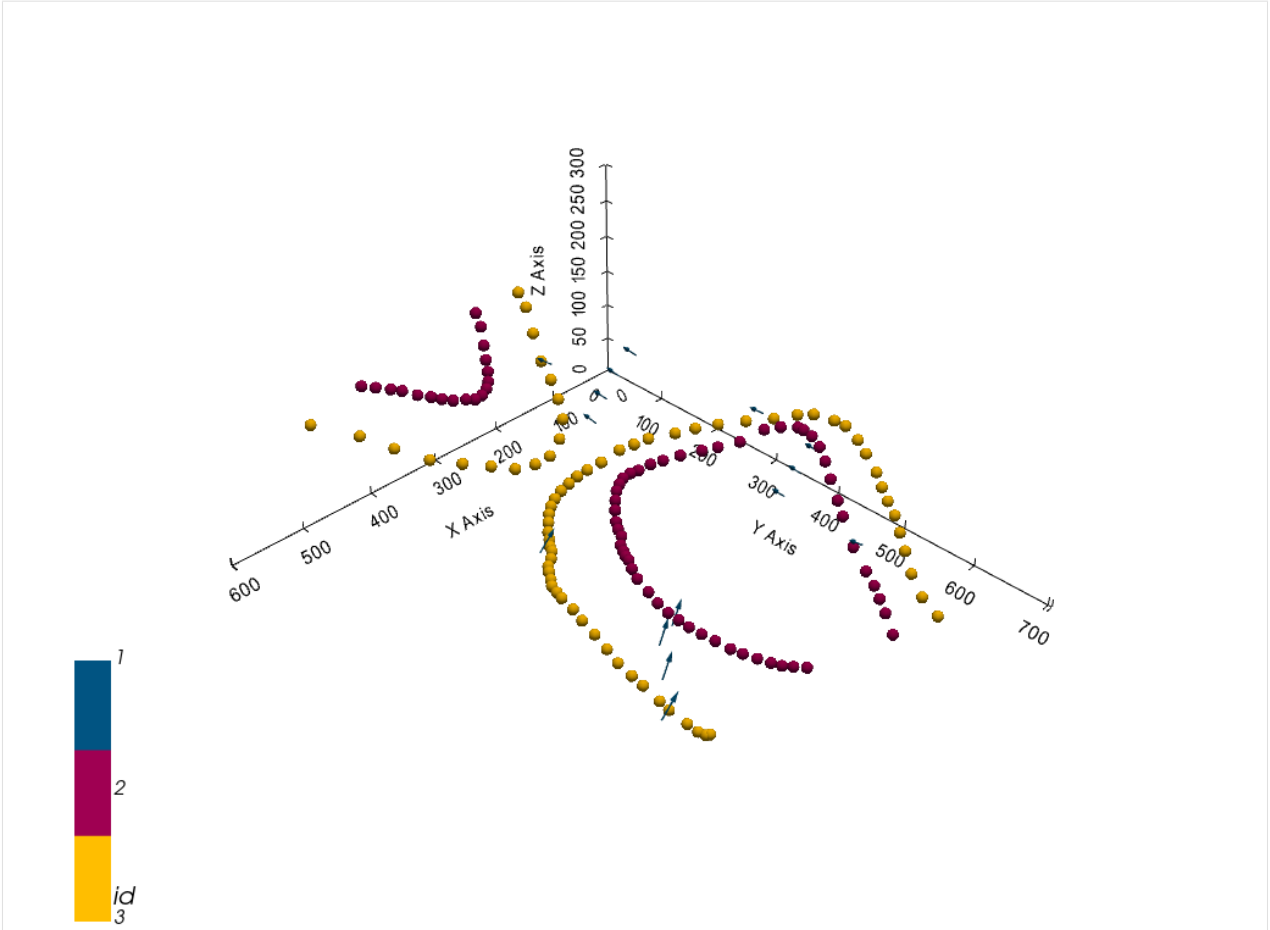
[33]:
```
Grid Object. Values:
array([[  3.005     ,   3.525     ,   1.5       ],
       [  3.005     ,   3.525     ,   4.5       ],
       [  3.005     ,   3.525     ,   7.5       ],
       ...,
       [598.49583333, 692.5       , 175.96000671],
       [598.49583333, 697.5       , 176.22775269],
       [598.49583333, 702.5       , 176.43305969]])
```

Plotting Input Data

[34]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[35]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[35]:
```
<gempy.plot.vista.GemPyToVista at 0x1e7ef0f5d30>
```

Setting the Interpolator

[36]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                   values
range              973.77
$C_o$            22576.81
drift equations       [3]
```

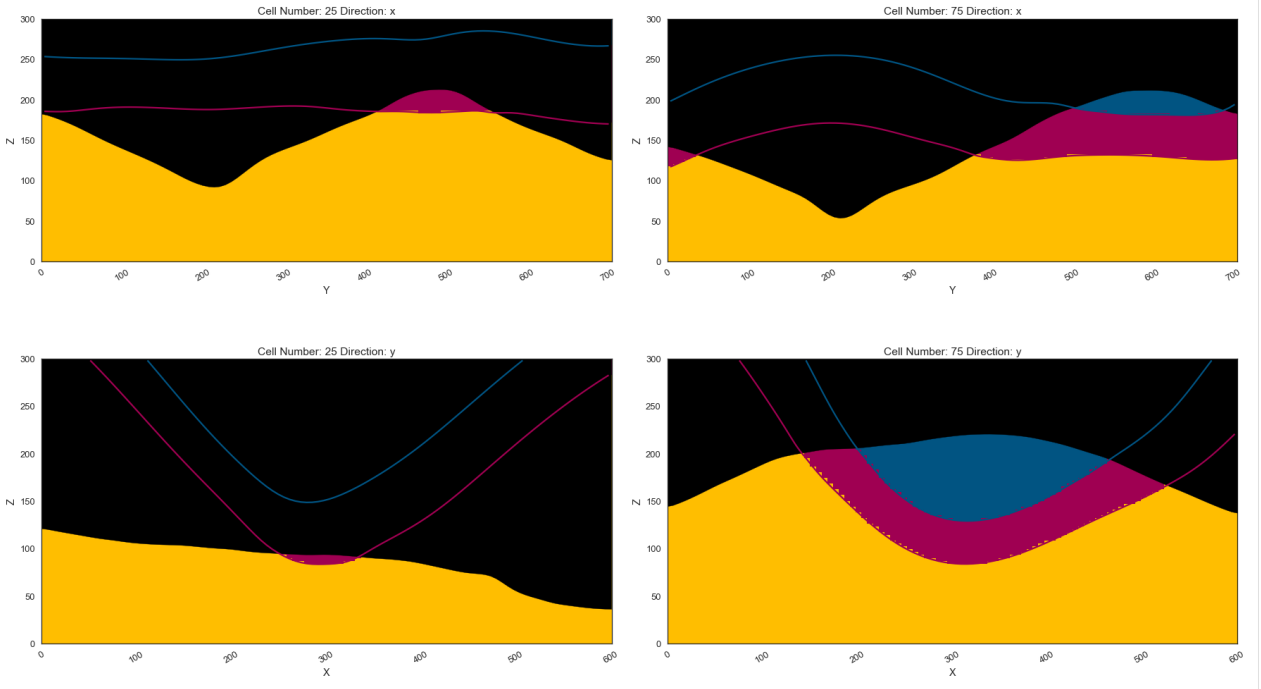
[36]:
```
<gempy.core.interpolator.InterpolatorModel at 0x1e7e8161e50>
```

Computing Model

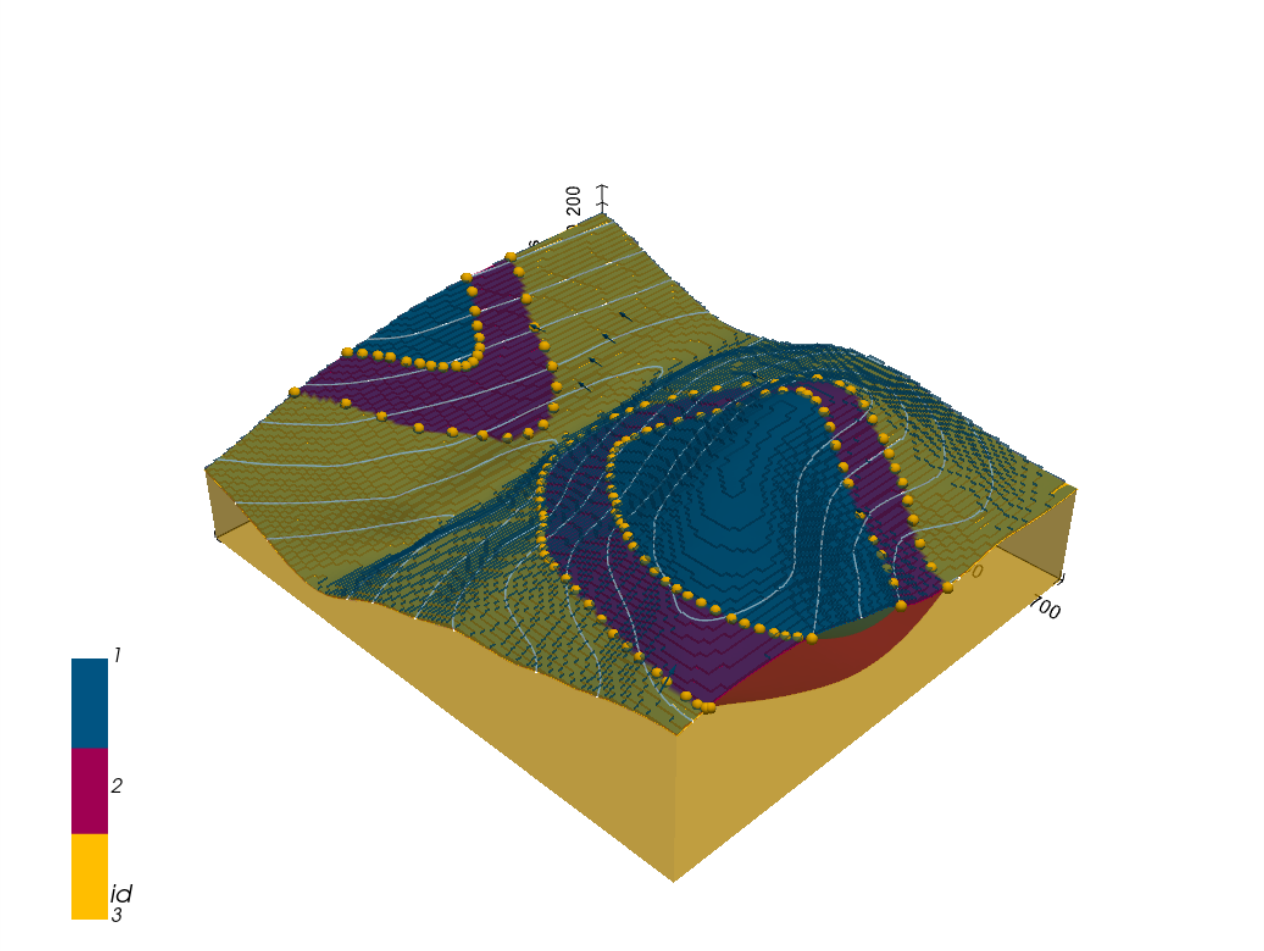
[37]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Plotting Cross Sections

[38]:
```
fig = gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75],
↪ show_topography=True, show_data=False)
```



[39]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



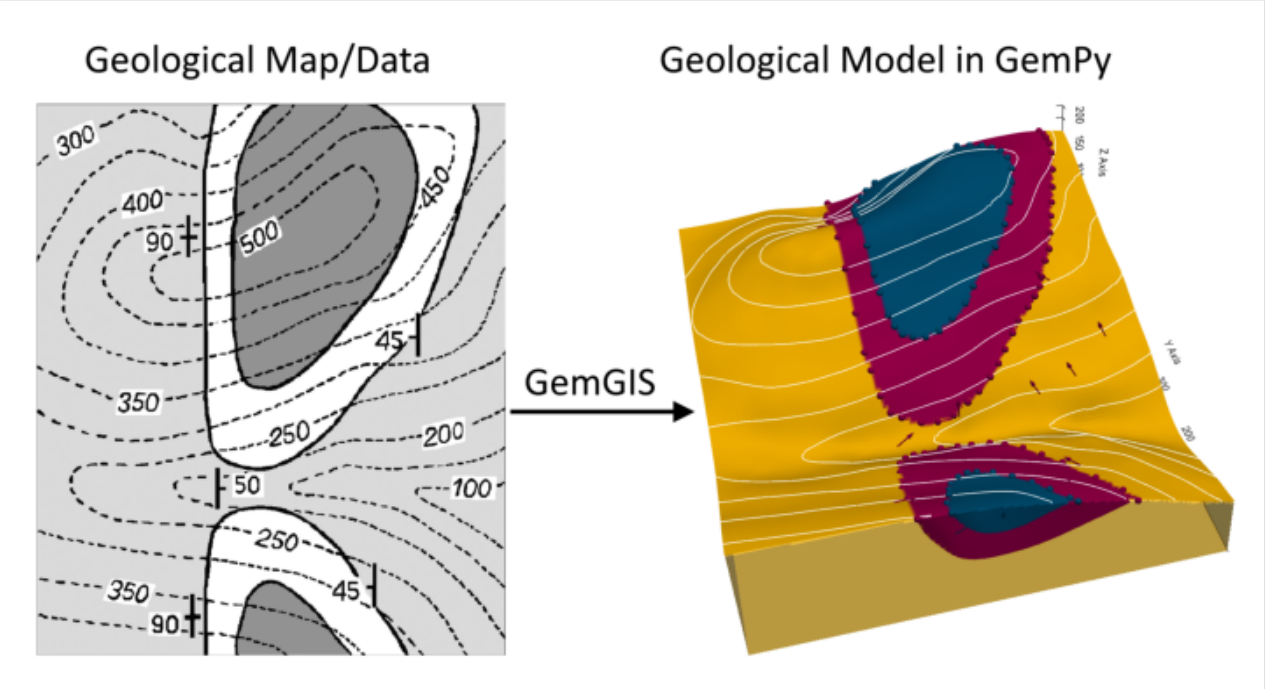
[ ]:

7.32 Example 32 - Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 601 m wide (W-E extent) and 705 m high (N-S extent). The vertical model extents varies between 0 m and 300 m. The model represents two folded stratigraphic units (blue and red) above an unspecified basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and were used to calculate orientations for the `GemPy` model. These will be loaded into the model directly. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example32.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.32.1 Licensing

Computational Geosciences and Reservoir Engineering, RWTH Aachen University, Authors: Alexander Juestel. For more information contact: alexander.juestel(at)rwth-aachen.de

This work is licensed under a Creative Commons Attribution 4.0 International License (http://creativecommons.org/licenses/by/4.0/)

7.32.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.32.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

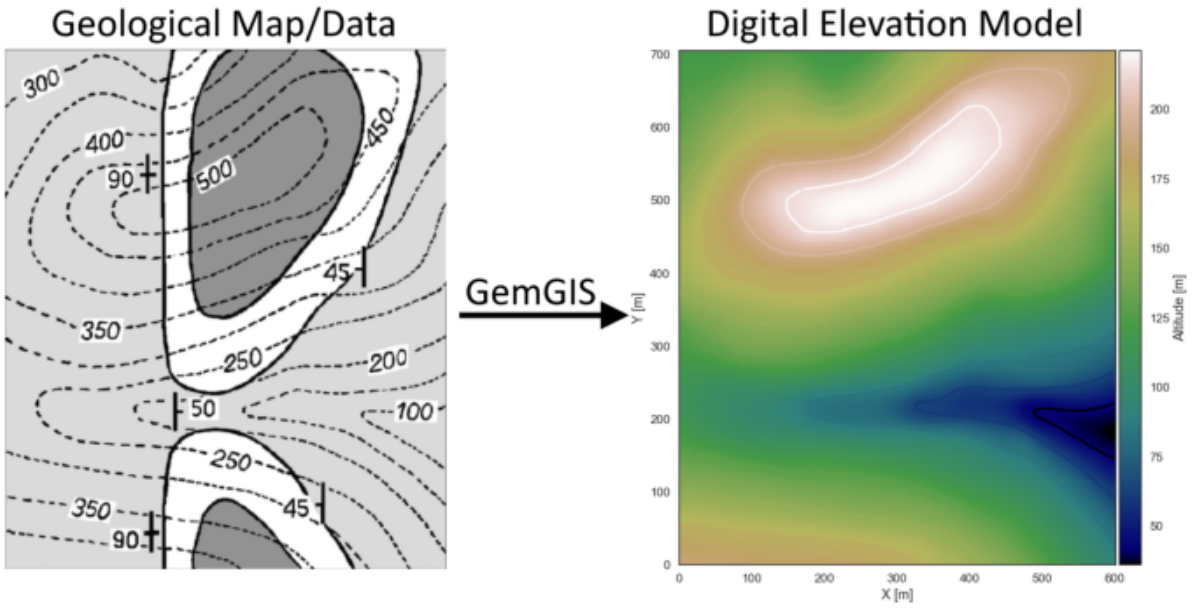
[4]:
```
file_path = 'data/example32/'
gg.download_gemgis_data.download_tutorial_data(filename="example32_folded_layers.zip",␣
↪dirpath=file_path)
```

```
Downloading file 'example32_folded_layers.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample32_folded_layers.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\example\data\example32'.
```

7.32.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
img = mpimg.imread('../images/dem_example32.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo32.shp')
topo['Z'] = topo['Z']*0.425
topo.head()
```

[6]:

	id	Z	geometry
0	None	170.00	LINESTRING (0.994 52.851, 36.954 48.266, 68.32...
1	None	148.75	LINESTRING (1.235 98.706, 42.022 94.603, 80.39...
2	None	127.50	LINESTRING (1.477 157.955, 44.435 147.818, 81...
3	None	127.50	LINESTRING (1.597 320.738, 11.492 310.722, 22...
4	None	106.25	LINESTRING (553.299 3.497, 534.233 30.527, 513...

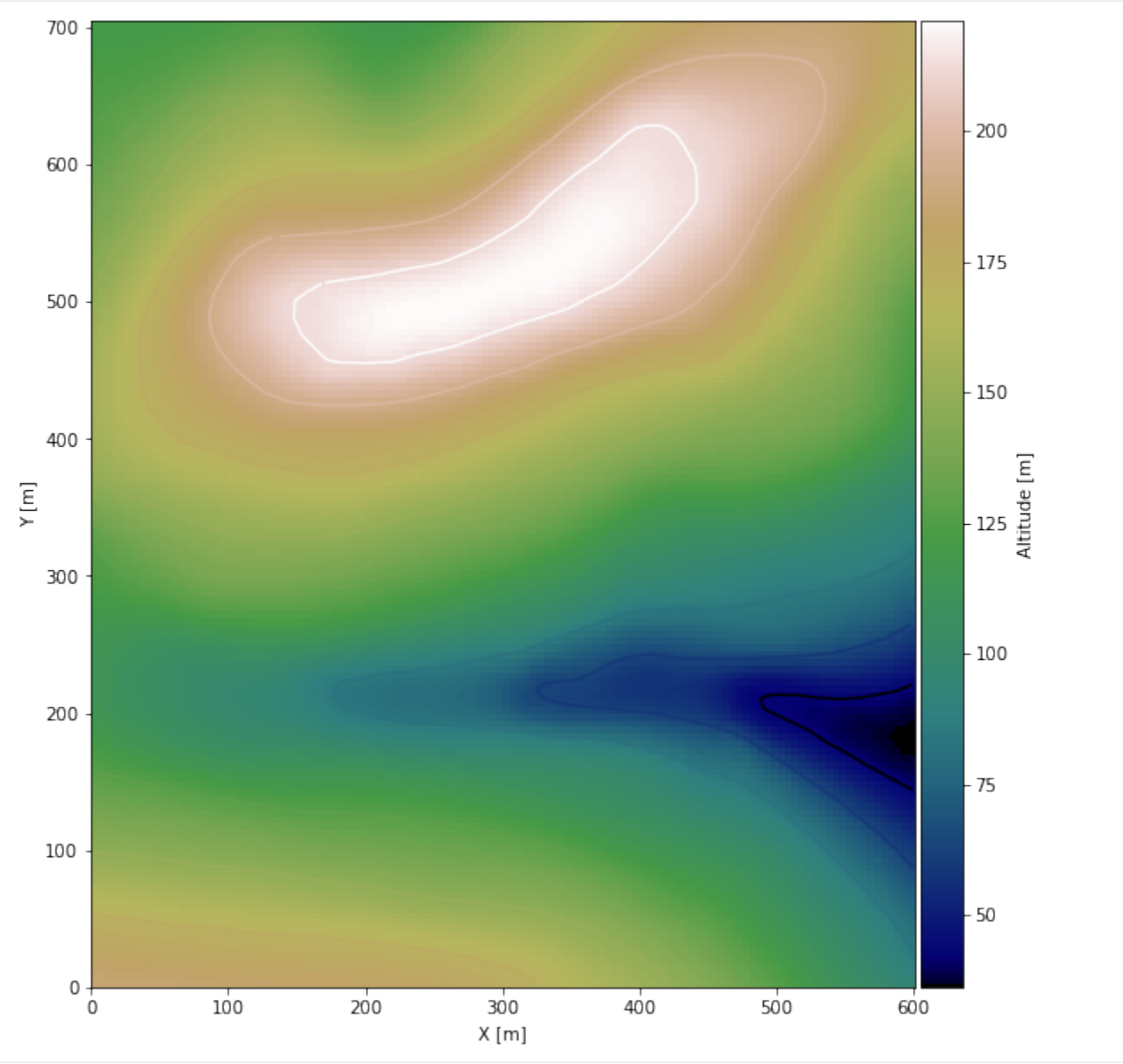
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt

from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0, 601, 0, 705], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
ax.set_xlim(0, 601)
ax.set_ylim(0, 705)
```

[8]:
```
(0.0, 705.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

gg.raster.save$_a s_t i f f(raster = topo_r aster, path = file_p ath +' raster32.tif', extent = [0, 601, 0, 705], crs ='EPSG : 4326', overwrite_f ile = True)$

Opening Raster

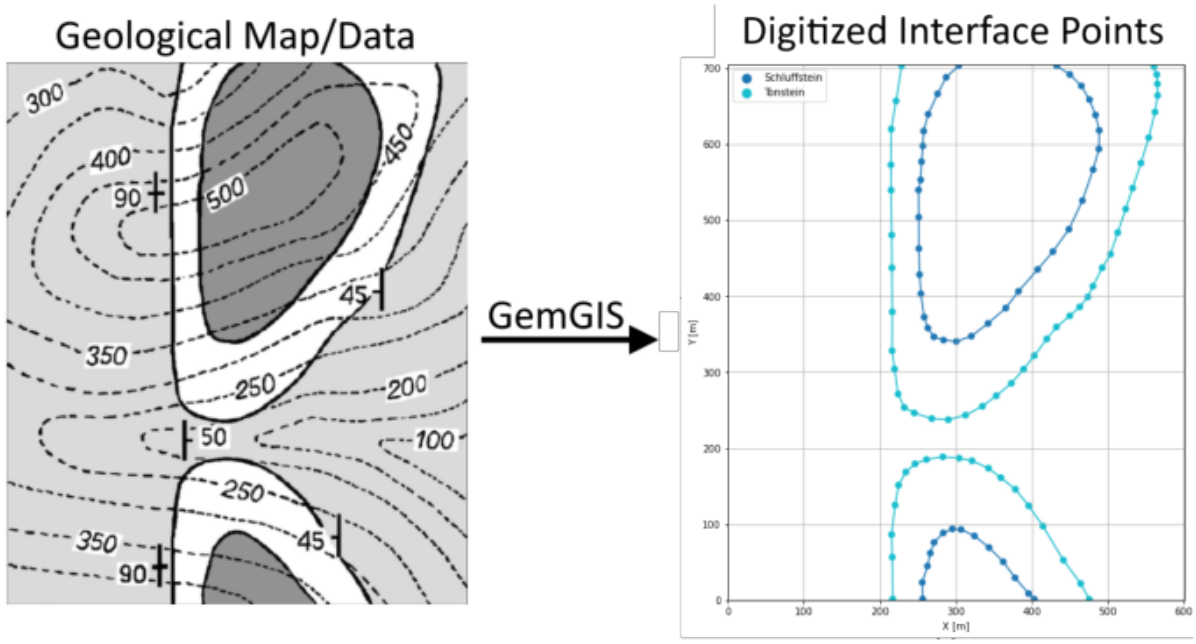
The previously computed and saved raster can now be opened using rasterio.

[9]:
```
topo_raster = rasterio.open(file_path + 'raster32.tif')
```

7.32.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[10]:
```
img = mpimg.imread('../images/interfaces_example32.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[11]:
```
interfaces = gpd.read_file(file_path + 'interfaces32.shp')
interfaces.head()
```

[11]:

	id	formation	geometry
0	None	Schluffstein	LINESTRING (256.467 1.534, 255.813 23.321, 262...
1	None	Tonstein	LINESTRING (217.032 1.534, 216.378 56.656, 215...
2	None	Tonstein	LINESTRING (228.797 703.524, 221.389 657.117, ...
3	None	Schluffstein	LINESTRING (303.746 703.307, 287.405 688.273, ...

Extracting Z coordinate from Digital Elevation Model

[12]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Tonstein',
↪'Schluffstein'])]
interfaces_coords
```

[12]:

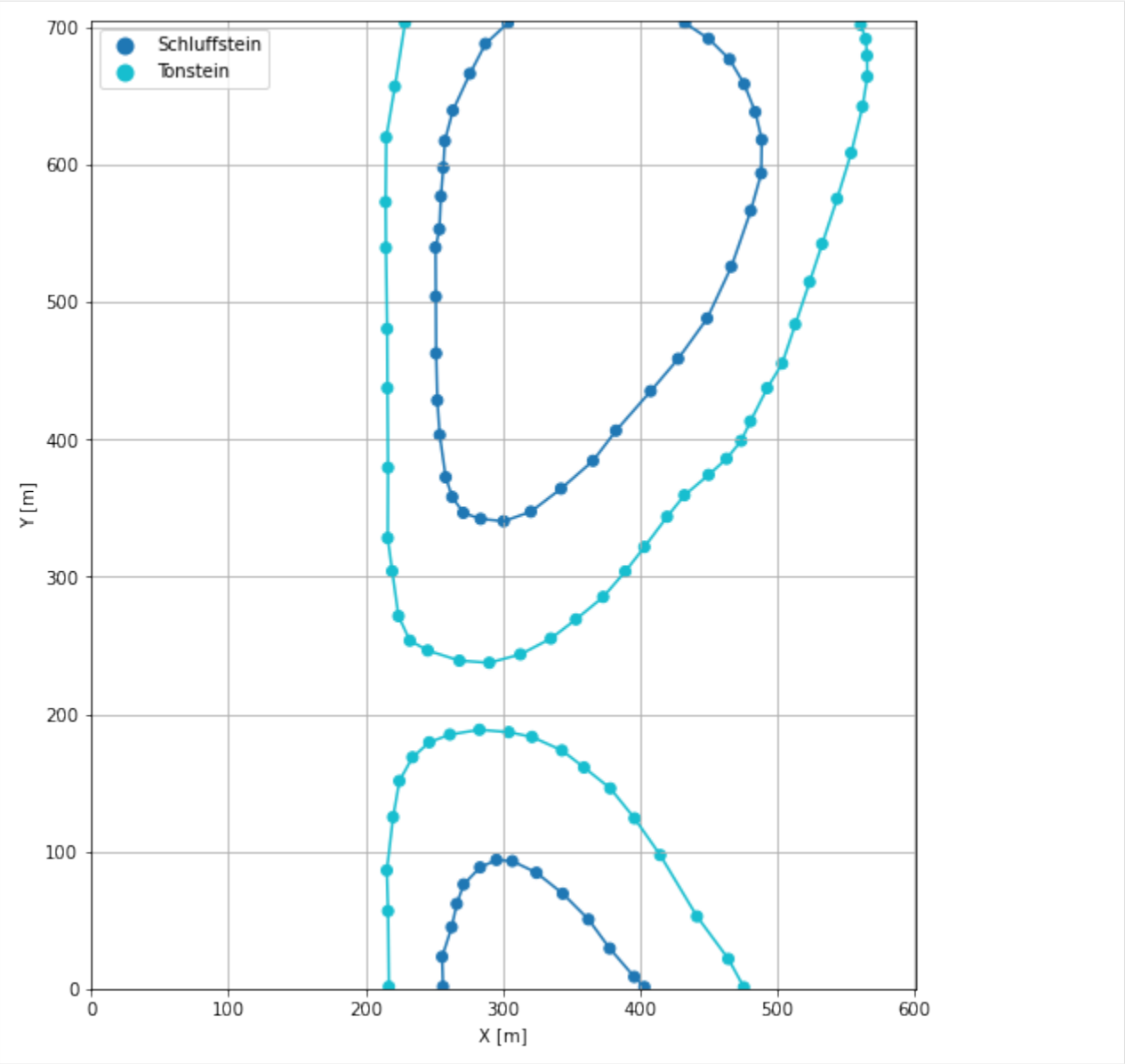
	formation	geometry	X	Y	Z
0	Schluffstein	POINT (256.467 1.534)	256.47	1.53	178.38
1	Schluffstein	POINT (255.813 23.321)	255.81	23.32	170.99
2	Schluffstein	POINT (262.785 44.891)	262.79	44.89	160.10
3	Schluffstein	POINT (266.489 61.885)	266.49	61.89	148.84
4	Schluffstein	POINT (271.500 76.047)	271.50	76.05	141.98
..	...	...	...	...	...
101	Schluffstein	POINT (484.145 638.598)	484.15	638.60	202.71
102	Schluffstein	POINT (476.084 658.860)	476.08	658.86	198.10
103	Schluffstein	POINT (465.408 676.944)	465.41	676.94	191.75
104	Schluffstein	POINT (450.157 691.977)	450.16	691.98	185.69
105	Schluffstein	POINT (432.945 703.307)	432.94	703.31	180.41

[106 rows x 5 columns]

Plotting the Interface Points

[13]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[13]:
```
(0.0, 705.0)
```

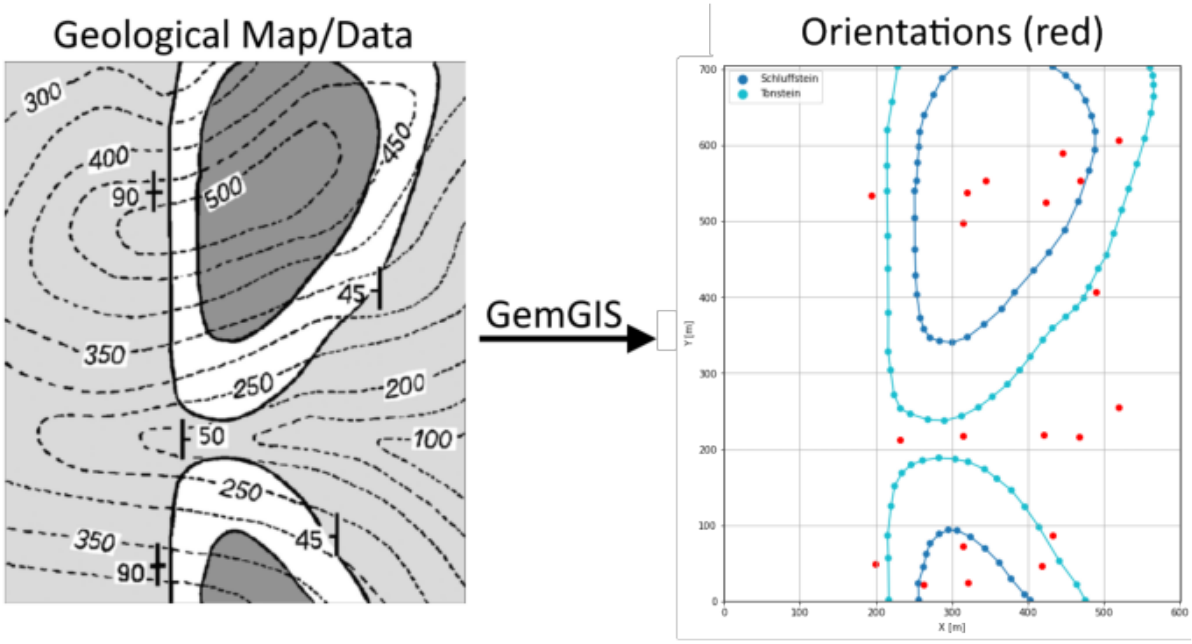


7.32.6 Orientations from Strike Lines and Map

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

In addition, orientations were provided on the map which were digitized as points and can be used right away.

[14]:
```
img = mpimg.imread('../images/orientations_example32.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



Orientations from Map

[15]:
```
orientations_points = gpd.read_file(file_path + 'orientations32.shp')
orientations_points = gg.vector.extract_xyz(gdf=orientations_points, dem=topo_raster)
orientations_points
```

[15]:

	formation	dip	azimuth	polarity	geometry	X	Y	Z
0	Tonstein	89.00	90.00	1.00	POINT (194.155 534.236)	194.15	534.24	203.22
1	Tonstein	45.00	270.00	1.00	POINT (489.592 406.562)	489.59	406.56	140.53
2	Tonstein	50.00	90.00	1.00	POINT (231.629 212.654)	231.63	212.65	80.31
3	Tonstein	89.00	90.00	1.00	POINT (199.384 48.813)	199.38	48.81	162.43
4	Tonstein	45.00	270.00	1.00	POINT (433.381 87.158)	433.38	87.16	116.59

Orientations from Strike Lines

[16]:
```
strikes = gpd.read_file(file_path + 'strikes32.shp')
strikes['Z'] = strikes['Z']*0.425
strikes
```

[16]:

	id	formation	Z	geometry
0	5	Tonstein1	170.00	LINESTRING (546.457 583.476, 546.022 704.178)
1	4	Tonstein1	148.75	LINESTRING (494.168 436.411, 492.425 705.050)
2	3	Tonstein1	127.50	LINESTRING (447.543 371.266, 443.621 703.742)
3	2	Tonstein1	106.25	LINESTRING (401.353 319.848, 400.482 705.050)
4	1	Tonstein1	106.25	LINESTRING (228.797 259.715, 227.054 704.178)
5	1	Tonstein2	106.25	LINESTRING (230.976 163.414, 230.104 261.022)
6	2	Tonstein2	106.25	LINESTRING (401.789 321.155, 396.560 126.376)
7	3	Tonstein2	127.50	LINESTRING (447.978 372.138, 438.828 57.528)
8	4	Tonstein2	148.75	LINESTRING (494.168 436.629, 491.117 1.316)
9	5	Tonstein2	170.00	LINESTRING (547.329 586.090, 545.586 -0.863)
10	1	Tonstein3	106.25	LINESTRING (231.847 162.107, 231.847 -0.427)
11	2	Tonstein3	106.25	LINESTRING (396.996 125.940, 399.610 0.445)
12	3	Tonstein3	127.50	LINESTRING (438.828 57.963, 438.828 0.880)
13	2	Schluffstein1	106.25	LINESTRING (268.014 63.628, 269.321 0.880)
14	1	Schluffstein1	106.25	LINESTRING (373.901 34.869, 375.644 0.009)
15	3	Schluffstein1	127.50	LINESTRING (256.685 23.103, 257.556 0.880)
16	4	Schluffstein2	191.25	LINESTRING (466.715 675.854, 467.587 527.700)
17	3	Schluffstein2	170.00	LINESTRING (423.576 453.623, 424.012 702.871)
18	1	Schluffstein2	148.75	LINESTRING (373.030 393.054, 373.465 703.307)
19	2	Schluffstein2	148.75	LINESTRING (265.400 352.965, 266.707 702.871)

Calculating Orientations for each formation

[17]:
```
orientations_claystone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Tonstein1'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone1
```

[17]:

	dip	azimuth	Z	geometry	polarity	formation	X	Y
0	0.00	0.00	106.25	POINT (314.421 497.198)	1.00	Tonstein1	314.42	497.20
1	26.23	269.64	116.88	POINT (423.250 524.977)	1.00	Tonstein1	423.25	524.98
2	24.15	269.44	138.12	POINT (469.439 554.117)	1.00	Tonstein1	469.44	554.12
3	21.76	269.66	159.38	POINT (519.768 607.278)	1.00	Tonstein1	519.77	607.28

[18]:
```
orientations_claystone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Tonstein2'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone2
```

[18]:
```
    dip  azimuth      Z                 geometry  polarity  formation      X  \
0  0.00     0.00 106.25  POINT (314.857 217.992)      1.00  Tonstein2 314.86
1 25.65   271.63 116.88  POINT (421.289 219.299)      1.00  Tonstein2 421.29
2 24.92   270.84 138.12  POINT (468.023 216.902)      1.00  Tonstein2 468.02
3 21.95   270.25 159.38  POINT (519.550 255.793)      1.00  Tonstein2 519.55

       Y
0 217.99
1 219.30
2 216.90
3 255.79
```

[19]:
```
orientations_claystone3 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Tonstein3'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone3
```

[19]:
```
    dip  azimuth      Z                 geometry  polarity  formation      X  \
0  0.00     0.00 106.25  POINT (315.075 72.016)      1.00  Tonstein3 315.08
1 28.45   269.01 116.88  POINT (418.565 46.307)      1.00  Tonstein3 418.57

      Y
0 72.02
1 46.31
```

[20]:
```
orientations_siltstone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Schluffstein1'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_siltstone1
```

[20]:
```
    dip  azimuth      Z                 geometry  polarity      formation  \
0  0.00     0.00 106.25  POINT (321.720 24.846)      1.00  Schluffstein1
1 61.03    88.69 116.88  POINT (262.894 22.123)      1.00  Schluffstein1

       X     Y
0 321.72 24.85
1 262.89 22.12
```

[21]:
```
orientations_siltstone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Schluffstein2'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_siltstone2
```

[21]:
```
    dip  azimuth      Z                 geometry  polarity      formation  \
0  0.00     0.00 148.75  POINT (319.650 538.049)      1.00  Schluffstein2
1  7.69   270.18 159.38  POINT (344.924 553.082)      1.00  Schluffstein2
2 26.43   269.99 180.62  POINT (445.473 590.012)      1.00  Schluffstein2

       X      Y
0 319.65 538.05
1 344.92 553.08
2 445.47 590.01
```

Merging Orientations

[22]:
```
import pandas as pd
orientations = pd.concat([orientations_points, orientations_claystone1, orientations_
↪claystone2, orientations_claystone3, orientations_siltstone1, orientations_siltstone2])
orientations['formation'] = ['Tonstein', 'Tonstein', 'Tonstein', 'Tonstein', 'Tonstein',
↪'Tonstein', 'Tonstein', 'Tonstein', 'Tonstein', 'Tonstein','Tonstein', 'Tonstein',
↪'Tonstein', 'Tonstein', 'Tonstein', 'Schluffstein', 'Schluffstein', 'Schluffstein',
↪'Schluffstein', 'Schluffstein']
orientations = orientations[orientations['formation'].isin(['Tonstein', 'Schluffstein
↪'])].reset_index()
orientations
```

[22]:

	index	formation	dip	azimuth	polarity	geometry	\
0	0	Tonstein	89.00	90.00	1.00	POINT (194.155 534.236)	
1	1	Tonstein	45.00	270.00	1.00	POINT (489.592 406.562)	
2	2	Tonstein	50.00	90.00	1.00	POINT (231.629 212.654)	
3	3	Tonstein	89.00	90.00	1.00	POINT (199.384 48.813)	
4	4	Tonstein	45.00	270.00	1.00	POINT (433.381 87.158)	
5	0	Tonstein	0.00	0.00	1.00	POINT (314.421 497.198)	
6	1	Tonstein	26.23	269.64	1.00	POINT (423.250 524.977)	
7	2	Tonstein	24.15	269.44	1.00	POINT (469.439 554.117)	
8	3	Tonstein	21.76	269.66	1.00	POINT (519.768 607.278)	
9	0	Tonstein	0.00	0.00	1.00	POINT (314.857 217.992)	
10	1	Tonstein	25.65	271.63	1.00	POINT (421.289 219.299)	
11	2	Tonstein	24.92	270.84	1.00	POINT (468.023 216.902)	
12	3	Tonstein	21.95	270.25	1.00	POINT (519.550 255.793)	
13	0	Tonstein	0.00	0.00	1.00	POINT (315.075 72.016)	
14	1	Tonstein	28.45	269.01	1.00	POINT (418.565 46.307)	
15	0	Schluffstein	0.00	0.00	1.00	POINT (321.720 24.846)	
16	1	Schluffstein	61.03	88.69	1.00	POINT (262.894 22.123)	
17	0	Schluffstein	0.00	0.00	1.00	POINT (319.650 538.049)	
18	1	Schluffstein	7.69	270.18	1.00	POINT (344.924 553.082)	
19	2	Schluffstein	26.43	269.99	1.00	POINT (445.473 590.012)	

	X	Y	Z
0	194.15	534.24	203.22
1	489.59	406.56	140.53
2	231.63	212.65	80.31
3	199.38	48.81	162.43
4	433.38	87.16	116.59
5	314.42	497.20	106.25
6	423.25	524.98	116.88
7	469.44	554.12	138.12
8	519.77	607.28	159.38
9	314.86	217.99	106.25
10	421.29	219.30	116.88
11	468.02	216.90	138.12
12	519.55	255.79	159.38
13	315.08	72.02	106.25
14	418.57	46.31	116.88
15	321.72	24.85	106.25
16	262.89	22.12	116.88

(continues on next page)

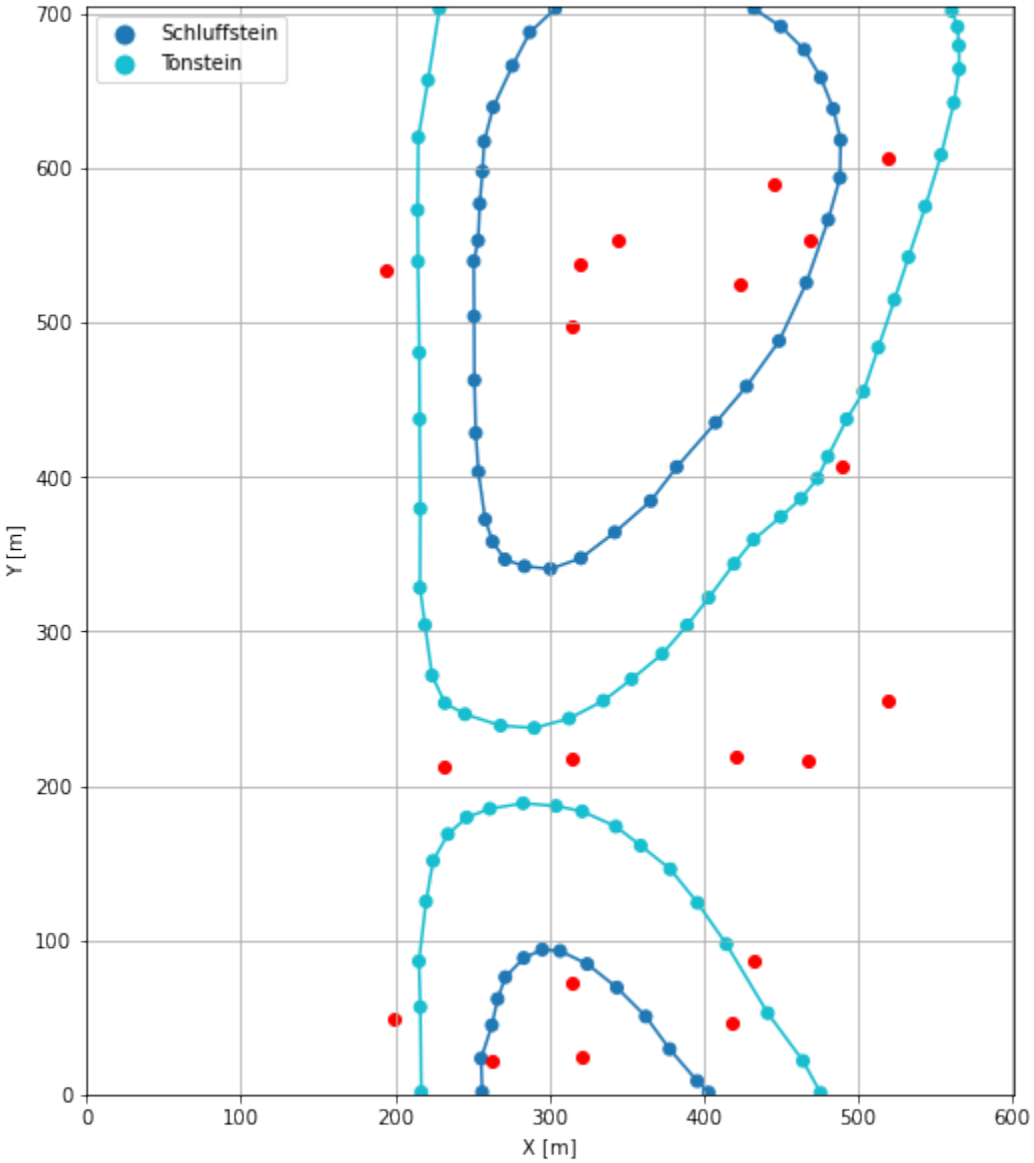
(continued from previous page)

```
17 319.65 538.05 148.75
18 344.92 553.08 159.38
19 445.47 590.01 180.62
```

Plotting the Orientations

[23]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[23]:
```
(0.0, 705.0)
```



7.32.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[24]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[25]:
```
geo_model = gp.create_model('Model32')
geo_model
```

[25]:
```
Model32  2022-04-17 15:06
```

Initiate Data

[26]:
```
gp.init_data(geo_model, [0, 601, 0, 705, 0, 300], [100, 100, 100],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[26]:
```
Model32  2022-04-17 15:06
```

Model Surfaces

[27]:
```
geo_model.surfaces
```

[27]:
```
        surface          series  order_surfaces    color  id
0  Schluffstein  Default series               1  #015482   1
1      Tonstein  Default series               2  #9f0052   2
```

Mapping the Stack to Surfaces

[28]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata1': ('Schluffstein', 'Tonstein'),
                          },
                         remove_unused_series=True)
geo_model.add_surfaces('Sandstein')
```

[28]:
```
        surface   series  order_surfaces    color  id
0  Schluffstein  Strata1               1  #015482   1
1      Tonstein  Strata1               2  #9f0052   2
2     Sandstein  Strata1               3  #ffbe00   3
```

Showing the Number of Data Points

[29]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[29]:
```
        surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0  Schluffstein  Strata1               1  #015482   1                 48               ␣
↪    5
1      Tonstein  Strata1               2  #9f0052   2                 58               ␣
↪   15
2     Sandstein  Strata1               3  #ffbe00   3                  0               ␣
↪    0
```

Loading Digital Elevation Model

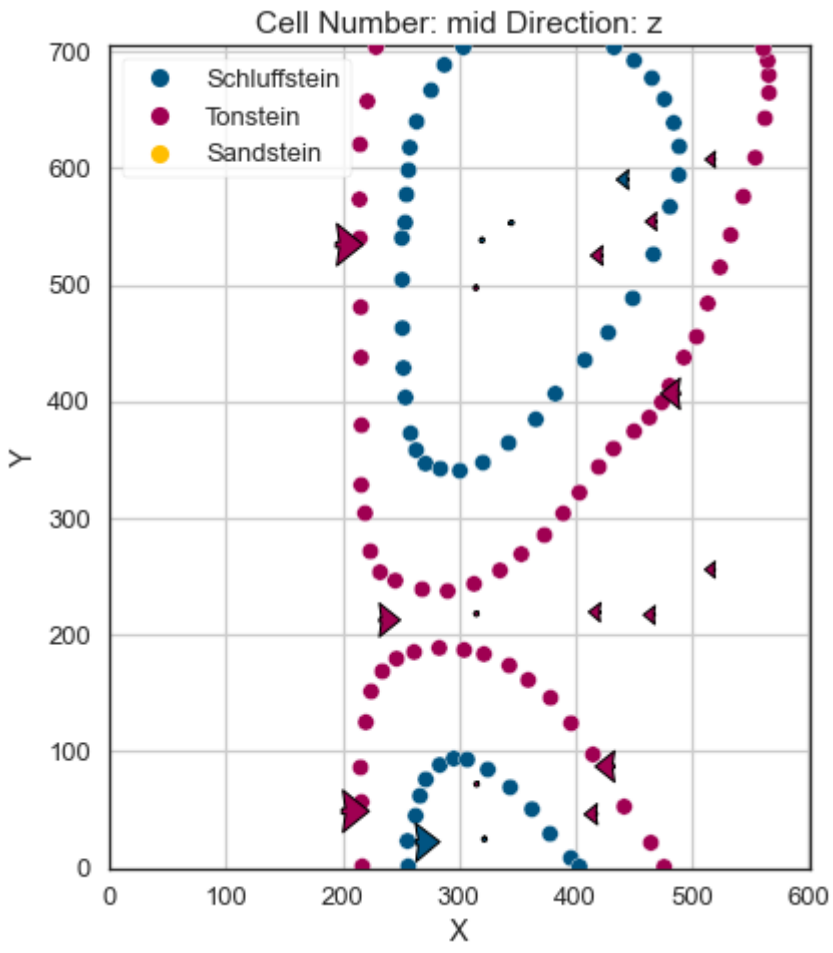
[30]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster32.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

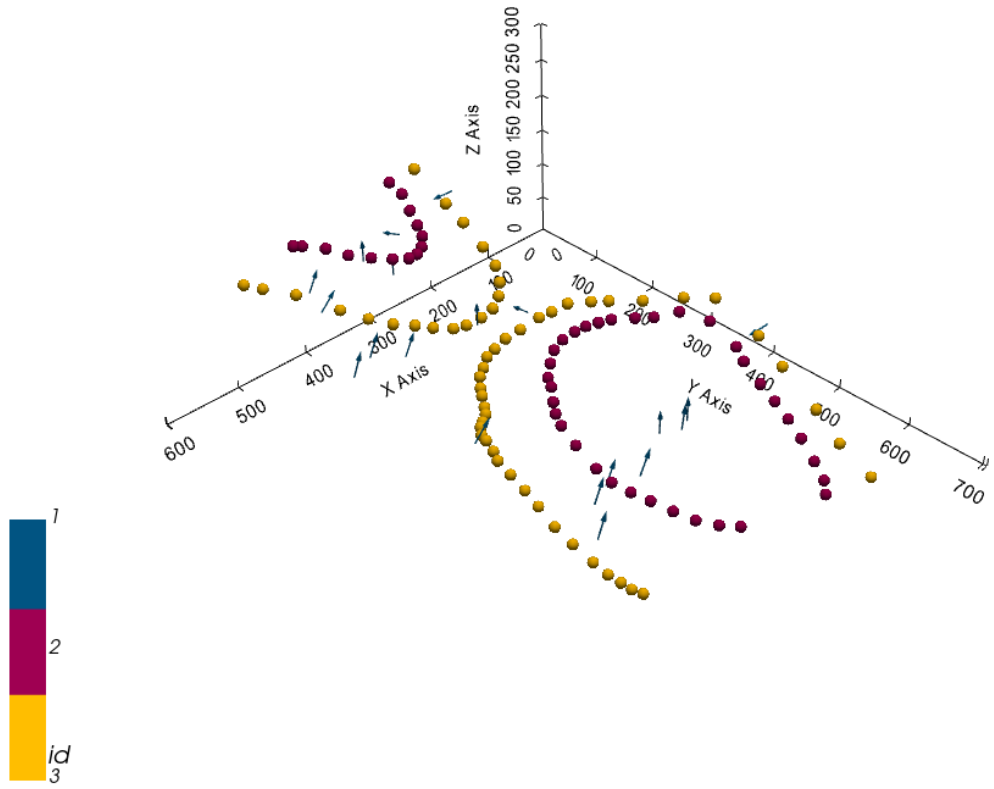
[30]:
```
Grid Object. Values:
array([[  3.005     ,   3.525     ,   1.5       ],
       [  3.005     ,   3.525     ,   4.5       ],
       [  3.005     ,   3.525     ,   7.5       ],
       ...,
       [598.49583333, 692.5       , 175.96000671],
       [598.49583333, 697.5       , 176.22775269],
       [598.49583333, 702.5       , 176.43305969]])
```

Plotting Input Data

[31]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[32]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[32]:
```
<gempy.plot.vista.GemPyToVista at 0x23712e34eb0>
```

Setting the Interpolator

[33]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range             973.77
```

(continues on next page)

(continued from previous page)

```
$C_o$           22576.81
drift equations      [3]
```

[33]:
```
<gempy.core.interpolator.InterpolatorModel at 0x2370bce2610>
```

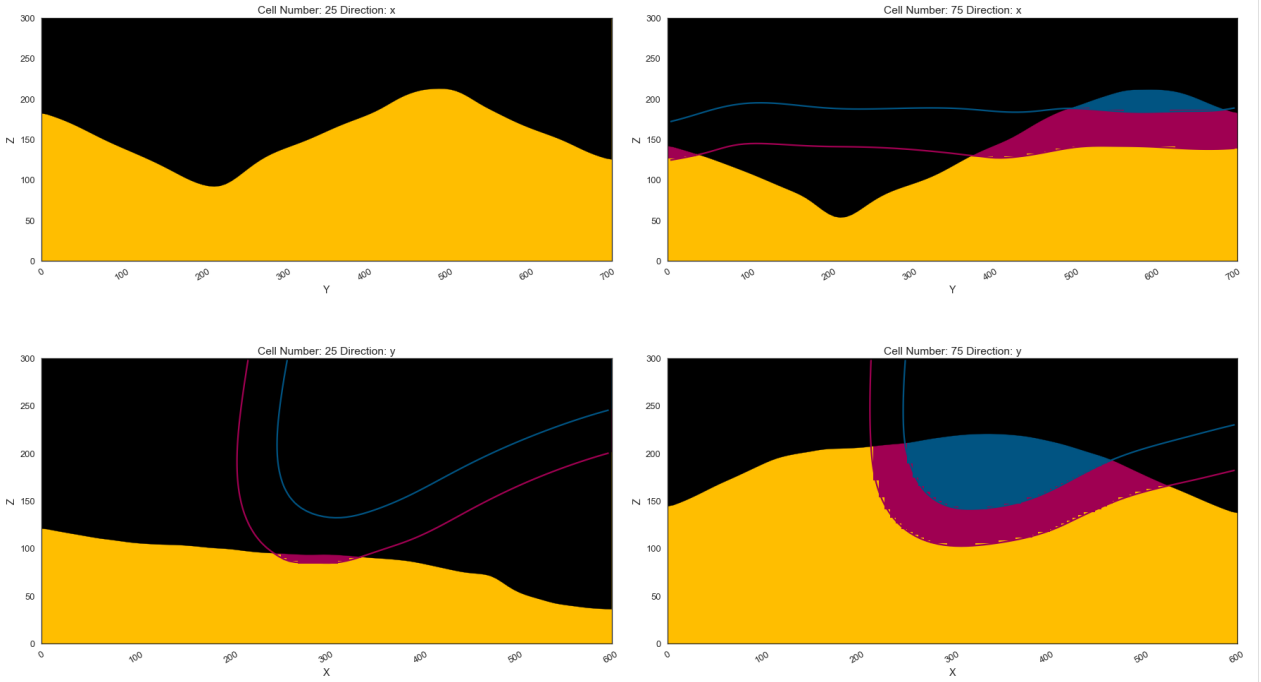
Computing Model

[34]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

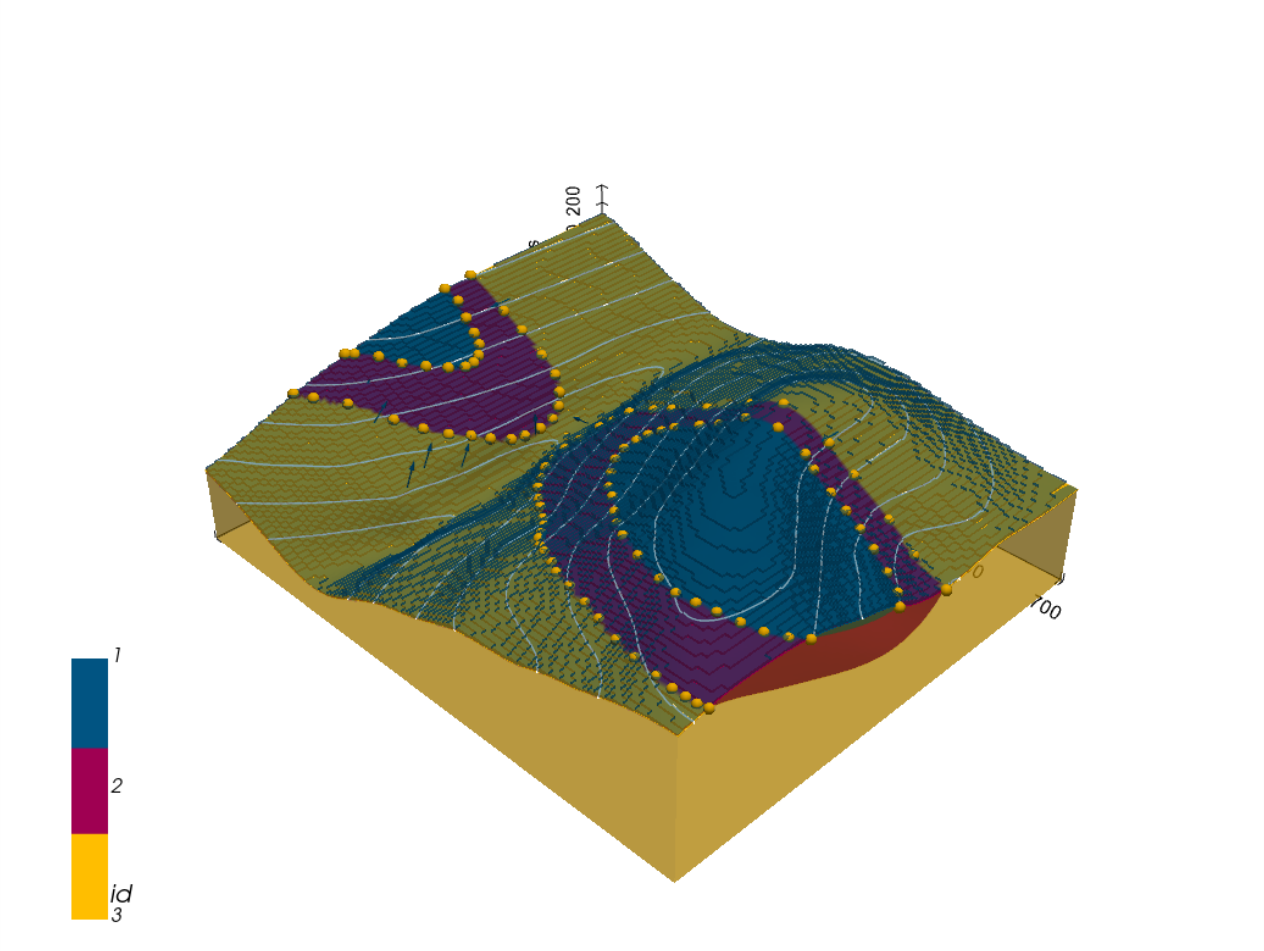
Plotting Cross Sections

[35]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 75, 25, 75], show_
↪topography=True, show_data=False)
```

[35]:
```
<gempy.plot.visualization_2d.Plot2D at 0x23713b99f40>
```



[36]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=True)
```



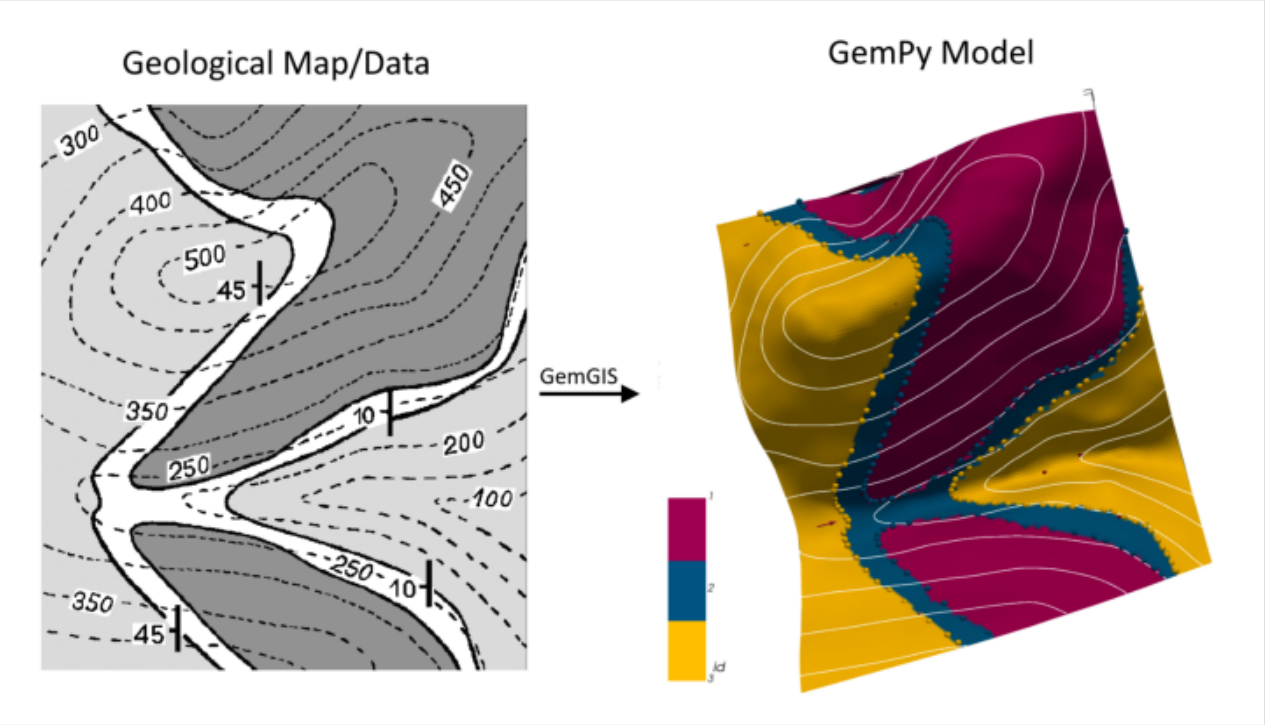
[ ]:

7.33 Example 33 - Folded Layers

This example will show how to convert the geological map below using `GemGIS` to a `GemPy` model. This example is based on digitized data. The area is 601 m wide (W-E extent) and 705 m high (N-S extent). The vertical model extents varies between 0 m and 300 m. The model represents two folded stratigraphic units (blue and red) above an unspecified basement (yellow). The map has been georeferenced with QGIS. The stratigraphic boundaries were digitized in QGIS. Strikes lines were digitized in QGIS as well and were used to calculate orientations for the `GemPy` model. These will be loaded into the model directly. The contour lines were also digitized and will be interpolated with `GemGIS` to create a topography for the model.

Map Source: Unknown

[1]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('../images/cover_example33.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



7.33.1 Licensing

7.33.2 Import GemGIS

If you have installed `GemGIS` via pip or conda, you can import `GemGIS` like any other package. If you have downloaded the repository, append the path to the directory where the `GemGIS` repository is stored and then import `GemGIS`.

[2]:
```
import warnings
warnings.filterwarnings("ignore")
import gemgis as gg
```

7.33.3 Importing Libraries and loading Data

All remaining packages can be loaded in order to prepare the data and to construct the model. The example data is downloaded from an external server using `pooch`. It will be stored in a data folder in the same directory where this notebook is stored.

[3]:
```
import geopandas as gpd
import rasterio
```

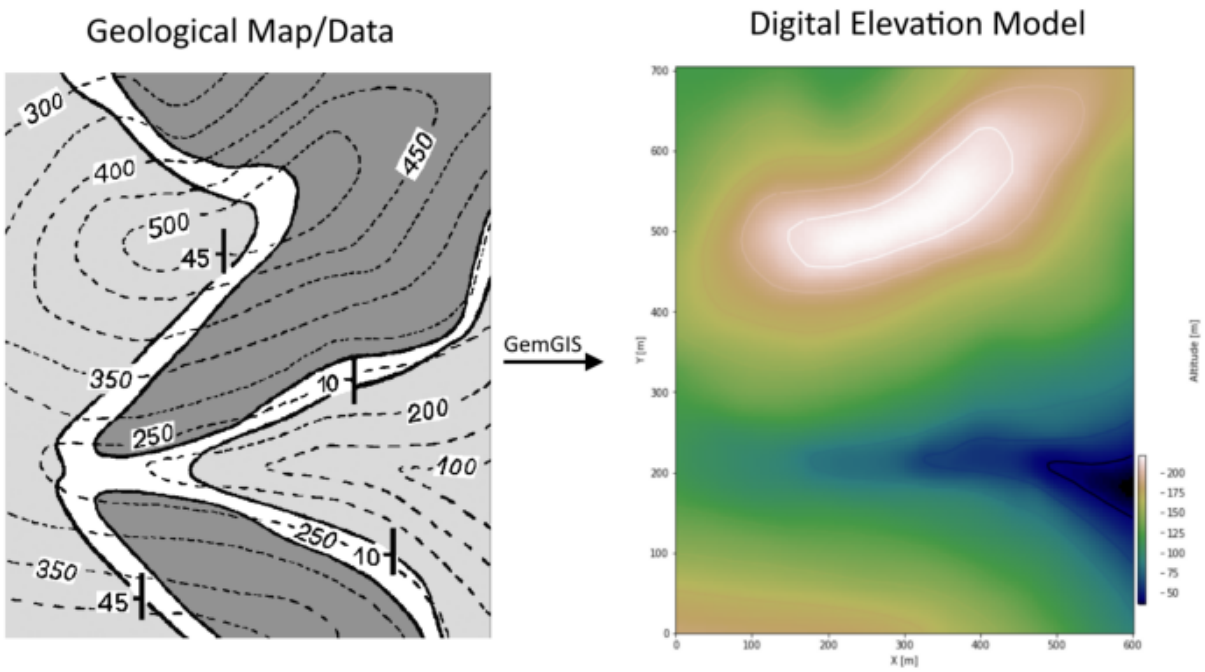
[4]:
```
file_path = 'data/example33/'
gg.download_gemgis_data.download_tutorial_data(filename="example33_folded_layers.zip",␣
↪dirpath=file_path)
```

```
Downloading file 'example33_folded_layers.zip' from 'https://rwth-aachen.sciebo.de/s/
↪AfXRsZywYDbUF34/download?path=%2Fexample33_folded_layers.zip' to 'C:\Users\ale93371\
↪Documents\gemgis\docs\getting_started\example\data\example33'.
```

7.33.4 Creating Digital Elevation Model from Contour Lines

The digital elevation model (DEM) will be created by interpolating contour lines digitized from the georeferenced map using the `SciPy` Radial Basis Function interpolation wrapped in `GemGIS`. The respective function used for that is `gg.vector.interpolate_raster()`.

[5]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img = mpimg.imread('../images/dem_example33.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[6]:
```
topo = gpd.read_file(file_path + 'topo33.shp')
# topo['Z'] = topo['Z']*0.425
topo.head()
```

[6]:

	id	Z	geometry
0	None	400	LINESTRING (0.994 52.851, 36.954 48.266, 68.32...

(continues on next page)

(continued from previous page)

```
1  None  350  LINESTRING (1.235 98.706, 42.022 94.603, 80.39...
2  None  300  LINESTRING (1.477 157.955, 44.435 147.818, 81...
3  None  300  LINESTRING (1.597 320.738, 11.492 310.722, 22...
4  None  250  LINESTRING (553.299 3.497, 534.233 30.527, 513...
```

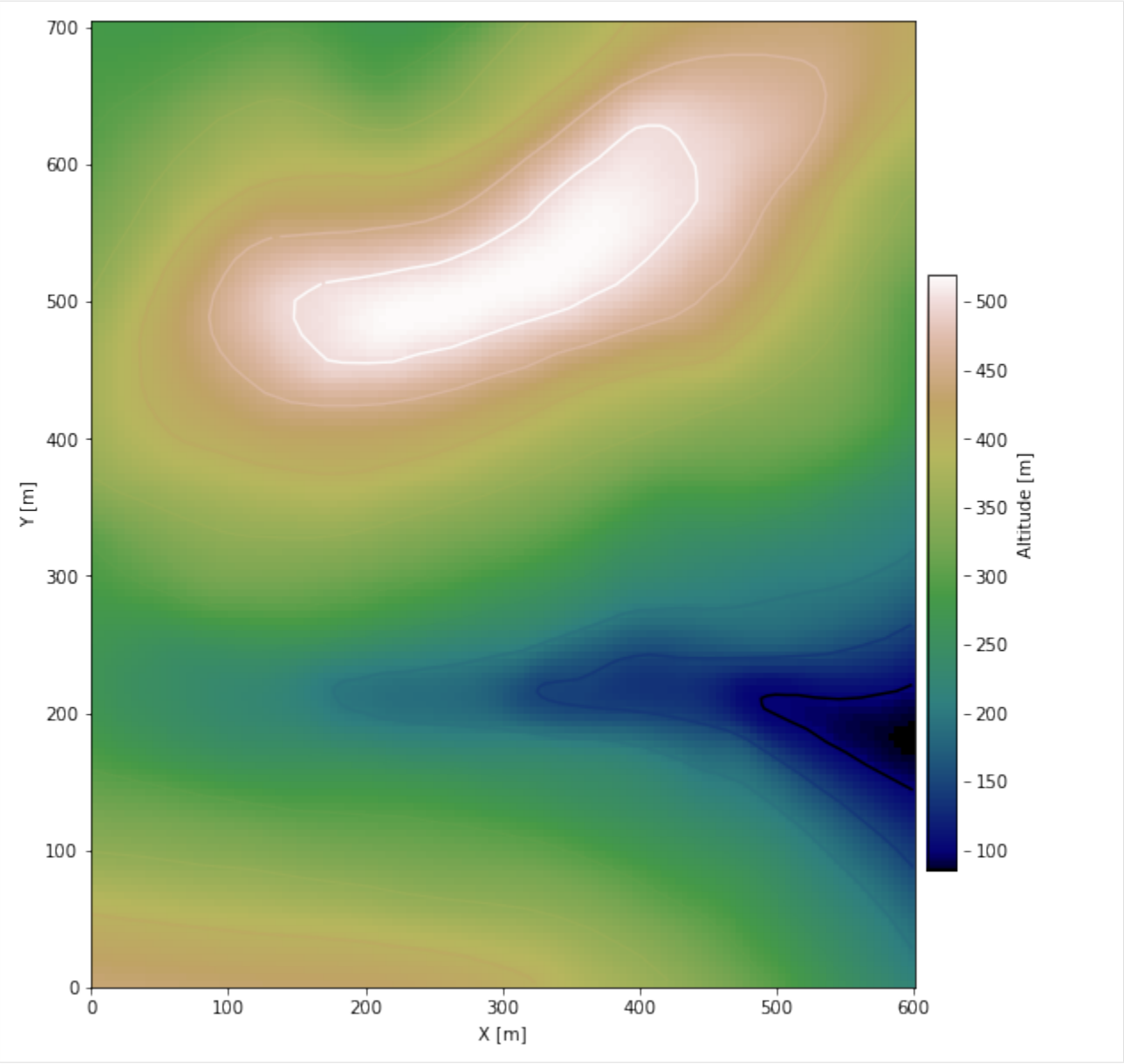
Interpolating the contour lines

[7]:
```
topo_raster = gg.vector.interpolate_raster(gdf=topo, value='Z', method='rbf', res=5)
```

Plotting the raster

[8]:
```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg

from mpl_toolkits.axes_grid1 import make_axes_locatable

fig, ax = plt.subplots(1, figsize=(10, 10))
topo.plot(ax=ax, aspect='equal', column='Z', cmap='gist_earth')
im = ax.imshow(topo_raster, origin='lower', extent=[0, 601, 0, 705], cmap='gist_earth')
divider = make_axes_locatable(ax)
cax = divider.append_axes("right", size="5%", pad=0.05)
cbar = plt.colorbar(im, cax=cax)
cbar.set_label('Altitude [m]')
ax.set_xlabel('X [m]')
ax.set_ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[8]:
```
(0.0, 705.0)
```



Saving the raster to disc

After the interpolation of the contour lines, the raster is saved to disc using `gg.raster.save_as_tiff()`. The function will not be executed as a raster is already provided with the example data.

[9]:
```
gg.raster.save_as_tiff(raster=topo_raster, path=file_path + 'raster33.tif', extent=[0,␣
↪601, 0, 705], crs='EPSG:4326', overwrite_file=True)
```

```
Raster successfully saved
```

Opening Raster

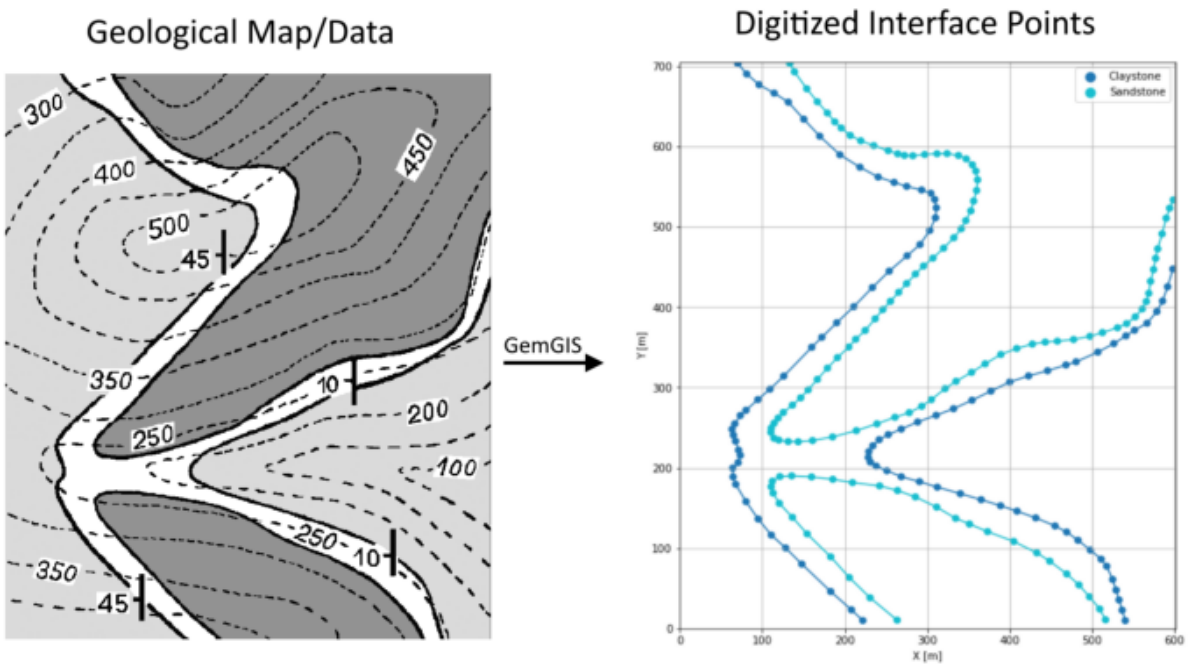
The previously computed and saved raster can now be opened using rasterio.

[10]:
```
topo_raster = rasterio.open(file_path + 'raster33.tif')
```

7.33.5 Interface Points of stratigraphic boundaries

The interface points will be extracted from LineStrings digitized from the georeferenced map using QGIS. It is important to provide a formation name for each layer boundary. The vertical position of the interface point will be extracted from the digital elevation model using the `GemGIS` function `gg.vector.extract_xyz()`. The resulting GeoDataFrame now contains single points including the information about the respective formation.

[11]:
```
img = mpimg.imread('../images/interfaces_example33.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



[12]:
```
interfaces = gpd.read_file(file_path + 'interfaces33.shp')
interfaces.head()
```

[12]:

	id	formation	geometry
0	None	Claystone	LINESTRING (70.299 703.464, 81.050 690.563, 96...
1	None	Sandstone	LINESTRING (263.603 10.236, 230.705 38.189, 20...
2	None	Claystone	LINESTRING (598.176 447.589, 591.725 425.549, ...
3	None	Sandstone	LINESTRING (133.300 703.787, 138.998 693.251, ...

Extracting Z coordinate from Digital Elevation Model

[13]:
```
interfaces_coords = gg.vector.extract_xyz(gdf=interfaces, dem=topo_raster)
interfaces_coords = interfaces_coords[interfaces_coords['formation'].isin(['Claystone',
↪'Sandstone'])]#
interfaces_coords
```

[13]:

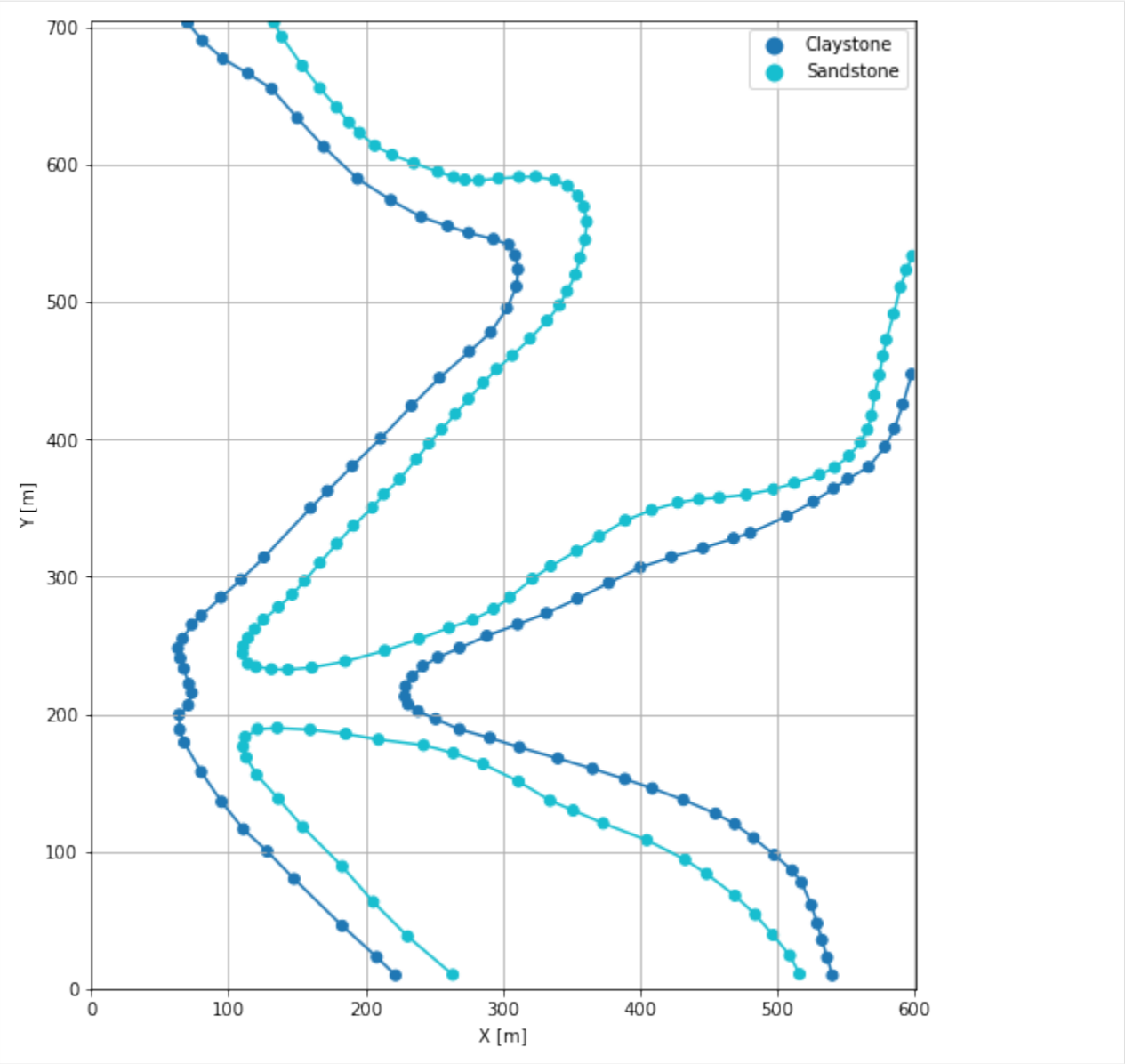
	formation	geometry	X	Y	Z
0	Claystone	POINT (70.299 703.464)	70.30	703.46	285.25
1	Claystone	POINT (81.050 690.563)	81.05	690.56	291.10
2	Claystone	POINT (96.102 677.232)	96.10	677.23	306.26
3	Claystone	POINT (114.378 666.696)	114.38	666.70	322.26
4	Claystone	POINT (131.580 655.514)	131.58	655.51	338.53
..	...	...	...	...	...
210	Sandstone	POINT (579.684 472.747)	579.68	472.75	313.90
211	Sandstone	POINT (585.059 491.453)	585.06	491.45	315.09
212	Sandstone	POINT (590.005 510.913)	590.00	510.91	320.30
213	Sandstone	POINT (594.198 523.491)	594.20	523.49	322.96
214	Sandstone	POINT (598.713 533.705)	598.71	533.70	326.64

[215 rows x 5 columns]

Plotting the Interface Points

[14]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[14]:
```
(0.0, 705.0)
```

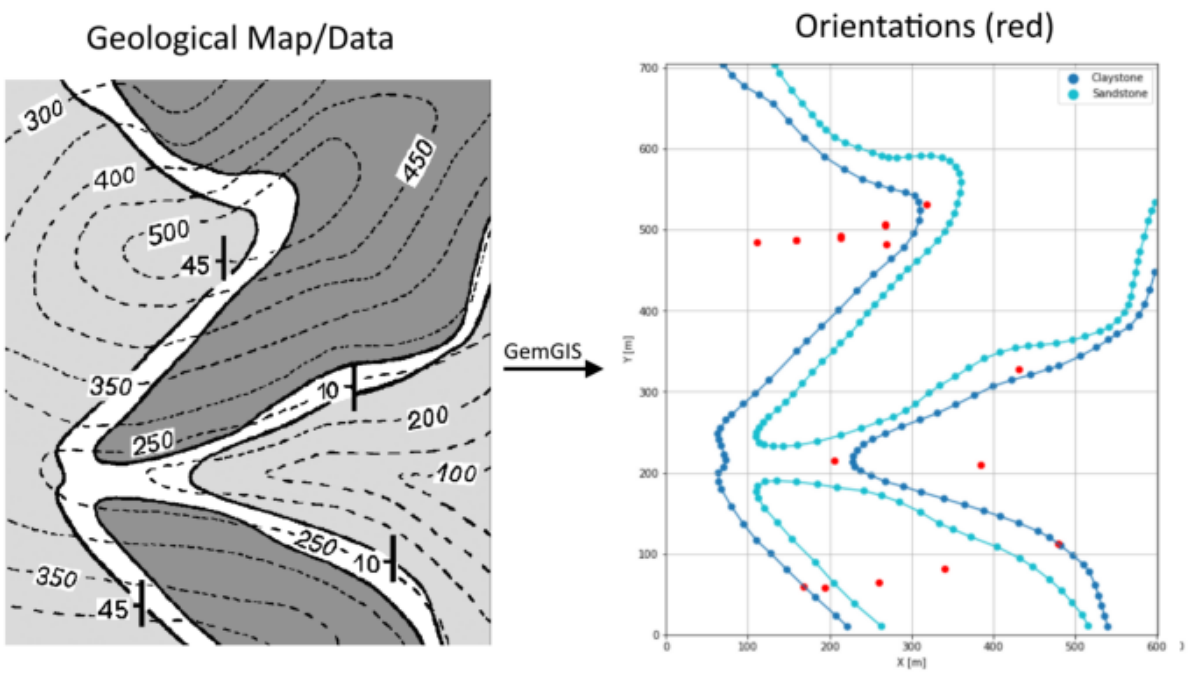


7.33.6 Orientations from Strike Lines and Map

Strike lines connect outcropping stratigraphic boundaries (interfaces) of the same altitude. In other words: the intersections between topographic contours and stratigraphic boundaries at the surface. The height difference and the horizontal difference between two digitized lines is used to calculate the dip and azimuth and hence an orientation that is necessary for `GemPy`. In order to calculate the orientations, each set of strikes lines/LineStrings for one formation must be given an id number next to the altitude of the strike line. The id field is already predefined in QGIS. The strike line with the lowest altitude gets the id number 1, the strike line with the highest altitude the the number according to the number of digitized strike lines. It is currently recommended to use one set of strike lines for each structural element of one formation as illustrated.

In addition, orientations were provided on the map which were digitized as points and can be used right away.

[15]:
```
img = mpimg.imread('../images/orientations_example33.png')
plt.figure(figsize=(10, 10))
imgplot = plt.imshow(img)
plt.axis('off')
plt.tight_layout()
```



Orientations from Map

[16]:
```
orientations_points = gpd.read_file(file_path + 'orientations33.shp')
orientations_points = gg.vector.extract_xyz(gdf=orientations_points, dem=topo_raster)
orientations_points
```

[16]:

	formation	dip	azimuth	polarity	geometry	X	\
0	Claystone	225.00	270.00	1.00	POINT (168.241 59.261)	168.24	
1	Claystone	10.00	270.00	1.00	POINT (479.807 112.801)	479.81	
2	Claystone	10.00	270.00	1.00	POINT (431.642 328.037)	431.64	
3	Claystone	225.00	270.00	1.00	POINT (269.516 481.992)	269.52	
4	Claystone	89.00	90.00	1.00	POINT (41.918 218.780)	41.92	
5	Claystone	89.00	90.00	1.00	POINT (38.416 483.518)	38.42	
6	Claystone	89.00	90.00	1.00	POINT (52.067 77.344)	52.07	
7	Claystone	89.00	90.00	1.00	POINT (39.027 658.036)	39.03	
8	Claystone	1.00	270.00	1.00	POINT (113.787 658.036)	113.79	
9	Claystone	1.00	270.00	1.00	POINT (116.395 479.830)	116.39	
10	Claystone	1.00	270.00	1.00	POINT (119.872 216.432)	119.87	
11	Claystone	1.00	270.00	1.00	POINT (134.650 79.083)	134.65	
12	Claystone	10.00	270.00	1.00	POINT (432.360 660.347)	432.36	
13	Claystone	10.00	270.00	1.00	POINT (430.959 478.252)	430.96	
14	Claystone	10.00	270.00	1.00	POINT (432.360 413.818)	432.36	
15	Claystone	10.00	270.00	1.00	POINT (433.761 560.895)	433.76	

(continues on next page)

(continued from previous page)

```
16  Claystone  10.00   270.00      1.00  POINT (446.367 221.918) 446.37
17  Claystone  10.00   270.00      1.00   POINT (437.963 21.614) 437.96

        Y      Z
0   59.26 373.80
1  112.80 232.69
2  328.04 247.93
3  481.99 514.97
4  218.78 249.89
5  483.52 398.11
6   77.34 365.65
7  658.04 297.54
8  658.04 333.16
9  479.83 475.90
10 216.43 229.37
11  79.08 355.83
12 660.35 464.11
13 478.25 434.01
14 413.82 350.30
15 560.90 501.68
16 221.92 130.60
17  21.61 328.75
```

Orientations from Strike Lines

[17]:
```
strikes = gpd.read_file(file_path + 'strikes33.shp')
# strikes['Z'] = strikes['Z']*0.425
strikes
```

[17]:

	id	formation	Z	geometry
0	5	Sandstone	500	LINESTRING (342.516 586.923, 342.408 497.796)
1	4	Sandstone	450	LINESTRING (294.458 589.825, 293.168 448.987)
2	3	Sandstone	400	LINESTRING (244.359 598.426, 242.638 392.651)
3	2	Sandstone	350	LINESTRING (180.927 639.495, 184.583 331.155)
4	1	Sandstone	300	LINESTRING (135.558 699.486, 134.590 279.926)
5	5	Claystone	500	LINESTRING (294.136 545.477, 293.491 480.326)
6	4	Claystone	450	LINESTRING (244.251 560.744, 242.746 433.451)
7	3	Claystone	400	LINESTRING (181.465 601.598, 184.260 374.213)
8	2	Claystone	350	LINESTRING (135.235 650.838, 135.020 323.361)
9	1	Claystone	300	LINESTRING (86.426 684.811, 89.221 279.926)
10	1	Claystone2	200	LINESTRING (245.111 235.632, 245.541 196.928)
11	2	Claystone2	250	LINESTRING (522.488 352.603, 526.359 55.444)
12	1	Sandstone2	250	LINESTRING (110.723 247.458, 111.583 177.361)
13	2	Sandstone2	250	LINESTRING (299.189 279.819, 299.619 155.967)

[18]:
```
strikes[strikes['formation'] == 'Claystone']
```

[18]:

	id	formation	Z	geometry
5	5	Claystone	500	LINESTRING (294.136 545.477, 293.491 480.326)
6	4	Claystone	450	LINESTRING (244.251 560.744, 242.746 433.451)
7	3	Claystone	400	LINESTRING (181.465 601.598, 184.260 374.213)

(continues on next page)

(continued from previous page)

```
8   2  Claystone  350  LINESTRING (135.235 650.838, 135.020 323.361)
9   1  Claystone  300    LINESTRING (86.426 684.811, 89.221 279.926)
```

Calculating Orientations for each formation

[19]:
```
orientations_claystone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Claystone'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone1['dip'] = orientations_claystone1['dip'] + 180
orientations_claystone1
```

[19]:
```
     dip  azimuth      Z                 geometry  polarity  formation      X  \
0 227.33   269.78 325.00  POINT (111.476 484.734)      1.00  Claystone 111.48
1 227.22   269.80 375.00  POINT (158.995 487.502)      1.00  Claystone 159.00
2 220.18   269.62 425.00  POINT (213.181 492.502)      1.00  Claystone 213.18
3 224.96   270.65 475.00  POINT (268.656 505.000)      1.00  Claystone 268.66

       Y
0 484.73
1 487.50
2 492.50
3 505.00
```

[20]:
```
orientations_claystone2 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Claystone2'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_claystone2
```

[20]:
```
    dip  azimuth      Z                 geometry  polarity   formation      X  \
0 10.16   269.26 225.00  POINT (384.875 210.152)      1.00  Claystone2 384.87

       Y
0 210.15
```

[21]:
```
orientations_sandstone1 = gg.vector.calculate_orientations_from_strike_
↪lines(gdf=strikes[strikes['formation'] == 'Sandstone'].sort_values(by='id',␣
↪ascending=True).reset_index())
orientations_sandstone1
```

[21]:
```
    dip  azimuth      Z                 geometry  polarity  formation      X  \
0 47.69   269.85 325.00  POINT (158.915 487.516)      1.00  Sandstone 158.91
1 40.39   269.68 375.00  POINT (213.127 490.432)      1.00  Sandstone 213.13
2 44.97   270.49 425.00  POINT (268.656 507.472)      1.00  Sandstone 268.66
3 46.12   270.39 475.00  POINT (318.138 530.883)      1.00  Sandstone 318.14

       Y
0 487.52
1 490.43
2 507.47
3 530.88
```

$\text{orientations}_{s}andstone2 \quad = \quad gg.vector.calculate_{o}rientations_{f}rom_{s}trike_{l}ines(gdf \quad =$

$strikes[strikes['formation'] == 'Sandstone2'].sort_values(by = 'id', ascending = True).reset_index())orientations_sandstone2$

Merging Orientations

[22]:
```
import pandas as pd
orientations = pd.concat([orientations_points, orientations_claystone1, orientations_
↪claystone2])#, orientations_sandstone1])#, orientations_sandstone2, ])
orientations['formation'] = ['Claystone', 'Claystone', 'Claystone', 'Claystone',
↪'Claystone', 'Claystone', 'Claystone', 'Claystone','Claystone', 'Claystone','Claystone
↪', 'Claystone', 'Claystone', 'Claystone','Claystone', 'Claystone', 'Claystone',
↪'Claystone', 'Claystone', 'Claystone','Claystone','Claystone', 'Claystone']#, 'Claystone
↪', 'Claystone', 'Claystone']#, 'Sandstone', 'Sandstone', 'Sandstone', 'Sandstone']#, 'Sandstone']
orientations = orientations[orientations['formation'].isin(['Claystone', 'Sandstone'])].
↪reset_index()
orientations.at[4, 'Z'] = 300
orientations.at[5, 'Z'] = 300
orientations.at[6, 'Z'] = 300
orientations.at[7, 'Z'] = 300

orientations.at[8, 'Z'] = 100
orientations.at[9, 'Z'] = 100
orientations.at[10, 'Z'] = 100
orientations.at[11, 'Z'] = 100

orientations.at[12, 'Z'] = 250
orientations.at[13, 'Z'] = 250
orientations.at[14, 'Z'] = 250
orientations.at[15, 'Z'] = 250
orientations.at[16, 'Z'] = 250
orientations.at[17, 'Z'] = 250
orientations
```

[22]:

	index	formation	dip	azimuth	polarity	geometry
0	0	Claystone	225.00	270.00	1.00	POINT (168.241 59.261)
1	1	Claystone	10.00	270.00	1.00	POINT (479.807 112.801)
2	2	Claystone	10.00	270.00	1.00	POINT (431.642 328.037)
3	3	Claystone	225.00	270.00	1.00	POINT (269.516 481.992)
4	4	Claystone	89.00	90.00	1.00	POINT (41.918 218.780)
5	5	Claystone	89.00	90.00	1.00	POINT (38.416 483.518)
6	6	Claystone	89.00	90.00	1.00	POINT (52.067 77.344)
7	7	Claystone	89.00	90.00	1.00	POINT (39.027 658.036)
8	8	Claystone	1.00	270.00	1.00	POINT (113.787 658.036)
9	9	Claystone	1.00	270.00	1.00	POINT (116.395 479.830)
10	10	Claystone	1.00	270.00	1.00	POINT (119.872 216.432)
11	11	Claystone	1.00	270.00	1.00	POINT (134.650 79.083)
12	12	Claystone	10.00	270.00	1.00	POINT (432.360 660.347)
13	13	Claystone	10.00	270.00	1.00	POINT (430.959 478.252)
14	14	Claystone	10.00	270.00	1.00	POINT (432.360 413.818)
15	15	Claystone	10.00	270.00	1.00	POINT (433.761 560.895)
16	16	Claystone	10.00	270.00	1.00	POINT (446.367 221.918)
17	17	Claystone	10.00	270.00	1.00	POINT (437.963 21.614)
18	0	Claystone	227.33	269.78	1.00	POINT (111.476 484.734)

(continues on next page)

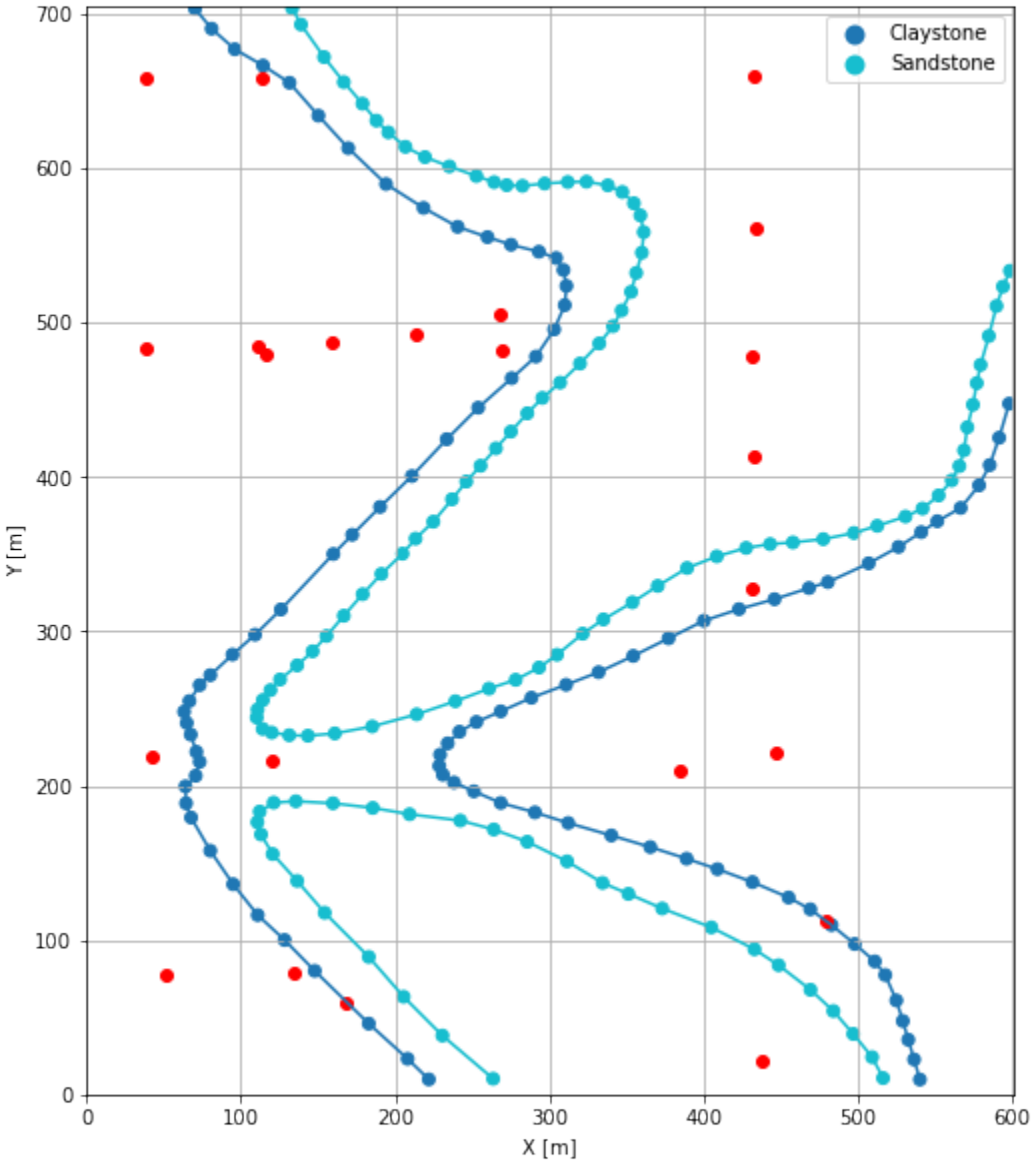
(continued from previous page)

```
19      1  Claystone 227.22   269.80      1.00  POINT (158.995 487.502)
20      2  Claystone 220.18   269.62      1.00  POINT (213.181 492.502)
21      3  Claystone 224.96   270.65      1.00  POINT (268.656 505.000)
22      0  Claystone  10.16   269.26      1.00  POINT (384.875 210.152)

        X      Y      Z
0  168.24  59.26 373.80
1  479.81 112.80 232.69
2  431.64 328.04 247.93
3  269.52 481.99 514.97
4   41.92 218.78 300.00
5   38.42 483.52 300.00
6   52.07  77.34 300.00
7   39.03 658.04 300.00
8  113.79 658.04 100.00
9  116.39 479.83 100.00
10 119.87 216.43 100.00
11 134.65  79.08 100.00
12 432.36 660.35 250.00
13 430.96 478.25 250.00
14 432.36 413.82 250.00
15 433.76 560.90 250.00
16 446.37 221.92 250.00
17 437.96  21.61 250.00
18 111.48 484.73 325.00
19 159.00 487.50 375.00
20 213.18 492.50 425.00
21 268.66 505.00 475.00
22 384.87 210.15 225.00
```

Plotting the Orientations

[23]:
```
fig, ax = plt.subplots(1, figsize=(10, 10))

interfaces.plot(ax=ax, column='formation', legend=True, aspect='equal')
interfaces_coords.plot(ax=ax, column='formation', legend=True, aspect='equal')
orientations.plot(ax=ax, color='red', aspect='equal')
plt.grid()
plt.xlabel('X [m]')
plt.ylabel('Y [m]')
plt.xlim(0, 601)
plt.ylim(0, 705)
```

[23]:
```
(0.0, 705.0)
```



7.33.7 GemPy Model Construction

The structural geological model will be constructed using the GemPy package.

[24]:
```
import gempy as gp
```

```
WARNING (theano.configdefaults): g++ not available, if using conda: `conda install m2w64-
↪toolchain`
WARNING (theano.configdefaults): g++ not detected ! Theano will be unable to execute␣
↪optimized C-implementations (for both CPU and GPU) and will default to Python␣
↪implementations. Performance will be severely degraded. To remove this warning, set␣
↪Theano flags cxx to an empty string.
WARNING (theano.tensor.blas): Using NumPy C-API based implementation for BLAS functions.
```

Creating new Model

[25]:
```
geo_model = gp.create_model('Model33')
geo_model
```

[25]:
```
Model33  2022-04-17 21:57
```

Initiate Data

[26]:
```
gp.init_data(geo_model, [0, 601, 0, 705, 0, 300], [50,50,50],
             surface_points_df=interfaces_coords,
             orientations_df=orientations,
             default_values=True)
```

```
Active grids: ['regular']
```

[26]:
```
Model33  2022-04-17 21:57
```

Model Surfaces

[27]:
```
geo_model.surfaces
```

[27]:
```
     surface          series  order_surfaces    color  id
0  Claystone  Default series               1  #015482   1
1  Sandstone  Default series               2  #9f0052   2
```

Mapping the Stack to Surfaces

[28]:
```
gp.map_stack_to_surfaces(geo_model,
                         {'Strata1': ( 'Sandstone', 'Claystone'), #
                          },
                         remove_unused_series=True)
geo_model.add_surfaces('Sandstein')
```

[28]:
```
     surface   series  order_surfaces    color  id
0  Claystone  Strata1               1  #015482   1
1  Sandstone  Strata1               2  #9f0052   2
2  Sandstein  Strata1               3  #ffbe00   3
```

Showing the Number of Data Points

[29]:
```
gg.utils.show_number_of_data_points(geo_model=geo_model)
```

[29]:
```
     surface   series  order_surfaces    color  id  No. of Interfaces  No. of␣
↪Orientations
0  Claystone  Strata1               1  #015482   1                 96              ␣
↪23
1  Sandstone  Strata1               2  #9f0052   2                119              ␣
↪0
2  Sandstein  Strata1               3  #ffbe00   3                  0              ␣
↪0
```

Loading Digital Elevation Model

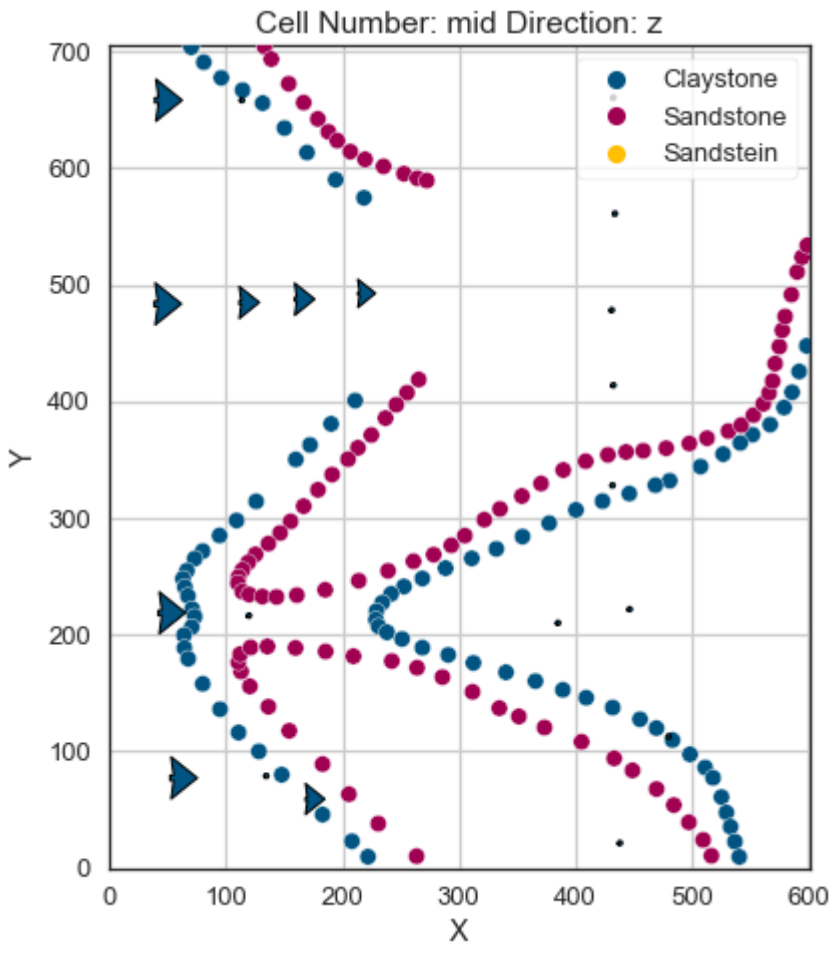
[30]:
```
geo_model.set_topography(source='gdal', filepath=file_path + 'raster33.tif')
```

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
storing converted file...
Active grids: ['regular' 'topography']
```

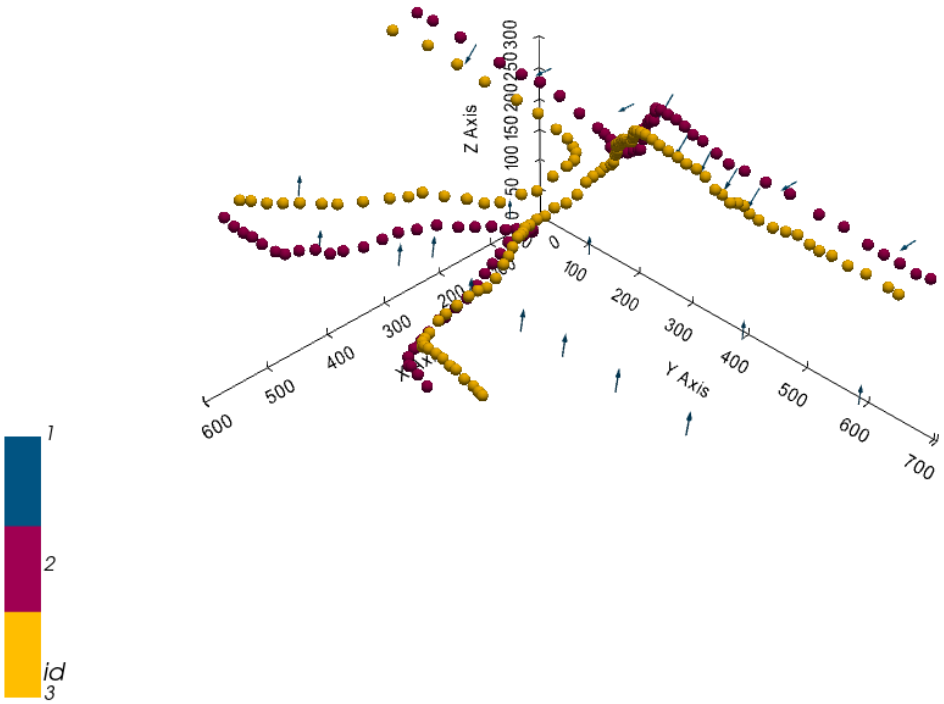
[30]:
```
Grid Object. Values:
array([[  6.01      ,   7.05      ,   3.        ],
       [  6.01      ,   7.05      ,   9.        ],
       [  6.01      ,   7.05      ,  15.        ],
       ...,
       [598.49583333, 692.5       , 414.02352905],
       [598.49583333, 697.5       , 414.65353394],
       [598.49583333, 702.5       , 415.13659668]])
```

Plotting Input Data

[31]:
```
gp.plot_2d(geo_model, direction='z', show_lith=False, show_boundaries=False)
plt.grid()
```



[32]:
```
gp.plot_3d(geo_model, image=False, plotter_type='basic', notebook=True)
```



[32]:
```
<gempy.plot.vista.GemPyToVista at 0x2e34529efd0>
```

Setting the Interpolator

[33]:
```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    verbose=[],
                    update_kriging=False
                    )
```

```
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                  values
range             973.77
```

(continues on next page)

(continued from previous page)

```
$C_o$           22576.81
drift equations      [3]
```

[33]:
```
<gempy.core.interpolator.InterpolatorModel at 0x2e3392ac2b0>
```

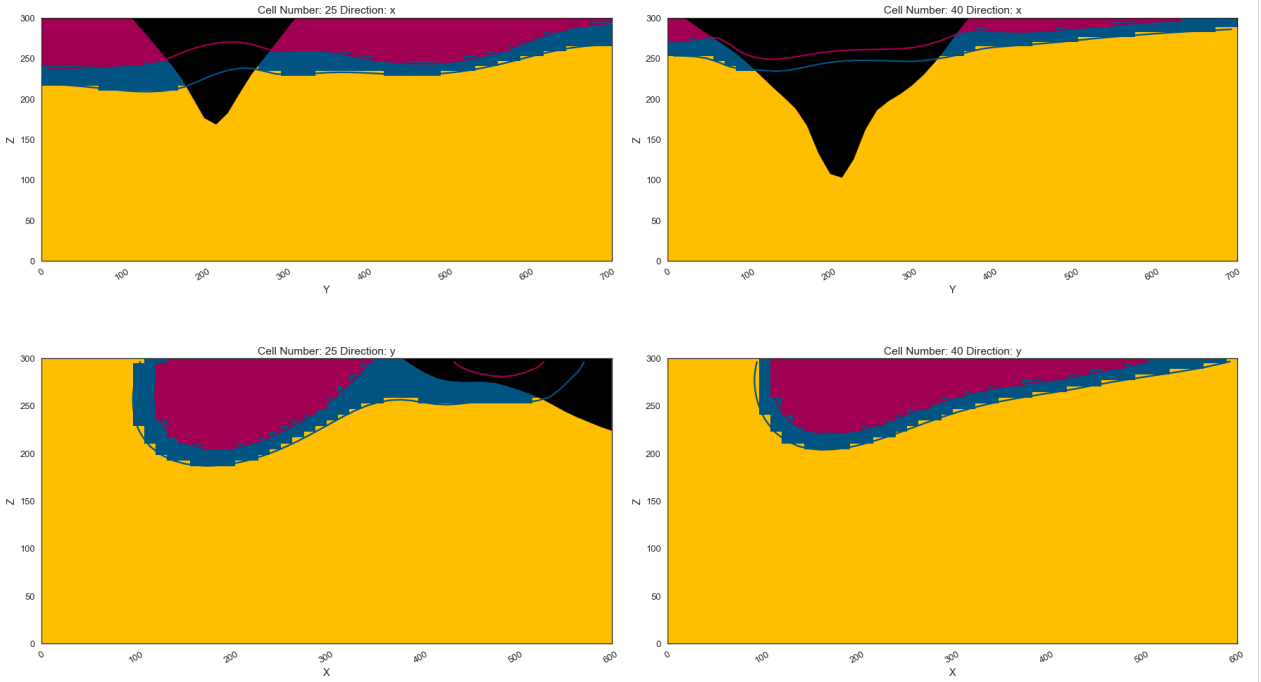
Computing Model

[34]:
```
sol = gp.compute_model(geo_model, compute_mesh=True)
```

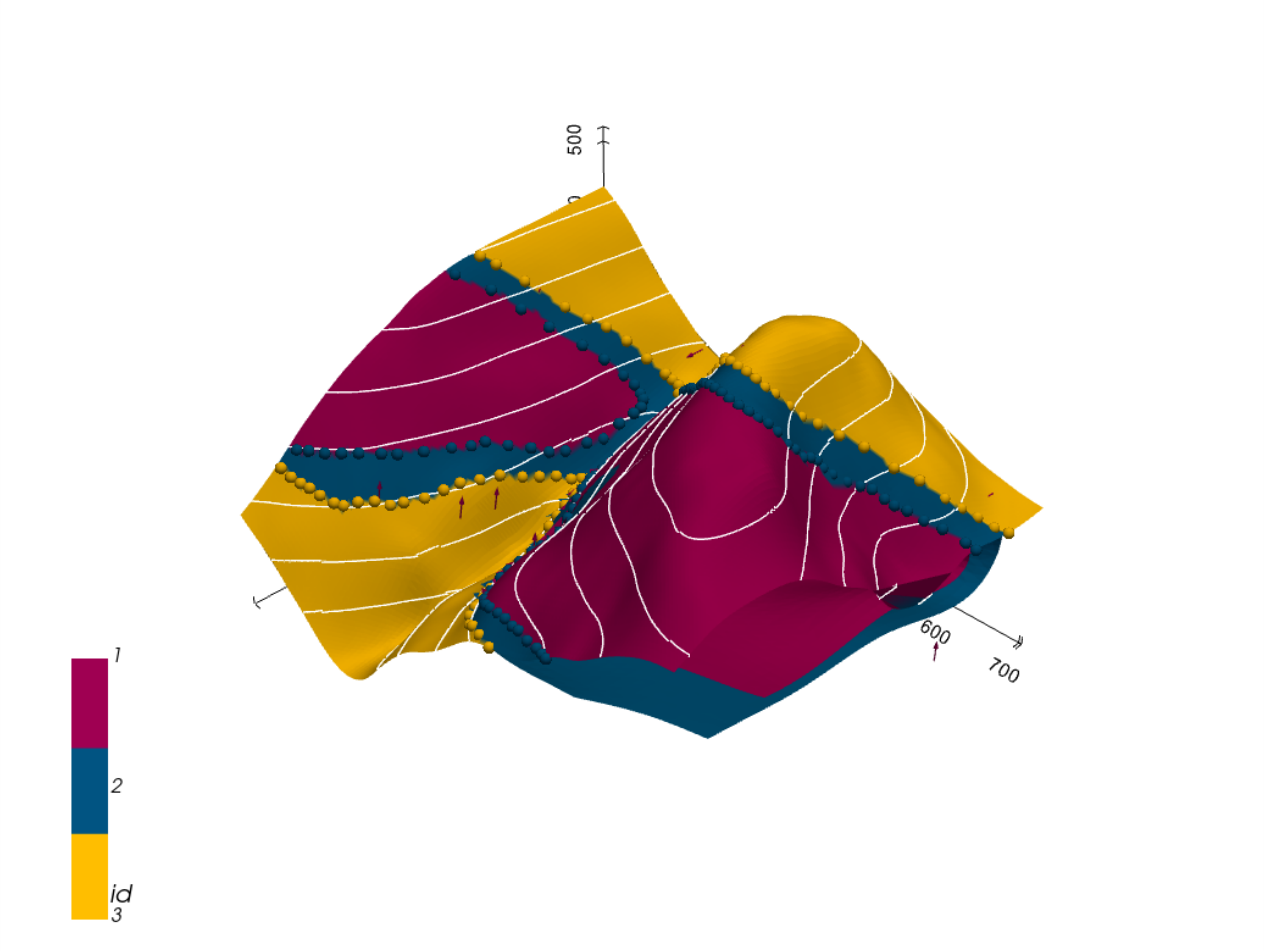
Plotting Cross Sections

[35]:
```
gp.plot_2d(geo_model, direction=['x', 'x', 'y', 'y'], cell_number=[25, 40, 25, 40], show_
↪topography=True, show_data=False)
```

[35]:
```
<gempy.plot.visualization_2d.Plot2D at 0x2e34e9a01f0>
```



[36]:
```
gpv = gp.plot_3d(geo_model, image=False, show_topography=True,
                 plotter_type='basic', notebook=True, show_lith=False)
```


500
600
700
1
2
id
3

[ ]:

CHAPTER

EIGHT

GEMGIS API REFERENCE

The API reference provides an overview of all functions and methods implemented in GemGIS.

8.1 Vector

The following sections provide an overview of the methods implemented in the GemGIS Vector module.

8.1.1 Extracting Coordinates from GeoDataFrames

The following methods are used to extract X and Y coordinates or X, Y, and Z coordinates from GeoDataFrames or Shapely Base Geometries for further usage.

gemgis.vector.extract_xy(gdf[, reset_index, ...])	Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings, Polygons, Geometry Collections) and returning a GeoDataFrame with X and Y coordinates as additional columns
gemgis.vector.extract_xy_linestring(gdf[, ...])	Extracting the coordinates of Shapely LineStrings within a GeoDataFrame and storing the X and Y coordinates in lists per LineString
gemgis.vector.extract_xy_linestrings(gdf[, ...])	Extracting X and Y coordinates from a GeoDataFrame (LineStrings) and returning a GeoDataFrame with X and Y coordinates as additional columns
gemgis.vector.extract_xy_points(gdf[, ...])	Extracting X and Y coordinates from a GeoDataFrame (Points) and returning a GeoDataFrame with X and Y coordinates as additional columns
gemgis.vector.extract_xyz(gdf[, dem, minz, ...])	Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and Z values from a NumPy nd.array or a Rasterio object and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns
gemgis.vector.extract_xyz_array(gdf, dem, extent)	Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and Z values from a NumPy nd.array and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns
gemgis.vector.extract_xyz_rasterio(gdf, dem)	Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and z values from a rasterio object and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns
gemgis.vector.extract_xyz_points(gdf)	Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely Points with Z components
gemgis.vector.extract_xyz_linestrings(gdf[, ...])	Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely LineStrings with Z components
gemgis.vector.extract_xyz_polygons(gdf[, ...])	Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely Polygons with Z components

gemgis.vector.extract_xy

gemgis.vector.**extract_xy**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *overwrite_xy: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings, Polygons, Geometry Collections) and returning a GeoDataFrame with X and Y coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data such as Shapely Points, LineStrings, MultiLineStrings or Polygons or data loaded from disc with

GeoPandas (i.e. Shape File)

- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **overwrite_xy** (*bool*) – Variable to overwrite existing X and Y values. Options include: `True` or `False`, default set to `False`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to re-project coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame with appended x, y columns and Point geometry features

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Changed in version 1.1: If a GeoDataFrame contains LineStrings and MultiLineStrings, the index of the exploded GeoDataFrame will now be reset. Not resetting the index will cause index errors later on.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Extracting X and Y Coordinates from Shapely Base Geometries
>>> gdf_xy = gg.vector.extract_xy(gdf=gdf, reset_index=False)
>>> gdf_xy
    formation   geometry                X       Y
0       Ton             POINT (19.150 293.313)  19.15   293.31
1       Ton             POINT (61.934 381.459)  61.93   381.46
2       Ton             POINT (109.358 480.946) 109.36  480.95
3       Ton             POINT (157.812 615.999) 157.81  616.00
4       Ton             POINT (191.318 719.094) 191.32  719.09
```

See also:

extract_xy_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points

extract_xy_linestring Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings and saving the X and Y coordinates as lists for each LineString

extract_xy_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings

---

Note: GeoDataFrames that contain multiple types of geometries are currently not supported. Please use `gdf = gdf.explode().reset_index(drop=True)` to create a GeoDataFrame with only one type of geometries

---

gemgis.vector.extract_xy_linestring

gemgis.vector.**extract_xy_linestring**(*gdf: geopandas.geodataframe.GeoDataFrame*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting the coordinates of Shapely LineStrings within a GeoDataFrame and storing the X and Y coordinates in lists per LineString

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type LineString
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*Optional[Sequence[float]]*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`

Returns **gdf** – GeoDataFrame containing the additional X and Y columns with lists of X and Y coordinates

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Sand1       LINESTRING (0.256 264.862, 10.593 276.734, 17....
1       None    Ton         LINESTRING (0.188 495.787, 8.841 504.142, 41.0...
2       None    Ton         LINESTRING (970.677 833.053, 959.372 800.023, ...
```

```
>>> # Extracting X and Y Coordinates from LineString Objects
>>> gdf_xy = gg.vector.extract_xy_linestring(gdf=gdf)
>>> gdf_xy
    id      formation   geometry                                                X
↪                                                  Y
0       None    Sand1       LINESTRING (0.256 264.862, 10.593 276.734, 17....   [0.
↪256327195431048, 10.59346813871597, 17.1349...       [264.86214748436396, 276.
↪73370778641777, 289.0...
1       None    Ton         LINESTRING (0.188 495.787, 8.841 504.142, 41.0...   [0.
↪1881868620686138, 8.840672956663411, 41.092...       [495.787213546976, 504.
↪1418419288791, 546.4230...
2       None    Ton         LINESTRING (970.677 833.053, 959.372 800.023, ...
↪[970.6766251230017, 959.3724321757514, 941.291...       [833.052616499831, 800.
↪0232029873156, 754.8012...
```

See also:

extract_xy_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings

extract_xy_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points

extract_xy Extracting X and Y coordinates from Vector Data

gemgis.vector.extract_xy_linestrings

gemgis.vector.**extract_xy_linestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_id: bool = True*, *drop_index: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *overwrite_xy: bool = False*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (LineStrings) and returning a GeoDataFrame with X and Y coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type LineString
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`

- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **overwrite_xy** (*bool*) – Variable to overwrite existing X and Y values. Options include: `True` or `False`, default set to `False`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*Optional[Sequence[float]]*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`

Returns **gdf** – GeoDataFrame with appended X and Y coordinates as additional columns and optional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Sand1       LINESTRING (0.256 264.862, 10.593 276.734, 17....
1       None    Ton         LINESTRING (0.188 495.787, 8.841 504.142, 41.0...
2       None    Ton         LINESTRING (970.677 833.053, 959.372 800.023, ...
```

```
>>> # Extracting X and Y Coordinates from LineString Objects
>>> gdf_xy = gg.vector.extract_xy_linestrings(gdf=gdf, reset_index=False)
>>> gdf_xy
    formation   geometry                X       Y
0       Sand1           POINT (0.256 264.862)   0.26    264.86
1       Sand1           POINT (10.593 276.734)  10.59   276.73
2       Sand1           POINT (17.135 289.090)  17.13   289.09
3       Sand1           POINT (19.150 293.313)  19.15   293.31
4       Sand1           POINT (27.795 310.572)  27.80   310.57
```

See also:

extract_xy_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points

extract_xy_linestring Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings and saving the X and Y coordinates as lists for each LineString

extract_xy Extracting X and Y coordinates from Vector Data

Note: The function was adapted to also extract Z coordinates from LineStrings

---

gemgis.vector.extract_xy_points

gemgis.vector.**extract_xy_points**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_id: bool = True*, *drop_index: bool = True*, *overwrite_xy: bool = False*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points) and returning a GeoDataFrame with X and Y coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type Point
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **overwrite_xy** (*bool*) – Variable to overwrite existing X and Y values. Options include: `True` or `False`, default set to `False`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`

Returns **gdf** – GeoDataFrame with appended X and Y coordinates as new columns and optional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation       geometry
0   None    Ton             POINT (19.150 293.313)
1   None    Ton             POINT (61.934 381.459)
2   None    Ton             POINT (109.358 480.946)
3   None    Ton             POINT (157.812 615.999)
4   None    Ton             POINT (191.318 719.094)
```

```
>>> # Extracting X and Y Coordinates from Point Objects
>>> gdf_xy = gg.vector.extract_xy_points(gdf=gdf, reset_index=False)
>>> gdf_xy
    formation       geometry                X       Y
0   Ton             POINT (19.150 293.313)  19.15   293.31
1   Ton             POINT (61.934 381.459)  61.93   381.46
2   Ton             POINT (109.358 480.946) 109.36  480.95
3   Ton             POINT (157.812 615.999) 157.81  616.00
4   Ton             POINT (191.318 719.094) 191.32  719.09
```

See also:

extract_xy_linestring Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings and saving the X and Y coordinates as lists for each LineString

extract_xy_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings

extract_xy Extracting X and Y coordinates from Vector Data

gemgis.vector.extract_xyz

gemgis.vector.**extract_xyz**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: Union[numpy.ndarray, rasterio.io.DatasetReader] = None*, *minz: float = None*, *maxz: float = None*, *extent: List[Union[int, float]] = None*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and Z values from a NumPy nd.array or a Rasterio object and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing Shapely Points, LineStrings, MultiLineStrings or Polygons

- **dem** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy ndarray or Rasterio object containing the height values, default value is None in case geometries contain Z values
- **minz** (*float*) – Value defining the minimum elevation of the data that needs to be returned, e.g. `minz=50`, default `None`
- **maxz** (*float*) – Value defining the maximum elevation of the data that needs to be returned, e.g. `maxz=500`, default `None`
- **extent** (*List[Union[float,int]]*) – List containing the extent of the np.ndarray, must be provided in the same CRS as the gdf, e.g. `extent=[0, 972, 0, 1069]`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z coordinates as additional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Loading raster file
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Extracting X, Y, and Z Coordinates from Shapely Base Geometries and DEM
>>> gdf_xyz = gg.vector.extract_xyz(gdf=gdf, dem=dem, reset_index=reset_index)
>>> gdf_xyz
    formation   geometry                X       Y       Z
0   Ton         POINT (19.150 293.313)  19.15   293.31  364.99
1       Ton             POINT (61.934 381.459)  61.93   381.46  400.34
2       Ton             POINT (109.358 480.946) 109.36  480.95  459.55
3       Ton             POINT (157.812 615.999) 157.81  616.00  525.69
4       Ton             POINT (191.318 719.094) 191.32  719.09  597.63
```

See also:

extract_xyz_array Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model as array

extract_xyz_rasterio Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation as rasterio object

gemgis.vector.extract_xyz_array

gemgis.vector.**extract_xyz_array**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: numpy.ndarray*, *extent: List[float]*, *minz: float = None*, *maxz: float = None*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and Z values from a NumPy nd.array and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing Shapely Points, LineStrings, MultiLineStrings or Polygons
- **dem** (*np.ndarray*) – NumPy ndarray containing the height values
- **extent** (*list*) – List containing the extent of the np.ndarray, must be provided in the same CRS as the gdf, e.g. `extent=[0, 972, 0, 1069]`
- **minz** (*float*) – Value defining the minimum elevation the data needs to be returned, e.g. `minz=50`, default `None`
- **maxz** (*float*) – Value defining the maximum elevation the data needs to be returned, e.g. `maxz=500`, default `None`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z coordinates

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Loading raster file
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Defining the extent of the array
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Extracting X, Y, and Z Coordinates from Shapely Base Geometries and array
>>> gdf_xyz = gg.vector.extract_xyz_array(gdf=gdf, dem=dem.read(1), extent=extent,␣
↪reset_index=reset_index)
>>> gdf_xyz
    formation   geometry                X       Y       Z
0   Ton         POINT (19.150 293.313)  19.15   293.31  364.99
1       Ton             POINT (61.934 381.459)  61.93   381.46  400.34
2       Ton             POINT (109.358 480.946) 109.36  480.95  459.55
3       Ton             POINT (157.812 615.999) 157.81  616.00  525.69
4       Ton             POINT (191.318 719.094) 191.32  719.09  597.63
```

See also:

extract_xyz_rasterio Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model as rasterio object

extract_xyz Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model

gemgis.vector.extract_xyz_rasterio

gemgis.vector.**extract_xyz_rasterio**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: rasterio.io.DatasetReader*, *minz: float = None*, *maxz: float = None*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and z values from a rasterio object and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing Shapely Points, LineStrings, MultiLineStrings or Polygons
- **dem** (*rasterio.io.DatasetReader*) – Rasterio object containing the height values
- **minz** (*float*) – Value defining the minimum elevation the data needs to be returned, e.g. `minz=50`, default `None`
- **maxz** (*float*) – Value defining the maximum elevation the data needs to be returned, e.g. `maxz=500`, default `None`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame, default `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z coordinates

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Loading raster file
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Extracting X, Y, and Z Coordinates from Shapely Base Geometries and raster
>>> gdf_xyz = gg.vector.extract_xyz_rasterio(gdf=gdf, dem=dem, reset_index=reset_
↪index)
>>> gdf_xyz
    formation   geometry                X       Y       Z
0   Ton         POINT (19.150 293.313)  19.15   293.31  364.99
1       Ton             POINT (61.934 381.459)  61.93   381.46  400.34
2       Ton             POINT (109.358 480.946) 109.36  480.95  459.55
3       Ton             POINT (157.812 615.999) 157.81  616.00  525.69
4       Ton             POINT (191.318 719.094) 191.32  719.09  597.63
```

See also:

extract_xyz_array Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model as array

extract_xyz Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model

gemgis.vector.extract_xyz_points

gemgis.vector.**extract_xyz_points**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely Points with Z components

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Shapely Points with X, Y, and Z components

Returns **gdf** – GeoDataFrame containing Shapely Points with appended X, Y, and Z columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Shapely Point
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> import geopandas as gpd
>>> point = Point(1,2,4)
>>> point.wkt
'POINT Z (0 0 0)'
```

```
>>> # Creating GeoDataFrame from Point
>>> gdf = gpd.GeoDataFrame(geometry=[point, point])
>>> gdf
    geometry
0   POINT Z (0.00000 0.00000 0.00000)
1   POINT Z (0.00000 0.00000 0.00000)
```

```
>>> # Extracting X, Y, and Z Coordinates from Point Objects
>>> gdf = gg.vector.extract_xyz_points(gdf=gdf)
>>> gdf
    geometry                              X       Y       Z
0   POINT Z (1.00000 2.00000 3.00000)   1.00    2.00    3.00
1   POINT Z (1.00000 2.00000 3.00000)   1.00    2.00    3.00
```

See also:

extract_xyz_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings with Z components

extract_xyz_polygons Extracting X and Y coordinates from a GeoDataFrame containing Shapely Polygons with Z component

gemgis.vector.extract_xyz_linestrings

gemgis.vector.**extract_xyz_linestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_index: bool = True*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely LineStrings with Z components

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Shapely LineStrings with X, Y, and Z components
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing Shapely Points with appended X, Y, and Z columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Shapely LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring = LineString(([1,2,3], [4,5,6]))
>>> linestring.wkt
'LINESTRING Z (1 2 3, 4 5 6)'
```

```
>>> # Creating GeoDataFrame from LineString
>>> gdf = gpd.GeoDataFrame(geometry=[linestring, linestring])
>>> gdf
    geometry
0   LINESTRING Z (1.00000 2.00000 3.00000, 4.00000...
1   LINESTRING Z (1.00000 2.00000 3.00000, 4.00000...
```

```
>>> # Extracting X, Y, and Z Coordinates from Point Objects
>>> gdf = gg.vector.extract_xyz_linestrings(gdf=gdf)
>>> gdf
    geometry                points          X       Y       Z
0   POINT (1.00000 2.00000) (1.0, 2.0, 3.0) 1.00    2.00    3.00
1   POINT (4.00000 5.00000) (4.0, 5.0, 6.0) 4.00    5.00    6.00
2   POINT (1.00000 2.00000) (1.0, 2.0, 3.0) 1.00    2.00    3.00
3   POINT (4.00000 5.00000) (4.0, 5.0, 6.0) 4.00    5.00    6.00
```

See also:

extract_xyz_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points with Z components

extract_xyz_polygons Extracting X and Y coordinates from a GeoDataFrame containing Shapely Polygons with Z component

gemgis.vector.extract_xyz_polygons

gemgis.vector.**extract_xyz_polygons**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_index: bool = True*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely Polygons with Z components

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Shapely Polygons with X, Y, and Z components
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing Shapely Points with appended X, Y, and Z columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Shapely Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> import geopandas as gpd
>>> polygon = Polygon([[0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]])
>>> polygon.wkt
'POLYGON Z ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))'
```

```
>>> # Creating GeoDataFrame from LineString
>>> gdf = gpd.GeoDataFrame(geometry=[polygon, polygon])
>>> gdf
    geometry
0       POLYGON Z ((0.00000 0.00000 1.00000, 1.00000 0...
1       POLYGON Z ((0.00000 0.00000 1.00000, 1.00000 0...
```

```
>>> # Extracting X, Y, and Z Coordinates from Point Objects
>>> gdf = gg.vector.extract_xyz_polygons(gdf=gdf)
>>> gdf
    geometry                points          X       Y       Z
0   POINT (0.00000 0.00000) [0.0, 0.0, 1.0] 0.00    0.00    1.00
1   POINT (1.00000 0.00000) [1.0, 0.0, 1.0] 1.00    0.00    1.00
2   POINT (1.00000 1.00000) [1.0, 1.0, 1.0] 1.00    1.00    1.00
3   POINT (0.00000 1.00000) [0.0, 1.0, 1.0] 0.00    1.00    1.00
```

See also:

extract_xyz_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points with Z component

extract_xyz_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings with Z components

8.1.2 Extracting Coordinates and Intersections from GeoDataFrames/Shapely Polygons

The following methods are used to extract coordinates and intersections from GeoDataFrames containing Shapely Polygons or from single Shapely Polygons or multiple Shapely Polygons.

gemgis.vector.extract_xy_from_polygon_intersection(gdf)	Calculating the intersections between Polygons; the table must be sorted by stratigraphic age
gemgis.vector.intersection_polygon_polygon(...)	Calculating the intersection between to Shapely Polygons
gemgis.vector.intersections_polygon_polygons(...)	Calculating the intersections between one polygon and a list of polygons
gemgis.vector.intersections_polygons_polygons(...)	Calculating the intersections between a list of Polygons

gemgis.vector.extract_xy_from_polygon_intersections

gemgis.vector.**extract_xy_from_polygon_intersections**(*gdf: geopandas.geodataframe.GeoDataFrame*, *extract_coordinates: bool = False*, *drop_index: bool = True*) → geopandas.geodataframe.GeoDataFrame

Calculating the intersections between Polygons; the table must be sorted by stratigraphic age

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Polygons of a geological map ordered by their stratigraphic age
- **extract_coordinates** (*bool*) – Variable to extract X and Y coordinates from resulting Shapely Objects. Options include: `True` or `False`, default set to `False`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`

Returns **intersections** – GeoDataFrame containing the intersections of the polygons of a geological map

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> import geopandas as gpd
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Creating GeoDataFrame from polygons and adding formation names
>>> gdf = gpd.GeoDataFrame(geometry=[polygon1, polygon2])
>>> gdf['formation'] = ['Formation1', 'Formation2']
>>> gdf
    geometry                                        formation
0   POLYGON (((0 0, 10 0, 10 10, 0 10, 0 0))    Formation1
1   POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))  Formation2
```

```
>>> # Extracting X an Y coordinates from polygon intersections
>>> intersection = gg.vector.extract_xy_from_polygon_intersections(gdf=gdf)
>>> intersection
    formation   geometry
0   Formation1  LINESTRING (10.0 0.0, 10.0 10.0)
```

See also:

intersection_polygon_polygon Intersecting a polygon with a polygon

intersections_polygon_polygons Intersecting a polygons with multiple polygons

intersections_polygons_polygons Intersecting multiple polygons with multiple polygons

gemgis.vector.intersection_polygon_polygon

gemgis.vector.**intersection_polygon_polygon**(*polygon1: shapely.geometry.polygon.Polygon*, *polygon2: shapely.geometry.polygon.Polygon*) → Union[shapely.geometry.linestring.LineString, shapely.geometry.polygon.Polygon]

Calculating the intersection between to Shapely Polygons

Parameters

- **polygon1** (*shapely.geometry.polygon.Polygon*) – First polygon used for intersecting, e.g. `polygon1=Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`
- **polygon2** (*shapely.geometry.polygon.Polygon*) – Second polygon used for intersecting, e.g. `polygon2=Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`

Returns **intersection** – Intersected geometry as Shapely Object

Return type Union[shapely.geometry.linestring.LineString, shapely.geometry.polygon.Polygon]

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Calculating the intersection between two polygons
>>> intersection = gg.vector.intersection_polygon_polygon(polygon1=polygon1,␣
↪polygon2=polygon2)
>>> intersection.wkt
'LINESTRING (10 0, 10 10)'
```

See also:

intersections_polygon_polygons Intersecting a polygon with mutiple polygons

intersections_polygons_polygons Intersecting multiple polygons with multiple polygons

extract_xy_from_polygon_intersections Extracting intersections between multiple polygons

gemgis.vector.intersections_polygon_polygons

gemgis.vector.**intersections_polygon_polygons**(*polygon1: shapely.geometry.polygon.Polygon*, *polygons2: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) → List[shapely.geometry.base.BaseGeometry]

Calculating the intersections between one polygon and a list of polygons

Parameters

- **polygon1** (*shapely.geometry.polygon.Polygon*) – First polygon used for intersecting, e.g. polygon1=Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
- **polygons2** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) – List of polygons as list or GeoDataFrame to get intersected

Returns **intersections** – List of intersected geometries

Return type List[shapely.geometry.base.BaseGeometry]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Creating list of polygons
>>> polygons2 = [polygon2, polygon2]
```

```
>>> # Calculating the intersections between a polygon with polygons
>>> intersection = gg.vector.intersections_polygon_polygons(polygon1=polygon1,␣
↪polygons2=polygons2)
>>> intersection
[<shapely.geometry.linestring.LineString at 0x231eaf22100>,
<shapely.geometry.linestring.LineString at 0x231eab22970>]
```

```
>>> # Inspecting the first element of the list
>>> intersection[0].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Creating the second element of the list
>>> intersection[1].wkt
'LINESTRING (10 0, 10 10)'
```

See also:

intersection_polygon_polygon Intersecting a polygon with a polygon

intersections_polygons_polygons Intersecting multiple polygons with multiple polygons

extract_xy_from_polygon_intersections Extracting intersections between multiple polygons

gemgis.vector.intersections_polygons_polygons

gemgis.vector.**intersections_polygons_polygons**(*polygons1: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*, *polygons2: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) → List[shapely.geometry.base.BaseGeometry]

Calculating the intersections between a list of Polygons

Parameters

- **polygons1** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) – List of Polygons or GeoDataFrame containing Polygons to be intersected
- **polygons2** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) – List of Polygons or GeoDataFrame containing Polygons to be intersected

Returns **intersections** – List of intersected geometries

Return type List[shapely.geometry.base.BaseGeometry]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating list of polygons
>>> polygons1 = [polygon1, polygon1]
```

```
>>> # Creating second polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Creating list of polygons
>>> polygons2 = [polygon2, polygon2]
```

```
>>> # Calculating intersections between polygons and polygons
>>> intersection = gg.vector.intersections_polygons_polygons(polygons1=polygons1,␣
↪polygons2=polygons2)
>>> intersection
[<shapely.geometry.linestring.LineString at 0x231eaf4dd90>,
<shapely.geometry.linestring.LineString at 0x231ec6e8df0>,
 <shapely.geometry.linestring.LineString at 0x231eaf4dc70>,
 <shapely.geometry.linestring.LineString at 0x231eaf4dd00>]
```

```
>>> # Inspecting the first element of the list
>>> intersection[0].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Inspecting the second element of the list
>>> intersection[1].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Inspecting the third element of the list
>>> intersection[2].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Inspecting the fourth element of the list
>>> intersection[3].wkt
'LINESTRING (10 0, 10 10)'
```

See also:

intersection_polygon_polygon Intersecting a polygon with a polygon

intersections_polygon_polygons Intersecting a polygons with multiple polygons

extract_xy_from_polygon_intersections Extracting intersections between multiple polygons

8.1.3 Calculating and extracting coordinates from cross sections

The following methods are used to calculate and extract coordinates from cross sections where data was stored as Shape Files and is now loaded as GeoDataFrames.

gemgis.vector.extract_interfaces_coordinates_from_cross_section(...)	Extracting coordinates of interfaces digitized on a cross section
gemgis.vector.extract_xyz_from_cross_sections(...)	Extracting X, Y, and Z coordinates from cross sections and digitized interfaces
gemgis.vector.calculate_coordinates_for_point_on_cross_section(...)	Calculating the coordinates for one point digitized on a cross section provided as Shapely LineString
gemgis.vector.calculate_coordinates_for_linestring_on_cross_section(...)	Calculating the coordinates of vertices for a LineString on a straight cross section provided as Shapely LineString
gemgis.vector.calculate_coordinates_for_linestrings_on_cross_section(...)	Calculating the coordinates of vertices for LineStrings on a straight cross section provided as Shapely LineString
gemgis.vector.extract_interfaces_coordinates_from_cross_section(...)	Extracting coordinates of interfaces digitized on a cross section

gemgis.vector.extract_interfaces_coordinates_from_cross_section

gemgis.vector.**extract_interfaces_coordinates_from_cross_section**(*linestring: shapely.geometry.linestring.LineString*, *interfaces_gdf: geopandas.geodataframe.GeoDataFrame*, *extract_coordinates: bool = True*) → geopandas.geodataframe.GeoDataFrame

Extracting coordinates of interfaces digitized on a cross section

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (20, 20)])`
- **interfaces_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the LineStrings of interfaces digitized on a cross section

Returns **gdf** – GeoDataFrame containing the extracted coordinates, depth/elevation data and additional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> import geopandas as gpd
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating GeoDataFrame from LineString
>>> gdf = gpd.GeoDataFrame(geometry=[interfaces, interfaces])
>>> gdf
    geometry
0   LINESTRING (2.0 -2.0, 5.0 -5.0)
1   LINESTRING (2.0 -2.0, 5.0 -5.0)
```

```
>>> # Extracting interfaces coordinates from cross sections
>>> gdf_points = gg.vector.extract_interfaces_coordinates_from_cross_
↪section(linestring=linestring, interfaces_gdf=gdf)
>>> gdf_points
    geometry                    X       Y       Z
```

(continues on next page)

(continued from previous page)

```
0   POINT (1.41421 -1.41421)    1.41    -1.41   -2.00
1   POINT (3.53553 -3.53553)    3.54    -3.54   -5.00
2   POINT (1.41421 -1.41421)    1.41    -1.41   -2.00
3   POINT (3.53553 -3.53553)    3.54    -3.54   -5.00
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for one LineString on cross sections

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

gemgis.vector.extract_xyz_from_cross_sections

gemgis.vector.**extract_xyz_from_cross_sections**(*profile_gdf: geopandas.geodataframe.GeoDataFrame*, *interfaces_gdf: geopandas.geodataframe.GeoDataFrame*, *profile_name_column: str = 'name'*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from cross sections and digitized interfaces

Parameters

- **profile_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the traces (LineStrings) of cross sections on a map and a profile name
- **interfaces_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the LineStrings of digitized interfaces, associated formation and the profile name
- **profile_name_column** (*str*) – Name of the profile column, default is `profile_name_column='name'`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z information of all extracted digitized interfaces on cross sections

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> import geopandas as gpd
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating GeoDataFrame from LineString and ad Profile names
>>> profile_gdf = gpd.GeoDataFrame(geometry=[linestring, linestring])
>>> profile_gdf['name'] = ['Profile1', 'Profile2']
>>> profile_gdf
    geometry    name
0       LINESTRING (0.0 0.0, 20.0 -20.0)        Profile1
1       LINESTRING (0.0 0.0, 20.0 -20.0)        Profile2
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating GeoDataFrame from LineString and ad Profile names
>>> gdf = gpd.GeoDataFrame(geometry=[interfaces, interfaces])
>>> gdf['name'] = ['Profile1', 'Profile2']
>>> gdf
    geometry    name
0       LINESTRING (2.0 -2.0, 5.0 -5.0) Profile1
1       LINESTRING (2.0 -2.0, 5.0 -5.0) Profile2
```

```
>>> # Extracting X, Y, and Z coordinates from cross sections
>>> gdf_points = gg.vector.extract_xyz_from_cross_sections(profile_gdf=profile_gdf,␣
↪interfaces_gdf=gdf)
>>> gdf_points
    name        geometry                        X       Y       Z
0       Profile1        POINT (1.41421 -1.41421)        1.41    -1.41   -2.00
1       Profile1        POINT (3.53553 -3.53553)        3.54    -3.54   -5.00
2       Profile2        POINT (1.41421 -1.41421)        1.41    -1.41   -2.00
3       Profile2        POINT (3.53553 -3.53553)        3.54    -3.54   -5.00
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for one LineString on cross sections

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

gemgis.vector.calculate_coordinates_for_point_on_cross_section

gemgis.vector.**calculate_coordinates_for_point_on_cross_section**(*linestring: shapely.geometry.linestring.LineString*, *point: Union[shapely.geometry.point.Point, Tuple[float, float]]*)

Calculating the coordinates for one point digitized on a cross section provided as Shapely LineString

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (20, 20)])`
- **point** (*Union[shapely.geometry.point.Point, Tuple[float, float]]*) – Shapely object or tuple of X and Y coordinates digitized on a cross section e.g. `point = Point(5, 0)`

Returns **point** – Shapely Point with real world X and Y coordinates extracted from cross section LineString on Map

Return type shapely.geometry.point.Point

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating Point
>>> point = Point(5, 0)
>>> point.wkt
'POINT (5 0)'
```

```
>>> # Calculating real world coordinates for point on cross section
>>> point_xy = gg.vector.calculate_coordinates_for_point_on_cross_
↪section(linestring=linestring, point=point)
>>> point_xy.wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

See also:

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for a LineString on a cross section

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

gemgis.vector.calculate_coordinates_for_linestring_on_cross_sections

gemgis.vector.**calculate_coordinates_for_linestring_on_cross_sections**(*linestring: shapely.geometry.linestring.LineString*, *interfaces: shapely.geometry.linestring.LineString*)

Calculating the coordinates of vertices for a LineString on a straight cross section provided as Shapely LineString

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (20, 20)])`
- **interfaces** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the interfaces points digitized on a cross section, e.g. `interfaces = LineString([(2, -2), (5, -5)])`

Returns **points** – List of Shapely Points with real world coordinates of digitized points on cross section

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Calculating coordinates for LineString on cross section
>>> points = gg.vector.calculate_coordinates_for_linestring_on_cross_
↪sections(linestring=linestring, interfaces=interfaces)
>>> points
[<shapely.geometry.point.Point at 0x231e8dc4d60>,
<shapely.geometry.point.Point at 0x231e5d9b070>]
```

```
>>> # Inspecting the first element of the list
>>> points[0].wkt
'POINT (1.414213562373095 -1.414213562373095)'
```

```
>>> # Inspecting the second element of the list
>>> points[1].wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

gemgis.vector.calculate_coordinates_for_linestrings_on_cross_sections

gemgis.vector.**calculate_coordinates_for_linestrings_on_cross_sections**(*linestring: shapely.geometry.linestring.LineString*, *linestring_interfaces_list: List[shapely.geometry.linestring.LineString*] → List[shapely.geometry.point.Point]

Calculating the coordinates of vertices for LineStrings on a straight cross section provided as Shapely LineString

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`
- **linestring_interfaces_list** (*List[shapely.geometry.linestring.LineString]*) – List containing Shapely LineStrings representing interfaces on cross sections

Returns **points** – List containing Shapely Points with real world coordinates of the digitized interfaces on the cross section

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating list of LineStrings
>>> linestring_interfaces_list = [interfaces, interfaces]
```

```
>>> # Calculating coordinates for LineStrings on cross section
>>> points = gg.vector.calculate_coordinates_for_linestrings_on_cross_
↪sections(linestring=linestring, linestring_interfaces_list=linestring_interfaces_
↪list)
>>> points
[<shapely.geometry.point.Point at 0x231e8019730>,
 <shapely.geometry.point.Point at 0x231e801e400>,
 <shapely.geometry.point.Point at 0x231e80192e0>,
 <shapely.geometry.point.Point at 0x231e80191f0>]
```

```
>>> # Inspecting the first element of the list
>>> points[0].wkt
'POINT (1.414213562373095 -1.414213562373095)'
```

```
>>> # Inspecting the second element of the list
>>> points[1].wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

```
>>> # Inspecting the third element of the list
>>> points[2].wkt
'POINT (1.414213562373095 -1.414213562373095)'
```

```
>>> # Inspecting the fourth element of the list
>>> points[3].wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for one LineString on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

8.1.4 Calculating and extracting angles from cross sections and maps

The following methods are used to calculate and extract angles from cross sections.

gemgis.vector.calculate_angle(linestring)	Calculating the angle of a LineString to the vertical
gemgis.vector.calculate_azimuth(gdf)	Calculating the azimuth for an orientation Geodataframe represented by LineStrings
gemgis.vector.calculate_strike_direction_straight_linestring(...)	Function to calculate the strike direction of a straight Shapely LineString.
gemgis.vector.calculate_strike_direction_bent_linestring(...)	Calculating the strike direction of a LineString with multiple elements
gemgis.vector.calculate_dipping_angle_linestring(...)	Calculating the dipping angle of a LineString digitized on a cross section
gemgis.vector.calculate_dipping_angles_linestrings(...)	Calculating the dipping angles of LineStrings digitized on a cross section
gemgis.vector.calculate_orientation_from_bent_cross_section(...)	Calculating the orientation of a LineString on a bent cross section provided as Shapely LineString
gemgis.vector.calculate_orientation_from_cross_section(...)	Calculating the orientation for one LineString on one cross sections
gemgis.vector.calculate_orientations_from_cross_sections(...)	Calculating orientations from a cross sections using multiple LineStrings
gemgis.vector.extract_orientations_from_cross_sections(...)	Calculating orientations digitized from cross sections
gemgis.vector.extract_orientations_from_map(gdf)	Calculating orientations from LineStrings
gemgis.vector.calculate_orientations_from_strike_lines(gdf)	Calculating orientations based on LineStrings representing strike lines
gemgis.vector.calculate_orientation_for_three_point_problem(gdf)	Calculating the orientation for a three point problem

gemgis.vector.calculate_angle

gemgis.vector.**calculate_angle**(*linestring: shapely.geometry.linestring.LineString*) → float

Calculating the angle of a LineString to the vertical

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString consisting of two vertices, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **angle** – Angle of a LineString to the vertical

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Calculating the strike angle of the LineString
>>> angle = gg.vector.calculate_angle(linestring=linestring)
>>> angle
135.0
```

See also:

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.calculate_azimuth

gemgis.vector.**calculate_azimuth**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) → List[Union[int, float]]

Calculating the azimuth for an orientation Geodataframe represented by LineStrings

Parameters **gdf** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame or list containing the LineStrings of orientations

Returns **azimuth_list** – List containing the azimuth values of the orientation LineString

Return type List[Union[float, int]]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring1 = LineString([(0, 0), (20, -20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(0, 0), (20, -10)])
>>> linestring2.wkt
'LINESTRING (0 0, 20 -10)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
>>> gdf
```

(continues on next page)

(continued from previous page)

```
    geometry
0       LINESTRING (0.0 0.0, 20.0 -20.0)
1       LINESTRING (0.0 0.0, 20.0 -10.0)
```

```
>>> # Calculating the azimuths of the LineStrings
>>> azimuths = gg.vector.calculate_azimuth(gdf=gdf)
>>> azimuths
[135.0, 116.56505117707799]
```

See also:

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.calculate_strike_direction_straight_linestring

gemgis.vector.**calculate_strike_direction_straight_linestring**(*linestring: shapely.geometry.linestring.LineString*) → float

Function to calculate the strike direction of a straight Shapely LineString. The strike will always be calculated from start to end point

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString representing the surface trace of a straight geological profile, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **angle** – Strike angle calculated from start to end point for a straight Shapely LineString

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Calculating the strike angle of the LineString
>>> angle = gg.vector.calculate_strike_direction_straight_
↪linestring(linestring=linestring)
>>> angle
45.0
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

Note: The LineString must only consist of two points (start and end point)

gemgis.vector.calculate_strike_direction_bent_linestring

gemgis.vector.**calculate_strike_direction_bent_linestring**(*linestring: shapely.geometry.linestring.LineString*) → List[float]

Calculating the strike direction of a LineString with multiple elements

Parameters **linestring** (*linestring: shapely.geometry.linestring.LineString*) – Shapely LineString containing more than two vertices, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **angles_splitted_linestrings** – List containing the strike angles of each line segment of the original

Return type List[float]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Calculating the strike angles for LineString elements
>>> angles = gg.vector.calculate_strike_direction_bent_
↪linestring(linestring=linestring)
>>> angles
[45.0, 45.0]
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

gemgis.vector.calculate_dipping_angle_linestring

gemgis.vector.**calculate_dipping_angle_linestring**(*linestring: shapely.geometry.linestring.LineString*)

Calculating the dipping angle of a LineString digitized on a cross section

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString digitized on a cross section, e.g. `linestring = LineString([(0, 0), (20, 20)])`

Returns **dip** – Dipping angle of the LineString

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating dipping angle from LineString
>>> angle = gg.vector.calculate_dipping_angle_linestring(linestring=linestring)
>>> angle
45.0
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.calculate_dipping_angles_linestrings

gemgis.vector.**calculate_dipping_angles_linestrings**(*linestring_list: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*)

Calculating the dipping angles of LineStrings digitized on a cross section

Parameters **linestring_list** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame containing LineStrings or list of LineStrings

Returns **dipping_angles** – List containing the dipping angles of LineStrings

Return type List[float]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating list of LineStrings
>>> linestring_list = [linestring, linestring]
```

```
>>> # Calculating dipping angles for LineStrings
>>> angles = gg.vector.calculate_dipping_angles_linestrings(linestring_
↪list=linestring_list)
>>> angles
[45.0, 45.0]
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.calculate_orientation_from_bent_cross_section

gemgis.vector.**calculate_orientation_from_bent_cross_section**(*profile_linestring: shapely.geometry.linestring.LineString*, *orientation_linestring: shapely.geometry.linestring.LineString*) → list

Calculating the orientation of a LineString on a bent cross section provided as Shapely LineString

Parameters

- **profile_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map e.g. `profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])`
- **orientation_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString representing an orientation measurement on the cross section, e.g. `orientation_linestring = LineString([(2, -2), (5, -5)])`

Returns **orientation** – List containing a Shapely Point with X and Y coordinates, the Z value, dip, azimuth and polarity values

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])
>>> profile_linestring.wkt
'LINESTRING (0 0, 5 10, 20 20)'
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Calculating the orientation from a bent cross section
>>> orientations = gg.vector.calculate_orientation_from_bent_cross_section(profile_
↪linestring=profile_linestring, orientation_linestring=orientation_linestring)
>>> orientations
[<shapely.geometry.point.Point at 0x231e7f00820>, -3.5, 45.0, 26.565051177078004, 1]
```

```
>>> # Inspecting the Point object of the list
>>> orientations[0].wkt
'POINT (1.565247584249853 3.130495168499706)'
```

See also:

calculate_orientation_from_cross_section Calculating the orientation of a LineString on a cross section

calculate_orientations_from_cross_section Calculating orientations for LineStrings on a cross section

extract_orientations_from_cross_sections Calculating the orientations for LineStrings on cross sections

gemgis.vector.calculate_orientation_from_cross_section

gemgis.vector.**calculate_orientation_from_cross_section**(*profile_linestring: shapely.geometry.linestring.LineString*, *orientation_linestring: shapely.geometry.linestring.LineString*) → list

Calculating the orientation for one LineString on one cross sections

Parameters

- **profile_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `profile_linestring = LineString([(0, 0), (20, 20)])`
- **orientation_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString representing an orientation measurement on the cross section e.g. `orientation_linestring = LineString([(2, -2), (5, -5)])`

Returns **orientation** – List containing a Shapely Point with X and Y coordinates, the Z value, dip, azimuth and polarity values

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> profile_linestring = LineString([(0, 0), (20, 20)])
>>> profile_linestring.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Calculating orientation orientation from cross section
>>> orientations = gg.vector.calculate_orientation_from_cross_section(profile_
↪linestring=profile_linestring, orientation_linestring=orientation_linestring)
>>> orientations
[<shapely.geometry.point.Point at 0x231e79a5370>, -3.5, 45.0, 45.0, 1]
```

```
>>> # Inspecting the Point object of the list
>>> orientations[0].wkt
'POINT (2.474873734152916 2.474873734152916)'
```

See also:

calculate_orientation_from_bent_cross_section Calculating the orientation of a LineString on a bent cross section

calculate_orientations_from_cross_section Calculating orientations for LineStrings on a cross section

extract_orientations_from_cross_sections Calculating the orientations for LineStrings on cross sections

gemgis.vector.calculate_orientations_from_cross_section

gemgis.vector.**calculate_orientations_from_cross_section**(*profile_linestring: shapely.geometry.linestring.LineString*, *orientation_linestrings: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*, *extract_coordinates: bool = True*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations from a cross sections using multiple LineStrings

Parameters

- **profile_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])`
- **orientations_linestrings** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame or list containing multiple orientation LineStrings
- **extract_coordinates** (*bool*) – Variable to extract the X and Y coordinates from point objects. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing the Shapely Points with X, Y coordinates, the Z value, dips, azimuths and polarities

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])
>>> profile_linestring.wkt
'LINESTRING (0 0, 5 10, 20 20)'
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating List of LineStrings
>>> orientations_list = [orientation_linestring, orientation_linestring]
```

```
>>> # Calculating orientations from cross sections
>>> orientations = gg.vector.calculate_orientations_from_cross_section(profile_
↪linestring=profile_linestring, orientation_linestrings=orientations_list)
>>> orientations
    X       Y       Z       dip     azimuth polarity    geometry
0   1.57    3.13    -3.50   45.00   26.57   1.00        POINT (1.56525 3.13050)
1   1.57    3.13    -3.50   45.00   26.57   1.00        POINT (1.56525 3.13050)
```

See also:

calculate_orientation_from_cross_section Calculating the orientation of a LineString on a cross section

calculate_orientation_from_bent_cross_section Calculating orientations of a LineStrings on a bent cross section

extract_orientations_from_cross_sections Calculating the orientations for LineStrings on cross sections

gemgis.vector.extract_orientations_from_cross_sections

gemgis.vector.**extract_orientations_from_cross_sections**(*profile_gdf: geopandas.geodataframe.GeoDataFrame*, *orientations_gdf: geopandas.geodataframe.GeoDataFrame*, *profile_name_column: str = 'name'*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations digitized from cross sections

Parameters

- **profile_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the different profile traces as LineStrings
- **orientations_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the orientation LineStrings for different profiles and formations
- **profile_name_column** (*str*) – Name of the profile column, e.g. `profile_name_column='name'`, default is `'name'`

Returns **gdf** – GeoDataFrame containing the orientation and location data for orientations digitized on cross sections

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> import geopandas as gpd
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating GeoDataFrame from LineString and adding profile names
>>> profile_gdf = gpd.GeoDataFrame(geometry=[linestring, linestring])
>>> profile_gdf['name'] = ['Profile2', 'Profile1']
>>> profile_gdf
    geometry                              name
0   LINESTRING (0.0 0.0, 20.0 -20.0)    Profile2
1   LINESTRING (0.0 0.0, 20.0 -20.0)    Profile1
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating GeoDataFrame from LineString and adding profile names
>>> orientations_gdf = gpd.GeoDataFrame(geometry=[orientation_linestring,␣
↪orientation_linestring])
>>> orientations_gdf
    geometry                          name
0   LINESTRING (2.0 -2.0, 5.0 -5.0) Profile2
1   LINESTRING (2.0 -2.0, 5.0 -5.0) Profile1
```

```
>>> # Extract orientations from cross sections
>>> orientations = gg.vector.extract_orientations_from_cross_sections(profile_
↪gdf=profile_gdf, orientations_gdf=orientations_gdf)
>>> orientations
    X       Y       Z       dip     azimuth     polarity    geometry              ␣
↪    name
0   2.47    -2.47   -3.50   45.00   135.00      1.00        POINT (2.47487 -2.
↪47487)    Profile2
1   2.47    -2.47   -3.50   45.00   135.00      1.00        POINT (2.47487 -2.
↪47487)    Profile1
```

gemgis.vector.extract_orientations_from_map

gemgis.vector.**extract_orientations_from_map**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dz: str = 'dZ'*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations from LineStrings

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the orientation LineStrings
- **dz** (*str*) – Name of the height difference column, e.g. dz='dZ'

Returns **gdf** – GeoDataFrame containing the orientation values

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring1 = LineString([(0, 0), (20, -20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(0, 0), (20, -10)])
>>> linestring2.wkt
'LINESTRING (0 0, 20 -10)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
>>> gdf['dZ'] = [100, 200]
>>> gdf
    geometry                            dz
0   LINESTRING (0.0 0.0, 20.0 -20.0)    100
1   LINESTRING (0.0 0.0, 20.0 -10.0)    200
```

```
>>> # Extracting orientations from map
>>> orientations = gg.vector.extract_orientations_from_map(gdf=gdf)
>>> orientations
    geometry            azimuth dip     X       Y       polarity
0   POINT (10.0 -10.0)  135.00  74.21   10.00   -10.00  1
1   POINT (10.0 -5.0)   116.57  83.62   10.00   -5.00   1
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.calculate_orientations_from_strike_lines

gemgis.vector.**calculate_orientations_from_strike_lines**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations based on LineStrings representing strike lines

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing LineStrings representing strike lines

Returns **gdf_orient** – GeoDataFrame containing the location of orientation measurements and their associated orientation values

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Changed in version 1.1.7.

Fixing indexing issue.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring1 = LineString([(0, 0), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Create second LineString
>>> linestring2 = LineString([(0, 10), (20, 30)])
>>> linestring2.wkt
'LINESTRING (0 10, 20 30)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
>>> gdf['Z'] = [100,200]
>>> gdf['id'] = [1,2]
>>> gdf
    geometry                              Z   id
0       LINESTRING (0.0 0.0, 20.0 20.0)     100 1
1       LINESTRING (0.0 10.0, 20.0 30.0)    200 2
```

```
>>> # Calculating orientations strike lines
>>> orientations = gg.vector.calculate_orientations_from_strike_lines(gdf=gdf)
>>> orientations
    dip     azimuth     Z           geometry            polarity        X           ␣
↪Y
0       85.96   135.00  150.00  POINT (10.0 15.0)       1.00        10.00       15.
↪00
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between two LineStrings

gemgis.vector.calculate_orientation_for_three_point_problem

gemgis.vector.**calculate_orientation_for_three_point_problem**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Calculating the orientation for a three point problem

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the three points and respective altitudes

Returns **orientation** – GeoDataFrame containing the calculated orientation value

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
>>> import geopandas as gpd
>>> points = gpd.read_file(filename='points.shp')
>>> points
    id  formation   Z   geometry
0   None    Coal    200 POINT (1842.732 602.462)
1   None    Coal    400 POINT (1696.262 1775.038)
2   None    Coal    600 POINT (104.302 1770.385)
```

```
>>> # Calculating Orientation
>>> orientation = gg.vector.calculate_orientation_for_three_point_
↪problem(gdf=points)
>>> orientation
    Z       formation   azimuth dip     polarity    X       Y       geometry
0   400.0   Coal        140.84  11.29   1           1214.43 1382.63 POINT (1214.432␣
↪1382.628)
```

8.1.5 Exploding Geometries

The following methods are used to explode geometries for further usage in GemGIS

gemgis.vector.explode_linestring(linestring)	Exploding a LineString to its vertices, also works for LineStrings with Z components
gemgis.vector.explode_linestring_to_elements(...)	Separating a LineString into its single elements and returning a list of LineStrings representing these elements, also works for LineStrings with Z components
gemgis.vector.explode_multilinestring(...)	Exploding a MultiLineString into a list of LineStrings
gemgis.vector.explode_multilinestrings(gdf)	Exploding Shapely MultiLineStrings stored in a GeoDataFrame to Shapely LineStrings
gemgis.vector.explode_polygon(polygon)	Exploding Shapely Polygon to list of Points
gemgis.vector.explode_polygons(gdf)	Converting a GeoDataFrame containing elements of geom_type Polygons to a GeoDataFrame with LineStrings
gemgis.vector.explode_geometry_collection(...)	Exploding a Shapely Geometry Collection to a List of Base Geometries
gemgis.vector.explode_geometry_collections(gdf)	Exploding Shapely Geometry Collections stored in GeoDataFrames to different Shapely Base Geometries

gemgis.vector.explode_linestring

gemgis.vector.**explode_linestring**(*linestring: shapely.geometry.linestring.LineString*) → List[shapely.geometry.point.Point]

Exploding a LineString to its vertices, also works for LineStrings with Z components

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString from which vertices are extracted, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **points_list** – List of extracted Shapely Points

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Exploding LineString into single points
>>> linestring_exploded = gg.vector.explode_linestring(linestring=linestring)
>>> linestring_exploded
[<shapely.geometry.point.Point at 0x20118cb27f0>,
<shapely.geometry.point.Point at 0x20118cb28b0>,
<shapely.geometry.point.Point at 0x20118cb26d0>]
```

```
>>> # Inspecting the first element of the list
>>> linestring_exploded[0].wkt
'POINT (0 0)'
```

```
>>> # Inspecting the second element of the list
>>> linestring_exploded[1].wkt
'POINT (10 10)'
```

```
>>> # Inspecting the third element of the list
>>> linestring_exploded[2].wkt
'POINT (20 20)'
```

See also:

explode_linestring_to_elements Exploding a LineString with more than two vertices into single LineStrings

gemgis.vector.explode_linestring_to_elements

gemgis.vector.**explode_linestring_to_elements**(*linestring: shapely.geometry.linestring.LineString*) → List[shapely.geometry.linestring.LineString]

Separating a LineString into its single elements and returning a list of LineStrings representing these elements, also works for LineStrings with Z components

Parameters **linestring** (*linestring: shapely.geometry.linestring.LineString*) – Shapely LineString containing more than two vertices, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **splitted_linestrings** – List containing the separate elements of the original LineString stored as LineStrings

Return type List[shapely.geometry.linestring.LineString]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Exploding LineString into single elements
>>> linestring_exploded = gg.vector.explode_linestring_to_
↪elements(linestring=linestring)
>>> linestring_exploded
[<shapely.geometry.linestring.LineString at 0x201448a2100>,
<shapely.geometry.linestring.LineString at 0x20144b5e610>]
```

```
>>> # Inspecting the first element of the list
>>> linestring_exploded[0].wkt
'LINESTRING (0 0, 10 10)'
```

```
>>> # Inspecting the second element of the list
>>> linestring_exploded[1].wkt
'LINESTRING (10 10, 20 20)'
```

See also:

explode_linestring Exploding a LineString into its single vertices

gemgis.vector.explode_multilinestring

gemgis.vector.**explode_multilinestring**(*multilinestring: shapely.geometry.multilinestring.MultiLineString*) → List[shapely.geometry.linestring.LineString]

Exploding a MultiLineString into a list of LineStrings

Parameters **multilinestring** (*shapely.geometry.multilinestring.MultiLineString*) – Shapely MultiLineString consisting of multiple LineStrings, e.g. `multilinestring = MultiLineString([((0, 0), (1, 1)), ((-1, 0), (1, 0))])`

Returns **splitted_multilinestring** – List of Shapely LineStrings

Return type List[shapely.geometry.linestring.LineString]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating MultiLineString
>>> import gemgis as gg
>>> from shapely.geometry import MultiLineString
>>> coords = [((0, 0), (1, 1)), ((-1, 0), (1, 0))]
>>> lines = MultiLineString(coords)
>>> lines.wkt
'MULTILINESTRING ((0 0, 1 1), (-1 0, 1 0))'
```

```
>>> lines_splitted = gg.vector.explode_multilinestrings(multilinestring=lines)
>>> lines_splitted
[<shapely.geometry.linestring.LineString at 0x2014a5f0ee0>,
<shapely.geometry.linestring.LineString at 0x20149dda430>]
```

```
>>> # Inspecting the first element of the list
>>> lines_splitted[0].wkt
'LINESTRING (0 0, 1 1)'
```

```
>>> # Inspecting the second element of the list
>>> lines_splitted[1].wkt
'LINESTRING (-1 0, 1 0)'
```

See also:

explode_multilinestrings Exploding a GeoDataFrame containing MultiLineStrings into a GeoDataFrame containing LineStrings only

gemgis.vector.explode_multilinestrings

gemgis.vector.**explode_multilinestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*) → geopandas.geodataframe.GeoDataFrame

Exploding Shapely MultiLineStrings stored in a GeoDataFrame to Shapely LineStrings

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type MultiLineString
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing LineStrings

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    geometry
0   MULTILINESTRING ((0.0 0.0, 1.0 1.0))
1   MULTILINESTRING ((0.0 0.0, 1.0 1.0))
```

```
>>> # Exploding MultiLineStrings into single LineStrings
>>> gdf_linestrings = gg.vector.explode_multilinestrings(gdf=gdf, reset_index=True)
>>> gdf_linestrings
    geometry
0       LINESTRING (0.0 0.0, 1.0 1.0)
1       LINESTRING (-1.0 0.0, 1.0 0.0)
2       LINESTRING (0.0 0.0, 1.0 1.0)
3       LINESTRING (-1.0 0.0, 1.0 0.0)
```

See also:

explode_multilinestring Exploding a MultiLineString into a list of single LineStrings

gemgis.vector.explode_polygon

gemgis.vector.**explode_polygon**(*polygon: shapely.geometry.polygon.Polygon*) → List[shapely.geometry.point.Point]

Exploding Shapely Polygon to list of Points

Parameters **polygon** (*shapely.geometry.polygon.Polygon*) – Shapely Polygon from which vertices are extracted, e.g. `polygon = Polygon([(0, 0), (1, 1), (1, 0)])`

Returns **point_list** – List containing the vertices of a polygon as Shapely Points

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon = Polygon([(0, 0), (1, 1), (1, 0)])
>>> polygon.wkt
'POLYGON ((0 0, 1 1, 1 0, 0 0))'
```

```
>>> # Exploding Polygon into single Points
>>> polygon_exploded = gg.vector.explode_polygon(polygon=polygon)
>>> polygon_exploded
[<shapely.geometry.point.Point at 0x201459734f0>,
<shapely.geometry.point.Point at 0x20145973670>,
<shapely.geometry.point.Point at 0x20145973640>,
<shapely.geometry.point.Point at 0x201459732e0>]
```

```
>>> # Inspecting the first element of the list
>>> polygon_exploded[0].wkt
'POINT (0 0)'
```

```
>>> # Inspecting the second element of the list
>>> polygon_exploded[1].wkt
'POINT (1 1)'
```

See also:

explode_polygons Exploding a GeoDataFrame containing Polygons into a GeoDataFrame containing LineStrings

gemgis.vector.explode_polygons

gemgis.vector.**explode_polygons**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Converting a GeoDataFrame containing elements of geom_type Polygons to a GeoDataFrame with LineStrings

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type Polygon

Returns **gdf_linestrings** – GeoDataFrame containing elements of type MultiLineString and LineString

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    geometry
0       POLYGON ((0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0))
1       POLYGON ((0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0))
```

```
>>> # Exploding Polygons into LineStrings
>>> gdf_exploded = gg.vector.explode_polygons(gdf=gdf)
>>> gdf_exploded
    geometry
0       LINESTRING (0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0)
1       LINESTRING (0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0)
```

See also:

explode_polygon Exploding a Polygon into single Points

gemgis.vector.explode_geometry_collection

gemgis.vector.**explode_geometry_collection**(*collection: shapely.geometry.collection.GeometryCollection*) → List[shapely.geometry.base.BaseGeometry]

Exploding a Shapely Geometry Collection to a List of Base Geometries

Parameters **collection** (*shapely.geometry.collection.GeometryCollection*) – Shapely Geometry Collection consisting of different Base Geometries

Returns **collection_exploded** – List of Base Geometries from the original Geometry Collection

Return type List[shapely.geometry.base.BaseGeometry]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Geometry Collection
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> a = LineString([(0, 0), (1, 1), (1,2), (2,2)])
>>> b = LineString([(0, 0), (1, 1), (2,1), (2,2)])
>>> collection = a.intersection(b)
>>> collection.wkt
'GEOMETRYCOLLECTION (POINT (2 2), LINESTRING (0 0, 1 1))'
```

```
>>> # Exploding Geometry collection into single Base Geometries
>>> collection_exploded = gg.vector.explode_geometry_
↪collection(collection=collection)
>>> collection_exploded
[<shapely.geometry.point.Point at 0x1faf63ccac0>,
<shapely.geometry.linestring.LineString at 0x1faf63ccb80>]
```

```
>>> # Inspecting the first element of the list
>>> collection_exploded[0].wkt
'POINT (2 2)'
```

```
>>> # Inspecting the second element of the list
>>> collection_exploded[1].wkt
'LINESTRING (0 0, 1 1)'
```

See also:

explode_geometry_collections Exploding a GeoDataFrame containing different Base Geometries

gemgis.vector.explode_geometry_collections

gemgis.vector.**explode_geometry_collections**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *remove_points: bool = True*) → geopandas.geodataframe.GeoDataFrame

Exploding Shapely Geometry Collections stored in GeoDataFrames to different Shapely Base Geometries

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type GeometryCollection
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **remove_points** (*bool*) – Variable to remove points from exploded GeoDataFrame. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing different geometry types

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Geometries
>>> import gemgis as gg
>>> from shapely.geometry import LineString, Polygon
>>> import geopandas as gpd
>>> a = LineString([(0, 0), (1, 1), (1,2), (2,2)])
>>> b = LineString([(0, 0), (1, 1), (2,1), (2,2)])
>>> collection = a.intersection(b)
>>> polygon = Polygon([(0, 0), (10, 0), (10, 10), (0, 10)])
```

```
>>> # Creating GeoDataFrame from Base Geometries
>>> gdf = gpd.GeoDataFrame(geometry=[a, b, collection, polygon])
>>> gdf
    geometry
0       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
1       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
2       GEOMETRYCOLLECTION (POINT (2.00000 2.00000), L...
3       POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1..
```

```
>>> # Explode Geometry Collection into single Base Geometries
>>> gdf_exploded = gg.vector.explode_geometry_collections(gdf=gdf)
>>> gdf_exploded
    geometry
0       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
1       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
2       LINESTRING (0.00000 0.00000, 1.00000 1.00000)
3       POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1...
```

See also:

explode_geometry_collection Exploding a Shapely Geometry Collection Object into a list of Base Geometries

8.1.6 Removing Points within Buffers

The following methods are used to remove Points within Buffers. This can be used to remove interface points in the vicinity of faults.

gemgis.vector.remove_object_within_buffer(...)	Removing object from a buffered object by providing a distance
gemgis.vector.remove_objects_within_buffer(...)	Removing objects from a buffered object by providing a distance
gemgis.vector.remove_interfaces_within_fault_buffer(...)	Functions to create a buffer around a GeoDataFrame containing fault data and removing interface points that are located within this buffer

gemgis.vector.remove_object_within_buffer

gemgis.vector.**remove_object_within_buffer**(*buffer_object: shapely.geometry.base.BaseGeometry*, *buffered_object: shapely.geometry.base.BaseGeometry*, *distance: Union[int, float] = None*, *buffer: bool = True*) → Tuple[shapely.geometry.base.BaseGeometry, shapely.geometry.base.BaseGeometry]

Removing object from a buffered object by providing a distance

Parameters

- **buffer_object** (*shapely.geometry.base.BaseGeometry*) – Shapely object for which a buffer will be created, e.g. `buffer_object=Point(0, 0)`
- **buffered_object** (*shapely.geometry.base.BaseGeometry*) – Shapely object that will be removed from the buffer, e.g. `buffered_object=LineString([(0, 0), (10, 10), (20, 20)])`
- **distance** (*Union[float, int]*) – Distance of the buffer around the geometry object, e.g. `distance=10`, default is `None`
- **buffer** (*bool*) – Variable to create a buffer. Options include: `True` or `False`, default set to `True`

Returns

- **result_out** (*shapely.geometry.base.BaseGeometry*) – Shapely object that remains after the buffering (outside the buffer)
- **result_in** (*shapely.geometry.base.BaseGeometry*) – Shapely object that was buffered (inside the buffer)

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> point = Point(0, 0)
>>> point.wkt
'POINT (0 0)'
```

```
>>> # Creating LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Removing object within buffer
>>> result_out, result_in = gg.vector.remove_object_within_buffer(buffer_
↪object=point, buffered_object=linestring, distance=10)
```

```
>>> # Inspecting the Base Geometry that remains outside
>>> result_out.wkt
'LINESTRING (7.071067811865473 7.071067811865473, 10 10, 20 20)'
```

```
>>> # Inspecting the Base Geometry that remains inside
>>> result_in.wkt
'LINESTRING (0 0, 7.071067811865473 7.071067811865473)'
```

See also:

remove_objects_within_buffer Removing several objects from one buffered object

remove_interfaces_within_fault_buffers Removing interfaces of layer boundaries within fault line buffers

gemgis.vector.remove_objects_within_buffer

gemgis.vector.**remove_objects_within_buffer**(*buffer_object: shapely.geometry.base.BaseGeometry*, *buffered_objects_gdf: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]]*, *distance: Union[int, float] = None*, *return_gdfs: bool = False*, *remove_empty_geometries: bool = False*, *extract_coordinates: bool = False*, *buffer: bool = True*) → Tuple[Union[List[shapely.geometry.base.BaseGeometry], geopandas.geodataframe.GeoDataFrame], Union[List[shapely.geometry.base.BaseGeometry], geopandas.geodataframe.GeoDataFrame]]

Removing objects from a buffered object by providing a distance

Parameters

- **buffer_object** (*shapely.geometry.base.BaseGeometry*) – Shapely object for which a buffer will be created, e.g. `buffer_object=Point(0, 0)`
- **buffered_object_gdf** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]]*) – GeoDataFrame or List of Base Geometries containing Shapely objects that will be buffered by the buffer object
- **distance** (*float, int*) – Distance of the buffer around the geometry object, e.g. `distance=10`
- **return_gdfs** (*bool*) – Variable to create GeoDataFrames of the created list of Shapely Objects. Options include: `True` or `False`, default set to `False`
- **remove_empty_geometries** (*bool*) – Variable to remove empty geometries. Options include: `True` or `False`, default set to `False`
- **extract_coordinates** (*bool*) – Variable to extract X and Y coordinates from resulting Shapely Objects. Options include: `True` or `False`, default set to `False`
- **buffer** (*bool*) – Variable to create a buffer. Options include: `True` or `False`, default set to `True`

Returns

- **result_out** (*list, gpd.geodataframe.GeoDataFrame*) – List or GeoDataFrame of Shapely objects that remain after the buffering (outside the buffer)
- **result_in** (*list, gpd.geodataframe.GeoDataFrame*) – List or GeoDataFrame of Shapely objects that was buffered (inside the buffer)

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> point = Point(0, 0)
>>> point.wkt
'POINT (0 0)'
```

```
>>> # Creating first LineString
>>> linestring1 = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(10, 0), (20, 10), (30, 20)])
>>> linestring2.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Create list of buffer objects
>>> buffer_objects = [linestring1, linestring2]
```

```
>>> # Removing objects within buffer
>>> result_out, result_in = gg.vector.remove_objects_within_buffer(buffer_
↪object=point, buffered_object_gdf=buffer_objects, distance=10)
```

```
>>> # Inspecting the Base Geometries that remain outside
>>> result_out
[<shapely.geometry.linestring.LineString at 0x2515421e4f0>,
<shapely.geometry.linestring.LineString at 0x2515421e3d0>]
```

```
>>> # Inspecting the Base Geometries that remain inside
>>> result_in
[<shapely.geometry.linestring.LineString at 0x2515421e310>,
<shapely.geometry.linestring.LineString at 0x2515421e6a0>]
```

See also:

remove_object_within_buffer Removing one object from one buffered object

remove_interfaces_within_fault_buffers Removing interfaces of layer boundaries within fault line buffers

gemgis.vector.remove_interfaces_within_fault_buffers

gemgis.vector.**remove_interfaces_within_fault_buffers**(*fault_gdf: geopandas.geodataframe.GeoDataFrame*, *interfaces_gdf: geopandas.geodataframe.GeoDataFrame*, *distance: Union[int, float] = None*, *remove_empty_geometries: bool = True*, *extract_coordinates: bool = True*) → Tuple[geopandas.geodataframe.GeoDataFrame, geopandas.geodataframe.GeoDataFrame]

Function to create a buffer around a GeoDataFrame containing fault data and removing interface points that are located within this buffer

Parameters

- **fault_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the fault data
- **interfaces_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the interface point data
- **distance** (*float, int*) – Distance of the buffer around the geometry object, e.g. `distance=10`
- **remove_empty_geometries** (*bool*) – Variable to remove empty geometries, Options include: `True` or `False` default `True`
- **extract_coordinates** (*bool*) – Variable to extract X and Y coordinates from resulting Shapely Objects, Options include: `True` or `False` default `True`

Returns

- **gdf_out** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the vertices located outside the fault buffer
- **gdf_in** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the vertices located inside the fault buffer

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
>>> import geopandas as gpd
>>> from shapely.geometry import Point, LineString
```

```
>>> # Creating first Point
>>> point1 = Point(0, 0)
>>> point1.wkt
'POINT (0 0)'
```

```
>>> # Creating second Point
>>> point2 = Point(5, 0)
>>> point2.wkt
'POINT (5 0)'
```

```
>>> # Creating GeoDataFrame from Points
>>> fault_gdf = gpd.GeoDataFrame(geometry=[point1, point2])
```

```
>>> # Creating first LineString
>>> linestring1 = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(10, 0), (20, 10), (30, 20)])
>>> linestring2.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> buffer_objects_gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
```

```
>>> # Removing interfaces within fault buffers
>>> result_out, result_in = gg.vector.remove_interfaces_within_fault_buffers(fault_
↪gdf=fault_gdf, interfaces_gdf=buffer_objects_gdf, distance=10)
```

```
>>> # Inspecting the Base Geometries that remain outside
>>> result_out
    geometry                    X       Y
0       POINT (7.07107 7.07107)         7.07    7.07
1       POINT (10.00000 10.00000)       10.00   10.00
2       POINT (20.00000 20.00000)       20.00   20.00
3       POINT (10.00000 0.00000)        10.00   0.00
4       POINT (20.00000 10.00000)       20.00   10.00
5       POINT (30.00000 20.00000)       30.00   20.00
```

```
>>> # Inspecting the Base Geometries that remain inside
>>> result_in
    geometry            X       Y
0       POINT (0.00000 0.00000) 0.00    0.00
1       POINT (7.07107 7.07107) 7.07    7.07
```

See also:

remove_object_within_buffer Removing one object from one buffered object

remove_objects_within_buffer Removing several objects from one buffered object

8.1.7 Vector Methods for Raster Data

The following methods are used to work with raster data

gemgis.vector.interpolate_raster(gdf[, ...])	Interpolating a raster/digital elevation model from point or line Shape file

gemgis.vector.interpolate_raster

gemgis.vector.**interpolate_raster**(*gdf: geopandas.geodataframe.GeoDataFrame*, *value: str = 'Z'*, *method: str = 'nearest'*, *n: int = None*, *res: int = 1*, *extent: List[Union[int, float]] = None*, *seed: int = None*, ***kwargs*) → numpy.ndarray

Interpolating a raster/digital elevation model from point or line Shape file

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing vector data of geom_type Point or Line containing the Z values of an area
- **value** (*str*) – Value to be interpolated, e.g. `value='Z'`, default is `'Z'`
- **method** (*string*) – Method used to interpolate the raster. Options include: `'nearest'`, `'linear'`, `'cubic'`, `'rbf'`
- **res** (*int*) – Resolution of the raster in X and Y direction, e.g. `res=50`
- **seed** (*int*) – Seed for the drawing of random numbers, e.g. `seed=1`
- **n** (*int*) – Number of samples used for the interpolation, e.g. `n=100`
- **extent** (*List[Union[float, int]]*) – Values for minx, maxx, miny and maxy values to define the boundaries of the raster, e.g. `extent=[0, 972, 0, 1069]`
- ****kwargs** (*optional keyword arguments*) – For kwargs for rbf and griddata see: https://docs.scipy.org/doc/scipy/reference/interpolate.html

Returns **array** – Array representing the interpolated raster/digital elevation model

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id  Z       geometry
0       None    400     LINESTRING (0.741 475.441, 35.629 429.247, 77....
1       None    300     LINESTRING (645.965 0.525, 685.141 61.866, 724...
2       None    400     LINESTRING (490.292 0.525, 505.756 40.732, 519...
3       None    600     LINESTRING (911.433 1068.585, 908.856 1026.831...
4       None    700     LINESTRING (228.432 1068.585, 239.772 1017.037...
```

```
>>> # Interpolating vector data
>>> raster = gg.vector.interpolate_raster(gdf=gdf, method='rbf')
>>> raster[:2]
array([[393.56371914, 393.50838517, 393.45386851, ..., 396.15856133,
    398.11421775, 400.06334288],
   [393.41982945, 393.36494645, 393.31088433, ..., 396.20694282,
    398.16690286, 400.12027997]])
```

8.1.8 Working with GPX Data

The following methods are used to work with GPX data

gemgis.vector.load_gpx(path[, layer])	Loading a GPX file as collection
gemgis.vector.load_gpx_as_dict(path[, layer])	Loading a GPX file as dict
gemgis.vector.load_gpx_as_geometry(path[, layer])	Loading a GPX file as Shapely Geometry

gemgis.vector.load_gpx

gemgis.vector.**load_gpx**(*path: str*, *layer: Union[int, str] = 'tracks'*) → Collection

Loading a GPX file as collection

Parameters

- **path** (*str*) – Path to the GPX file, e.g. `path='file.gpx'`
- **layer** (*Union[int, str]*) – The integer index or name of a layer in a multi-layer dataset, e.g. `layer='tracks'`, default is `tracks`

Returns **gpx** – Collection containing the GPX data

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> gpx = gg.vector.load_gpx(path='file.gpx', layer='tracks')
>>> gpx
<open Collection 'file.gpx:tracks', mode 'r' at 0x24f1c90ffa0>
```

See also:

load_gpx_as_dict Loading a GPX file as dict

load_gpx_as_geometry Loading a GPX file as Shapely BaseGeometry

gemgis.vector.load_gpx_as_dict

gemgis.vector.**load_gpx_as_dict**(*path: str*, *layer: Union[int, str] = 'tracks'*) → Collection

Loading a GPX file as dict

Parameters

- **path** (*str*) – Path to the GPX file, e.g. `path='file.gpx'`
- **layer** (*Union[int, str]*) – The integer index or name of a layer in a multi-layer dataset, e.g. `layer='tracks'`, default is `tracks`

Returns **gpx_dict** – Dict containing the GPX data

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> gpx = gg.vector.load_gpx_as_dict(path='file.gpx', layer='tracks')
>>> gpx
{'type': 'Feature',
 'id': '0',
 'properties': OrderedDict([('name',
               'First half marathon distance of the year'),
              ('cmt', None),
              ('desc', None),
              ('src', None),
              ('link1_href', None),
              ('link1_text', None),
              ('link1_type', None),
              ('link2_href', None),
              ('link2_text', None),
              ('link2_type', None),
              ('number', None),
              ('type', '9')]),
 'geometry': {'type': 'MultiLineString',
  'coordinates': [[(8.496285, 52.705566),
    (8.49627, 52.705593),
    (8.496234, 52.705629),...]]}}
```

See also:

load_gpx_as Loading a GPX file as Collection

load_gpx_as_geometry Loading a GPX file as Shapely BaseGeometry

gemgis.vector.load_gpx_as_geometry

gemgis.vector.**load_gpx_as_geometry**(*path: str*, *layer: Union[int, str] = 'tracks'*) → shapely.geometry.base.BaseGeometry

Loading a GPX file as Shapely Geometry

Parameters

- **path** (*str*) – Path to the GPX file, e.g. `path='file.gpx'`
- **layer** (*Union[int, str]*) – The integer index or name of a layer in a multi-layer dataset, e.g. `layer='tracks'`, default is `tracks`

Returns **shape** – Shapely BaseGeometry containing the geometry data of the GPX file

Return type shapely.geometry.base.BaseGeometry

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> gpx = gg.vector.load_gpx_as_geometry(path='file.gpx', layer='tracks')
>>> gpx.wkt
'MULTILINESTRING ((8.496285 52.705566, 8.496270000000001 52.705593, 8.
↪496233999999999 52.705629, 8.496205
52.705664, 8.496181 52.705705, 8.496171 52.705754,...)
```

See also:

load_gpx Loading a GPX file as Collection

load_gpx_as_dict Loading a GPX file as dict

8.1.9 Miscellaneous vector data methods

The following methods are further vector data methods used in GemGIS

gemgis.vector.calculate_distance_linestrings(...)	Calculating the minimal distance between two LineStrings
gemgis.vector.calculate_midpoint_linestring(...)	Calculating the midpoint of a LineString with two vertices
gemgis.vector.calculate_midpoints_linestrings(...)	Calculating the midpoints of LineStrings with two vertices each
gemgis.vector.clip_by_bbox(gdf, bbox[, ...])	Clipping vector data contained in a GeoDataFrame to a provided bounding box/extent
gemgis.vector.clip_by_polygon(gdf, polygon)	Clipping vector data contained in a GeoDataFrame to a provided bounding box/extent
gemgis.vector.create_bbox(extent)	Creating a rectangular polygon from the provided bounding box values, with counter-clockwise order by default.
gemgis.vector.create_buffer(geom_object, ...)	Creating a buffer around a Shapely LineString or a Point
gemgis.vector.create_unified_buffer(...)	Creating a unified buffer around Shapely LineStrings or Points
gemgis.vector.create_linestring_from_points(...)	Creating a LineString object from a GeoDataFrame containing surface points at a given altitude and for a given formation
gemgis.vector.create_linestring_from_xyz_points(...)	Creating a LineString from an array or GeoDataFrame containing X, Y, and Z coordinates of points.
gemgis.vector.create_linestring_gdf(gdf)	Creating LineStrings from Points
gemgis.vector.create_linestrings_from_contourlines(...)	Creating LineStrings from PyVista Contour Lines and save them as list or GeoDataFrame
gemgis.vector.create_linestrings_from_xyz_points(...)	Creating LineStrings from a GeoDataFrame containing X, Y, and Z coordinates of vertices of multiple LineStrings
gemgis.vector.create_polygons_from_faces(...)	Extracting faces from PyVista PolyData as Shapely Polygons
gemgis.vector.unify_linestrings(linestrings)	Unifying adjacent LineStrings to form LineStrings with multiple vertices
gemgis.vector.unify_polygons(polygons[, ...])	Unifying adjacent triangular polygons to form larger objects

gemgis.vector.calculate_distance_linestrings

gemgis.vector.**calculate_distance_linestrings**(*ls1: shapely.geometry.linestring.LineString*, *ls2: shapely.geometry.linestring.LineString*) → float

Calculating the minimal distance between two LineStrings

Parameters

- **ls1** (*shapely.geometry.linestring.LineString*) – LineString 1, e.g. `ls1 = LineString([(0, 0), (10, 10), (20, 20)])`
- **ls2** (*shapely.geometry.linestring.LineString*) – LineString 2, e.g. `ls2 = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **distance** – Minimum distance between two Shapely LineStrings

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineStrings
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring1 = LineString([(0, 0), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(0, 10), (20, 30)])
>>> linestring2.wkt
'LINESTRING (0 10, 20 30)'
```

```
>>> # Calculating distance between LineStrings
>>> distance = gg.vector.calculate_distance_linestrings(ls1=linestring1,␣
↪ls2=linestring2)
>>> distance
7.0710678118654755
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.calculate_midpoint_linestring

gemgis.vector.**calculate_midpoint_linestring**(*linestring: shapely.geometry.linestring.LineString*) → shapely.geometry.point.Point

Calculating the midpoint of a LineString with two vertices

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – LineString consisting of two vertices from which the midpoint will be extracted, e.g. `linestring = LineString([(0, 0), (20, 20)])`

Returns **point** – Shapely Point representing the midpoint of the LineString

Return type shapely.geometry.point.Point

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Calculating the midpoint of a LineString
>>> midpoint = gg.vector.calculate_midpoint_linestring(linestring=linestring)
>>> midpoint.wkt
'POINT (10 -10)'
```

See also:

calculate_midpoints_linestrings Calculating the midpoints of LineStrings

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.calculate_midpoints_linestrings

gemgis.vector.**calculate_midpoints_linestrings**(*linestring_gdf: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) → List[shapely.geometry.point.Point]

Calculating the midpoints of LineStrings with two vertices each

Parameters **linestring_gdf** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame containing LineStrings or list of LineStrings of which the midpoints will be calculated

Returns **midpoint_list** – List of Shapely Points representing the midpoints of the provided LineStrings

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating list of LineStrings
>>> linestring_list = [linestring, linestring]
```

```
>>> # Calculating midpoints from LineStrings
>>> midpoints = gg.vector.calculate_midpoints_linestrings(linestring_gdf=linestring_
↪list)
>>> midpoints
[<shapely.geometry.point.Point at 0x231ea982880>,
 <shapely.geometry.point.Point at 0x231ea982700>]
```

```
>>> # Inspecting the first element of the list
>>> midpoints[0].wkt
'POINT (10 -10)'
```

```
>>> # Inspecting the second element of the list
>>> midpoints[1].wkt
'POINT (10 -10)'
```

See also:

calculate_midpoint_linestring Calculating the midpoint of one LineString

gemgis.vector.clip_by_bbox

gemgis.vector.**clip_by_bbox**(*gdf: geopandas.geodataframe.GeoDataFrame*, *bbox: List[Union[int, float]]*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*) → geopandas.geodataframe.GeoDataFrame

Clipping vector data contained in a GeoDataFrame to a provided bounding box/extent

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing vector data that will be clipped to a provided bounding box/extent
- **bbox** (*List[Union[float, int]]*) – Bounding box of minx, maxx, miny, maxy values to clip the GeoDataFrame, , e.g. `bbox=[0, 972, 0, 1069]`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing vector data clipped by a bounding box

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id  geometry
0       None    POINT (281.526 902.087)
1       None    POINT (925.867 618.577)
2       None    POINT (718.131 342.799)
3       None    POINT (331.011 255.684)
4       None    POINT (300.083 600.535)
```

```
>>> # Returning the length of the original gdf
>>> len(gdf)
50
```

```
>>> # Defining bounding box
>>> bbox = [0,972, 0, 1069]
```

```
>>> # Clipping data by bounding box
>>> gdf_clipped = gg.vector.clip_by_bbox(gdf=gdf, bbox=bbox)
>>> gdf_clipped
    geometry             X       Y
0       POINT (281.526 902.087) 281.53  902.09
1       POINT (925.867 618.577) 925.87  618.58
2       POINT (718.131 342.799) 718.13  342.80
3       POINT (331.011 255.684) 331.01  255.68
4       POINT (300.083 600.535) 300.08  600.54
```

```
>>> # Returning the length of the clipped gdf
>>> len(gdf_clipped)
25
```

See also:

clip_by_polygon Clipping vector data with a Shapely Polygon

gemgis.vector.clip_by_polygon

gemgis.vector.**clip_by_polygon**(*gdf: geopandas.geodataframe.GeoDataFrame*, *polygon: shapely.geometry.polygon.Polygon*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*) → geopandas.geodataframe.GeoDataFrame

Clipping vector data contained in a GeoDataFrame to a provided bounding box/extent

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing vector data that will be clipped to a provided bounding box/extent

- **polygon** (*polygon: shapely.geometry.polygon*) – Shapely Polygon defining the extent of the data, e.g. `polygon = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing vector data clipped by a bounding box

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id  geometry
0       None    POINT (281.526 902.087)
1       None    POINT (925.867 618.577)
2       None    POINT (718.131 342.799)
3       None    POINT (331.011 255.684)
4       None    POINT (300.083 600.535)
```

```
>>> # Returning the length of the original gdf
>>> len(gdf)
50
```

```
>>> # Creating Shapely Polygon
>>> from shapely.geometry import Polygon
>>> polygon = Polygon([(0,0),(972, 0), (972,1069), (0, 1069)])
>>> polygon.wkt
'POLYGON ((0 0, 972 0, 972 1069, 0 1069, 0 0))'
```

```
>>> # Clipping data by the polygon
>>> gdf_clipped = gg.vector.clip_by_polygon(gdf=gdf, polygon=polygon)
>>> gdf_clipped
    geometry            X       Y
```

(continues on next page)

(continued from previous page)

```
0       POINT (281.526 902.087) 281.53  902.09
1       POINT (925.867 618.577) 925.87  618.58
2       POINT (718.131 342.799) 718.13  342.80
3       POINT (331.011 255.684) 331.01  255.68
4       POINT (300.083 600.535) 300.08  600.54
```

```
>>> # Returning the length of the clipped gdf
>>> len(gdf_clipped)
25
```

See also:

clip_by_bbox Clipping vector data with a bbox

gemgis.vector.create_bbox

gemgis.vector.**create_bbox**(*extent: List[Union[int, float]]*) → shapely.geometry.polygon.Polygon

Creating a rectangular polygon from the provided bounding box values, with counter-clockwise order by default.

Parameters **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny, maxy values, e.g. extent=[0, 972, 0, 1069]

Returns **bbox** – Rectangular polygon based on extent

Return type shapely.geometry.polygon.Polygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Defining extent
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Creating bounding box
>>> bbox = gg.vector.create_bbox(extent=extent)
>>> bbox.wkt
'POLYGON ((972 0, 972 1069, 0 1069, 0 0, 972 0))'
```

gemgis.vector.create_buffer

gemgis.vector.**create_buffer**(*geom_object: shapely.geometry.base.BaseGeometry*, *distance: Union[float, int]*) → shapely.geometry.polygon.Polygon

Creating a buffer around a Shapely LineString or a Point

Parameters

- **geom_object** (*shapely.geometry.base.BaseGeometry*) – Shapely LineString or Point, e.g. geom_object=Point(0, 0)

- **distance** (*float, int*) – Distance of the buffer around the geometry object, e.g. `distance=10`

Returns **polygon** – Polygon representing the buffered area around a geometry object

Return type shapely.geometry.polygon.Polygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> point = Point(0,0)
>>> point.wkt
'POINT (0 0)'
```

```
>>> # Creating Buffer around Point
>>> point_buffered = gg.vector.create_buffer(geom_object=point, distance=10)
>>> point_buffered.wkt
'POLYGON ((100 0, 99.5184726672197 -9.801714032956051, 98.07852804032305 -19.
↪50903220161281, 95.69403357322089
-29.02846772544621, 92.38795325112869 -38.26834323650894, 88.19212643483553...))'
```

See also:

create_unified_buffer Creating a unified buffer around Shapely LineStrings or Points

gemgis.vector.create_unified_buffer

gemgis.vector.**create_unified_buffer**(*geom_object: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]], distance: Union[numpy.ndarray, List[Union[int, float]], float, int]*) → shapely.geometry.multipolygon.MultiPolygon

Creating a unified buffer around Shapely LineStrings or Points

Parameters

- **geom_object** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]]*) – GeoDataFrame or List of Shapely objects
- **distance** (*Union[np.ndarray, List[Union[float, int]], Union[float, int]]*) – Distance of the buffer around the geometry object, e.g. `distance=10`

Returns **polygon** – Polygon representing the buffered area around a geometry object

Return type shapely.geometry.multipolygon.MultiPolygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> point1 = Point(0,0)
>>> point1.wkt
'POINT (0 0)'
```

```
>>> # Creating Point
>>> point2 = Point(20,20)
>>> point2.wkt
'POINT (20 20)'
```

```
>>> # Creating list of points
>>> point_list = [point1, point2]
```

```
>>> # Creating unified buffer
>>> unified_buffer = gg.vector.create_unified_buffer(geom_object=point_list,␣
↪distance=10)
>>> unified_buffer
'MULTIPOLYGON (((10 0, 9.95184726672197 -0.980171403295605, 9.807852804032306 -1.
↪950903220161281, 9.56940335732209
-2.902846772544621, 9.23879532511287 -3.826834323650894,...)))'
```

See also:

create_buffer Creating a buffer around a Shapely LineString or Point

gemgis.vector.create_linestring_from_points

gemgis.vector.**create_linestring_from_points**(*gdf: geopandas.geodataframe.GeoDataFrame*, *formation: str*, *altitude: Union[int, float]*) → shapely.geometry.linestring.LineString

Creating a LineString object from a GeoDataFrame containing surface points at a given altitude and for a given formation

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points of intersections between topographic contours and layer boundaries
- **formation** (*str*) – Name of the formation, e.g. `formation='Layer1'`
- **altitude** (*Union[int, float]*) – Value of the altitude of the points, e.g. `altitude=100`

Returns **linestring** – LineString containing a LineString object

Return type shapely.geometry.linestring.LineString

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating points
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> import geopandas as gpd
>>> point1 = Point(0,0)
>>> point2 = Point (10,10)
```

```
>>> # Creating GeoDataFrame from points and adding additional information
>>> gdf = gpd.GeoDataFrame(geometry=[point1, point2])
>>> gdf['formation'] = 'Layer1'
>>> gdf['Z'] = 100
>>> gdf
    geometry            formation   Z
0   POINT (0.0 0.0)     Layer1      100
1   POINT (10.0 10.0)   Layer1      100
```

```
>>> # Creating LineString from Points
>>> linestring = gg.vector.create_linestring_from_points(gdf=gdf, formation='Layer1
↪', altitude=100)
>>> linestring.wkt
'LINESTRING (0 0, 10 10)'
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.create_linestring_from_xyz_points

gemgis.vector.**create_linestring_from_xyz_points**(*points: Union[numpy.ndarray, geopandas.geodataframe.GeoDataFrame]*, *nodata: Union[int, float] = 9999.0*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*, *drop_nan: bool = True*) → shapely.geometry.linestring.LineString

Create LineString from an array or GeoDataFrame containing X, Y, and Z coordinates of points.

Parameters

- **points** (*Union[np.ndarray, gpd.geodataframe.GeoDataFrame]*) – NumPy Array or GeoDataFrame containing XYZ points.
- **nodata** (*Union[int, float]*) – Nodata value to filter out points outside a designated area, e.g. `nodata=9999.0`, default is `9999.0`.
- **xcol** (*str*) – Name of the X column in the dataset, e.g. `xcol='X'`, default is `'X'`.
- **ycol** (*str*) – Name of the Y column in the dataset, e.g. `ycol='Y'`, default is `'Y'`.

- **zcol** (*str*) – Name of the Z column in the dataset, e.g. `zcol='Z'`, default is `'Z'`.
- **drop_nan** (*bool*) – Boolean argument to drop points that contain a `nan` value as Z value. Options include `True` and `False`, default is `True`.

Returns **line** – LineString Z constructed from provided point values

Return type shapely.geometry.linestring.LineString

New in version 1.0.x.

Changed in version 1.1: Adding argument *drop_nan* and code to drop coordinates that contain `nan` values as Z coordinates.

Example

```
>>> # Loading Libraries and creating points
>>> import gemgis as gg
>>> import numpy as np
>>> points = np.array([[3.23, 5.69, 2.03],[3.24, 5.68, 2.02],[3.25, 5.67, 1.97],[3.
↪26, 5.66, 1.95]])
```

```
>>> # Creating LineStrings from points
>>> linestring = gg.vector.create_linestring_from_xyz_points(points=points)
>>> linestring.wkt
'LINESTRING Z (3.23 5.69 2.03, 3.24 5.68 2.02, 3.25 5.67 1.97, 3.26 5.66 1.95)'
```

gemgis.vector.create_linestring_gdf

gemgis.vector.**create_linestring_gdf**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Creating LineStrings from Points

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points of intersections between topographic contours and layer boundaries

Returns **gdf_linestring** – GeoDataFrame containing LineStrings

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Points
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> import geopandas as gpd
>>> point1 = Point(0,0)
>>> point2 = Point (10,10)
```

```
>>> # Creating GeoDataFrame from points and adding additional information
>>> gdf = gpd.GeoDataFrame(geometry=[point1, point2])
>>> gdf['formation'] = 'Layer1'
>>> gdf['Z'] = 100
>>> gdf['id'] = 1
>>> gdf
    geometry            formation   Z   id
0   POINT (0.0 0.0)     Layer1      100 1
1   POINT (10.0 10.0)   Layer1      100 1
```

```
>>> # Creating LineString GeoDataFrame
>>> linestring_gdf = gg.vector.create_linestring_gdf(gdf=gdf)
>>> linestring_gdf
    index formation     Z       id      geometry
0   0     Layer1        100     1       LINESTRING (0.00000 0.00000, 10.00000 10.
↪00000)
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.create_linestrings_from_contours

gemgis.vector.**create_linestrings_from_contours**(*contours: pyvista.core.pointset.PolyData*, *return_gdf: bool = True*, *crs: Union[str, pyproj.crs.crs.CRS] = None*) → Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame]

Creating LineStrings from PyVista Contour Lines and save them as list or GeoDataFrame

Parameters

- **contours** (*pv.core.pointset.PolyData*) – PyVista PolyData dataset containing contour lines extracted from a mesh
- **return_gdf** (*bool*) – Variable to create GeoDataFrame of the created list of Shapely Objects. Options include: `True` or `False`, default set to `True`
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`

Returns **linestrings** – List of LineStrings or GeoDataFrame containing the contours that were converted

Return type Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> contours = pv.read('file.vtk')
>>> contours
Header
PolyData    Information
N Cells     36337
N Points    36178
X Bounds    3.233e+07, 3.250e+07
Y Bounds    5.704e+06, 5.798e+06
Z Bounds    -2.400e+03, 3.500e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -2.400e+03  3.500e+02
```

```
>>> # Extracting LineStrings from contours
>>> gdf = gg.vector.create_linestrings_from_contours(contours=contours)
>>> gdf
    geometry                                                Z
0   LINESTRING Z (32409587.930 5780538.824 -2350.0...   -2350.00
1   LINESTRING Z (32407304.336 5777048.086 -2050.0...   -2050.00
2   LINESTRING Z (32408748.977 5778005.047 -2200.0...   -2200.00
3   LINESTRING Z (32403693.547 5786613.994 -2400.0...   -2400.00
4   LINESTRING Z (32404738.664 5782672.480 -2350.0...   -2350.00
```

gemgis.vector.create_linestrings_from_xyz_points

gemgis.vector.**create_linestrings_from_xyz_points**(*gdf: geopandas.geodataframe.GeoDataFrame*, *groupby: str*, *nodata: Union[int, float] = 9999.0*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*, *dem: Union[numpy.ndarray, rasterio.io.DatasetReader] = None*, *extent: List[Union[int, float]] = None*, *return_gdf: bool = True*, *drop_nan: bool = True*) → Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame]

Creating LineStrings from a GeoDataFrame containing X, Y, and Z coordinates of vertices of multiple LineStrings

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing extracted X, Y, and Z coordinates of LineStrings
- **groupby** (*str*) – Name of a unique identifier the LineStrings can be separated from each other, e.g. `groupby='Object_ID'`
- **nodata** (*Union[int, float]*) – Nodata value to filter out points outside a designated area, e.g. `nodata=9999.0`, default is `9999.0`
- **xcol** (*str*) – Name of the X column in the dataset, e.g. `xcol='X'`, default is `'X'`

- **ycol** (*str*) – Name of the Y column in the dataset, e.g. `ycol='Y'`, default is `'Y'`
- **zcol** (*str*) – Name of the Z column in the dataset, e.g. `zcol='Z'`, default is `'Z'`
- **dem** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy ndarray or rasterio object containing the height values, default value is `None` in case geometries contain Z values
- **extent** (*List[Union[float, int]]*) – Values for minx, maxx, miny and maxy values to define the boundaries of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **return_gdf** (*bool*) – Variable to either return the data as GeoDataFrame or as list of LineStrings. Options include: `True` or `False`, default set to `True`
- **drop_nan** (*bool*) – Boolean argument to drop points that contain a `nan` value as Z value. Options include `True` and `False`, default is `True`

Returns **linestrings** – List of LineStrings or GeoDataFrame containing the LineStrings with Z component

Return type Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Changed in version 1.1: Removed manual dropping of additional columns. Now automatically drops unnecessary coloumns. Adding argument *drop_nan* and code to drop coordinates that contain `nan` values as Z coordinates.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
```

```
>>> # Creating LineStrings with Z component from gdf
>>> gdf_linestring = gg.vector.create_linestrings_from_xyz_points(gdf=gdf, groupby=
↪'ABS')
>>> gdf_linestring
```

gemgis.vector.create_polygons_from_faces

gemgis.vector.**create_polygons_from_faces**(*mesh: pyvista.core.pointset.PolyData*, *crs: Union[str, pyproj.crs.crs.CRS]*, *return_gdf: bool = True*) → Union[List[shapely.geometry.polygon.Polygon], geopandas.geodataframe.GeoDataFrame]

Extracting faces from PyVista PolyData as Shapely Polygons

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista PolyData dataset
- **crs** – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`

Returns **polygons** – Triangular Shapely Polygons representing the faces of the mesh

Return type Union[List[shapely.geometry.polygon.Polygon], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Importing Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> mesh = pv.read(filename='mesh.vtk')
>>> mesh
Header
PolyData    Information
N Cells     29273
N Points    40343
X Bounds    2.804e+05, 5.161e+05
Y Bounds    5.640e+06, 5.833e+06
Z Bounds    -8.067e+03, 1.457e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -8.067e+03  1.457e+02
```

```
>>> # Create polygons from mesh faces
>>> polygons = gg.vector.create_polygons_from_faces(mesh=mesh)
>>> polygons
    geometry
0   POLYGON Z ((297077.414 5677487.262 -838.496, 2...
1   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
2   POLYGON Z ((297437.539 5676992.094 -816.608, 2...
3   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
4   POLYGON Z ((295827.680 5680951.574 -825.328, 2...
```

gemgis.vector.unify_linestrings

gemgis.vector.**unify_linestrings**(*linestrings: Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame], crs: Union[str, pyproj.crs.crs.CRS] = None, return_gdf: bool = True*) → Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame]

Unifying adjacent LineStrings to form LineStrings with multiple vertices

Parameters

- **linestrings** (*Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]*) – LineStrings consisting of two vertices representing extracted contour lines
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`
- **return_gdf** (*bool*) – Variable to either return the data as GeoDataFrame or as list of LineStrings. Options include: `True` or `False`, default set to `True`

Returns **linestrings_merged** – Merged Shapely LineStrings

Return type Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> linestrings = gpd.read_file(filename='file.shp')
>>> linestrings
    geometry                                                Z
0   LINESTRING Z (32409587.930 5780538.824 -2350.0...   -2350.00
1   LINESTRING Z (32407304.336 5777048.086 -2050.0...   -2050.00
2   LINESTRING Z (32408748.977 5778005.047 -2200.0...   -2200.00
3   LINESTRING Z (32403693.547 5786613.994 -2400.0...   -2400.00
4   LINESTRING Z (32404738.664 5782672.480 -2350.0...   -2350.00
```

```
>>> # Merging linestrings
>>> polygons_linestrings = gg.vector.unify_linestrings(linestrings=linestrings)
>>> polygons_linestrings
    geometry
0   LINESTRING Z (32331825.641 5708789.973 -200.00...
1   LINESTRING Z (32334315.359 5723032.766 -250.00...
2   LINESTRING Z (32332516.312 5722028.768 -250.00...
3   LINESTRING Z (32332712.750 5721717.561 -250.00...
4   LINESTRING Z (32332516.312 5722028.768 -250.00...
```

gemgis.vector.unify_polygons

gemgis.vector.**unify_polygons**(*polygons: Union[List[shapely.geometry.polygon.Polygon], geopandas.geodataframe.GeoDataFrame]*, *crs: Union[str, pyproj.crs.crs.CRS] = None*, *return_gdf: bool = True*) → Union[List[shapely.geometry.polygon.Polygon], geopandas.geodataframe.GeoDataFrame]

Unifying adjacent triangular polygons to form larger objects

Parameters

- **polygons** (*Union[List[shapely.geometry.polygon.Polygon], gpd.geodataframe.GeoDataFrame]*) – Triangular Shapely Polygons representing the faces of the mesh
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`
- **return_gdf** (*bool*) – Variable to either return the data as GeoDataFrame or as list of LineStrings. Options include: `True` or `False`, default set to `True`

Returns **polygons_merged** – Merged Shapely Polygons

Return type Union[List[shapely.geometry.polygon.Polygon], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> polygons = gpd.read_file(filename='file.shp')
>>> polygons
    geometry
0   POLYGON Z ((297077.414 5677487.262 -838.496, 2...
1   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
2   POLYGON Z ((297437.539 5676992.094 -816.608, 2...
3   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
4   POLYGON Z ((295827.680 5680951.574 -825.328, 2...
```

```
>>> # Merging polygons
>>> polygons_merged = gg.vector.unify_polygons(polygons=polygons)
>>> polygons_merged
    geometry
0   POLYGON Z ((396733.222 5714544.109 -186.252, 3...
1   POLYGON Z ((390252.635 5712409.037 -543.142, 3...
2   POLYGON Z ((391444.965 5710989.453 -516.000, 3...
3   POLYGON Z ((388410.007 5710903.900 -85.654, 38...
4   POLYGON Z ((384393.963 5714293.104 -614.106, 3...
```

8.1.10 Special Methods

The following methods are special methods used in GemGIS

gemgis.vector.set_dtype(gdf[, dip, azimuth, ...])	Checking and setting the dtypes of the input data GeoDataFrame
gemgis.vector.sort_by_stratigraphy(gdf, ...)	Sorting a GeoDataFrame by a provided list of Stratigraphic Units
gemgis.vector.subtract_geom_objects(...)	Subtracting Shapely geometry objects from each other and returning the left over object
gemgis.vector.create_hexagon(center, radius)	Function to create one hexagon
gemgis.vector.create_hexagon_grid(gdf, radius)	Function to create a grid of hexagons based on a GeoDataFrame containing Polygons and a radius provided for the single hexagons
gemgis.vector.create_voronoi_polygons(gdf)	Function to create Voronoi Polygons from Point GeoDataFrame using the SciPy Spatial Voronoi class (https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html#scipy.spatial.Voronoi)

gemgis.vector.set_dtype

gemgis.vector.**set_dtype**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dip: str = 'dip'*, *azimuth: str = 'azimuth'*, *formation: str = 'formation'*, *polarity: str = 'polarity'*, *x: str = 'X'*, *y: str = 'Y'*, *z: str = 'Z'*) → geopandas.geodataframe.GeoDataFrame

Checking and setting the dtypes of the input data GeoDataFrame

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the input vector data with uncorrected dtypes
- **dip** (*str*) – Name of the column containing the dip data, e.g `dip='dip'`
- **azimuth** (*str*) – Name of the column containing the azimuth data, e.g `azimuth='azimuth'`
- **formation** (*str*) – Name of the column containing the formation data, e.g `formation='formation'`
- **polarity** (*str*) – Name of the column containing the polarity data, e.g `polarity='polarity'`
- **x** (*str*) – Name of the column containing the x coordinates, e.g `x='X'`
- **y** (*str*) – Name of the column containing the y coordinates, e.g `y='Y'`
- **z** (*str*) – Name of the column containing the z coordinates, e.g `z='Z'`

Returns **gdf** – GeoDataFrame containing the input vector data with corrected dtypes

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
```

```
>>> # Setting the data types
>>> gdf_dtypes = gg.vector.set_dtype(gdf=gdf)
```

gemgis.vector.sort_by_stratigraphy

gemgis.vector.**sort_by_stratigraphy**(*gdf: geopandas.geodataframe.GeoDataFrame*, *stratigraphy: List[str]*, *formation_column: str = 'formation'*) → geopandas.geodataframe.GeoDataFrame

Sorting a GeoDataFrame by a provided list of Stratigraphic Units

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the unsorted input polygons

- **stratigraphy** (*List[str]*) – List containing the stratigraphic units sorted by age, e.g. `stratigraphy=['Layer1' , 'Layer2']`
- **formation_column** (*str*) – Name of the formation column, default is formation, e.g. `formation_colum='formation'`

Returns **gdf_sorted** – GeoDataFrame containing the sorted input polygons

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> import geopandas as gpd
>>> polygon1 = Polygon([(0, 0), (1, 1), (1, 0)])
>>> polygon1.wkt
'POLYGON ((0 0, 1 1, 1 0, 0 0))'
```

```
>>> # Creating second polygon
>>> polygon2 = Polygon([(0, 0), (2, 2), (2, 0)])
>>> polygon2.wkt
'POLYGON ((0 0, 2 2, 2 0, 0 0))'
```

```
>>> # Creating GeoDataFrame from polygons
>>> gdf = gpd.GeoDataFrame(geometry=[polygon1, polygon2])
>>> gdf['formation'] = ['Layer2', 'Layer1']
>>> gdf
    geometry                                                formation
0   POLYGON ((0.00000 0.00000, 1.00000 1.00000, 1....   Layer2
1   POLYGON ((10.00000 0.00000, 20.00000 0.00000, ...   Layer1
```

```
>>> # Creating stratigraphy list
>>> stratigraphy = ['Layer1' , 'Layer2']
```

```
>>> # Sorting GeoDataFrame by stratigraphy
>>> gdf_sorted = gg.vector.sort_by_stratigraphy(gdf=gdf, stratigraphy=stratigraphy)
>>> gdf_sorted
    geometry                                                formation
0   POLYGON ((10.00000 0.00000, 20.00000 0.00000, ...   Layer1
1   POLYGON ((0.00000 0.00000, 1.00000 1.00000, 1....   Layer2
```

gemgis.vector.subtract_geom_objects

gemgis.vector.**subtract_geom_objects**(*geom_object1: shapely.geometry.base.BaseGeometry*, *geom_object2: shapely.geometry.base.BaseGeometry*) → shapely.geometry.base.BaseGeometry

Subtracting Shapely geometry objects from each other and returning the left over object

Parameters

- **geom_object1** (*shapely.geometry.base.BaseGeometry*) – Shapely object from which other object will be subtracted, e.g. `geom_object1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`
- **geom_object2** (*shapely.geometry.base.BaseGeometry*) – Shapely object which will be subtracted from other object e.g. `geom_object2 = Polygon([[5, 0], [15, 0], [15, 10], [5, 10], [5, 0]])`

Returns **result** – Shapely object from which the second object was subtracted

Return type shapely.geometry.base.BaseGeometry

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[5, 0], [15, 0], [15, 10], [5, 10], [5, 0]])
>>> polygon2.wkt
'POLYGON ((5 0, 15 0, 15 10, 5 10, 5 0))'
```

```
>>> # Subtracting geometries from each other
>>> difference = gg.vector.subtract_geom_objects(geom_object1=polygon1, geom_
↪object2=polygon2)
>>> difference.wkt
'POLYGON ((5 0, 0 0, 0 10, 5 10, 5 0))'
```

gemgis.vector.create_hexagon

gemgis.vector.**create_hexagon**(*center: shapely.geometry.point.Point*, *radius: Union[int, float]*)

Function to create one hexagon

Parameters

- **center** (*shapely.geometry.Point*) – Shapely Point representing the center of the hexagon
- **radius** (*int, float*) – Radius of the hexagon

Returns **geometry.Polygon(hex_coords)** – Shapley Polygon in the shape of a hexagon

Return type shapely.geometry.Polygon

New in version 1.0.x.

Changed in version 1.1.3: Optimized creation of hexagon

See also:

create_hexagon_grid Creating a hexagon grid

gemgis.vector.create_hexagon_grid

gemgis.vector.**create_hexagon_grid**(*gdf: geopandas.geodataframe.GeoDataFrame*, *radius: Union[int, float]*, *crop_gdf: bool = True*)

Function to create a grid of hexagons based on a GeoDataFrame containing Polygons and a radius provided for the single hexagons

Parameters

- **gdf** (*gpd.GeoDataFrame*) – GeoDataFrame containing the polygons for which a hexagon grid is created
- **radius** (*int, float*) – Radius of the hexagon
- **crop_gdf** (*bool*) – Boolean to define if the resulting GeoDataFrame should be cropped to the extend of the provided GeoDataFrame Options include: `True` or `False`, default set to `True`

Returns **hex_gdf** – GeoDataFrame containing the hexagon grid

Return type gpd.GeoDataFrame

New in version 1.0.x.

Changed in version 1.1.3: Optimized creation of hexagon

See also:

create_hexagon Creating one hexagon based on a given center and radius

gemgis.vector.create_voronoi_polygons

gemgis.vector.**create_voronoi_polygons**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Function to create Voronoi Polygons from Point GeoDataFrame using the SciPy Spatial Voronoi class (https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html#scipy.spatial.Voronoi)

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the Shapely Points

Returns **gdf_polygons** – GeoDataFrame containing the valid Voronoi Polygons

Return type gpd.geodataframe.GeoDataFrame

New in version 1.1.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file('file.shp')
>>> gdf_polygons = gg.vector.create_voronoi_polygons(gdf=gdf)
```

8.2 Raster

The following sections provide an overview of the methods implemented in the GemGIS Raster module.

8.2.1 Raster Calculations

The following methods are used to perform calculations on rasters

gemgis.raster.calculate_aspect(raster[, ...])	Calculating the aspect based on a digital elevation model/raster
gemgis.raster.calculate_difference(raster1, ...)	Calculating the difference between two rasters
gemgis.raster.calculate_hillshades(raster[, ...])	Calculating Hillshades based on digital elevation model/raster
gemgis.raster.calculate_slope(raster[, ...])	Calculating the slope based on digital elevation model/raster

gemgis.raster.calculate_aspect

gemgis.raster.**calculate_aspect**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *band_no: int = 1*) → numpy.ndarray

Calculating the aspect based on a digital elevation model/raster

Parameters

- **raster** (*np.ndarray, rasterio.io.DatasetReader*) – NumPy array or rasterio object containing the elevation data
- **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny and maxy coordinates representing the raster extent if raster is passed as array, e.g. `extent=[0, 972, 0, 1069]`
- **band_no** (*int*) – Band number of the raster to be used for calculating the hillshades, e.g. band_no=1, default is 1

Returns **aspect** – NumPy array containing the aspect values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Calculating the aspect of a raster
>>> aspect = gg.raster.calculate_aspect(raster=raster)
>>> aspect
array([[246.37328, 245.80156, 245.04022, ..., 269.87958, 270.11377,
270.32904],....], dtype=float32)
```

See also:

calculate_hillshades Calculating the hillshades of a raster

calculate_slope Calculating the slope of a raster

calculate_difference Calculating the difference between two rasters

gemgis.raster.calculate_difference

gemgis.raster.**calculate_difference**(*raster1: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *raster2: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *flip_array: bool = False*) → numpy.ndarray

Calculating the difference between two rasters

Parameters

- **raster1** (*np.ndarray*) – First array
- **raster2** (*np.ndarray*) – Second array
- **flip_array** (*bool*) – Variable to flip the array. Options include: `True` or `False`, default set to `False`

Returns **array_diff** – Array containing the difference between array1 and array2

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and Files
>>> import gemgis as gg
>>> import rasterio
>>> raster1 = rasterio.open(fp='raster1.tif')
>>> raster2 = rasterio.open(fp='raster2.tif')
```

```
>>> # Calculate difference between two rasters
>>> difference = gg.raster.calculate_difference(raster1=raster1, raster2=raster2)
>>> difference
array([[-10., -10., -10., ..., -10., -10., -10.],.....], dtype=float32)
```

See also:

calculate_hillshades Calculating the hillshades of a raster

calculate_slope Calculating the slope of a raster

calculate_aspect Calculating the aspect of a raster

gemgis.raster.calculate_hillshades

gemgis.raster.**calculate_hillshades**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *azdeg: Union[int, float] = 225*, *altdeg: Union[int, float] = 45*, *band_no: int = 1*) → numpy.ndarray

Calculating Hillshades based on digital elevation model/raster

Parameters

- **raster** (*np.ndarray, rasterio.io.DatasetReader*) – NumPy array or rasterio object containing the elevation data
- **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny and maxy points representing the extent of the raster if raster is passed as array, e.g. `extent=[0, 972, 0, 1069]`
- **azdeg** (*Union[int, float]*) – Azimuth value for the light source direction, e.g. `azdeg=225`, default is `225` degrees
- **altdeg** (*Union[int, float]*) – Altitude value for the light source, e.g. `altdeg=45`, default is `45` degrees
- **band_no** (*int*) – Band number of the raster to be used for calculating the hillshades, e.g. `band_no=1`, default is `1`

Returns **hillshades** – NumPy array containing the hillshade color values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Calculating hillshades from raster
>>> hillshades = gg.raster.calculate_hillshades(raster=raster)
>>> hillshades
array([[250.04817, 250.21147, 250.38988, ..., 235.01764, 235.0847 ,
235.0842 ], ....], dtype=float32)
```

See also:

calculate_slope Calculating the slope of a raster

calculate_aspect Calculating the aspect of a raster

calculate_difference Calculating the difference between two rasters

gemgis.raster.calculate_slope

gemgis.raster.**calculate_slope**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *band_no: int = 1*) → numpy.ndarray

Calculating the slope based on digital elevation model/raster

Parameters

- **raster** (*np.ndarray, rasterio.io.DatasetReader*) – NumPy array or rasterio object containing the elevation data
- **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny and maxy coordinates representing the raster extent if raster is passed as array, e.g. `extent=[0, 972, 0, 1069]`
- **band_no** (*int*) – Band number of the raster to be used for calculating the hillshades, e.g. band_no=1, default is 1

Returns **slope** – NumPy array containing the slope values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Calculating the slope of a raster
>>> slope = gg.raster.calculate_slope(raster=raster)
>>> slope
array([[37.092472, 36.95191 , 36.649662, ..., 21.988844, 22.367924,
22.584248],....], dtype=float32)
```

See also:

calculate_hillshades Calculating the hillshades of a raster

calculate_aspect Calculating the aspect of a raster

calculate_difference Calculating the difference between two rasters

8.2.2 Sampling from a Raster

The following methods are used to sample values from a raster

gemgis.raster.sample_from_array(array, ...)	Sampling the value of a np.ndarray at a given point and given the arrays true extent
gemgis.raster.sample_from_rasterio(raster, ...)	Sampling the value of a rasterio object at a given point within the extent of the raster
gemgis.raster.sample_interfaces(raster[, ...])	Sampling interfaces from a raster
gemgis.raster.sample_orientations(raster[, ...])	Sampling orientations from a raster
gemgis.raster.sample_randomly(raster[, n, ...])	Sampling randomly from a raster (array or rasterio object) using sample_from_array or sample_from_rasterio and a randomly drawn point within the array/raster extent

gemgis.raster.sample_from_array

gemgis.raster.**sample_from_array**(*array: numpy.ndarray*, *extent: Sequence[float]*, *point_x: Union[float, int, list, numpy.ndarray]*, *point_y: Union[float, int, list, numpy.ndarray]*) → Union[numpy.ndarray, float]

Sampling the value of a np.ndarray at a given point and given the arrays true extent

Parameters

- **array** (*np.ndarray*) – Array containing the raster values
- **extent** (*list*) – List containing the values for the extent of the array (minx,maxx,miny,maxy), e.g. `extent=[0, 972, 0, 1069]`
- **point_x** (*Union[float, int, list, np.ndarray]*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`
- **point_y** (*Union[float, int, list, np.ndarray]*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`

Returns **sample** – Value/s of the raster at the provided position/s

Return type Union[np.ndarray, float]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Getting array data
>>> array = raster.read()
```

```
>>> # Sampling values from an array
>>> value = gg.raster.sample_from_array(array=array, extent=[0, 972, 0, 1069],␣
↪point_x=500, point_y=500)
```

(continues on next page)

(continued from previous page)

```
>>> value
562.0227
```

See also:

sample_from_rasterio Sample values from rasterio object

sample_randomly Sample randomly from rasterio object or NumPy array

sample_orientations Sample orientations from raster

sample_interfaces Sample interfaces from raster

gemgis.raster.sample_from_rasterio

gemgis.raster.**sample_from_rasterio**(*raster: rasterio.io.DatasetReader*, *point_x: Union[float, int, list, numpy.ndarray]*, *point_y: Union[float, int, list, numpy.ndarray]*, *sample_outside_extent: bool = True*, *sample_all_bands: bool = False*) → Union[list, float]

Sampling the value of a rasterio object at a given point within the extent of the raster

Parameters

- **raster** (*rasterio.io.DatasetReader*) – Rasterio Object containing the height information
- **point_x** (*list, np.ndarray, float, int*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`
- **point_y** (*list, np.ndarray, float, int*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`
- **sample_outside_extent** (*bool*) – Allow sampling outside the extent of the rasterio object. Options include: `True` or `False`, default set to `True`
- **sample_all_bands** (*bool*) – Allow sampling from all bands returning Options include: `True` or `False`, default set to `False`

Returns **sample** – Value/s of the raster at the provided position/s

Return type list, float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling values from a rasterio object
>>> value = gg.raster.sample_from_rasterio(raster=raster, point_x=500, point_y=500)
>>> value
561.646728515625
```

See also:

sample_from_array Sample values from NumPy array

sample_randomly Sample randomly from rasterio object or NumPy array

sample_orientations Sample orientations from raster

sample_interfaces Sample interfaces from raster

gemgis.raster.sample_interfaces

gemgis.raster.**sample_interfaces**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *point_x: Union[float, int, list, numpy.ndarray] = None*, *point_y: Union[float, int, list, numpy.ndarray] = None*, *random_samples: int = None*, *formation: str = None*, *seed: int = None*, *sample_outside_extent: bool = False*, *crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*) → geopandas.geodataframe.GeoDataFrame

Sampling interfaces from a raster

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster or arrays from which points are being sampled
- **extent** (*List[Union[int, float]]*) – List containing the extent of the raster (minx, maxx, miny, maxy), e.g. `extent=[0, 972, 0, 1069]`
- **point_x** (*Union[float, int, list, np.ndarray]*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`, default is `None`
- **point_y** (*Union[float, int, list, np.ndarray]*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`, default is `None`
- **random_samples** (*int*) – Number of random samples to be drawn, e.g. `random_samples=10`, default is `None`
- **formation** (*str*) – Name of the formation the raster belongs to, e.g. `formation='Layer1'`, default is `None`
- **seed** (*int*) – Integer to set a seed for the drawing of random values, e.g. `seed=1`, default is `None`
- **sample_outside_extent** (*bool*) – Allow sampling outside the extent of the rasterio object. Options include: `True` or `False`, default is `False`
- **crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Coordinate reference system to be passed to the GeoDataFrame upon creation, e.g. `crs='EPSG:4647`

Returns **gdf** – GeoDataFrame containing the sampled interfaces

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling interfaces from an array or rasterio object
>>> gdf = gg.raster.sample_interfaces(raster=raster, point_x=500, point_y=500)
>>> gdf
    X       Y       Z       geometry
0   500.00  500.00  561.65  POINT (500.000 500.000)
```

See also:

sample_from_array Sample values from NumPy array

sample_from_rasterio Sample values from rasterio object

sample_randomly Sample randomly from rasterio object or NumPy array

sample_orientations Sample orientations from raster

gemgis.raster.sample_orientations

gemgis.raster.**sample_orientations**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *point_x: Union[float, int, list, numpy.ndarray] = None*, *point_y: Union[float, int, list, numpy.ndarray] = None*, *random_samples: int = None*, *formation: str = None*, *seed: int = None*, *sample_outside_extent: bool = False*, *crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*) → geopandas.geodataframe.GeoDataFrame

Sampling orientations from a raster

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster or arrays from which points are being sampled
- **extent** (*List[Union[int, float]]*) – List containing the extent of the raster (minx, maxx, miny, maxy), e.g. `extent=[0, 972, 0, 1069]`
- **point_x** (*Union[float, int, list, np.ndarray]*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`, default is `None`
- **point_y** (*Union[float, int, list, np.ndarray]*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`, default is `None`
- **random_samples** (*int*) – Number of random samples to be drawn, e.g. `random_samples=10`, default is `None`
- **formation** (*str*) – Name of the formation the raster belongs to, e.g. `formation='Layer1'`, default is `None`
- **seed** (*int*) – Integer to set a seed for the drawing of random values, e.g. `seed=1`, default is `None`

- **sample_outside_extent** (*bool*) – Allow sampling outside the extent of the rasterio object. Options include: `True` or `False`, default is `False`
- **crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Coordinate reference system to be passed to the GeoDataFrame upon creation, e.g. `crs='EPSG:4647`

Returns **gdf** – GeoDataFrame containing the sampled interfaces

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling orientations from an array or rasterio object
>>> gdf = gg.raster.sample_orientations(raster=raster, point_x=500, point_y=500)
>>> gdf
    X       Y       Z       geometry                dip     azimuth polarity
0   500.00  500.00  561.65  POINT (500.000 500.000) 19.26   145.55  1
```

See also:

sample_from_array Sample values from NumPy array

sample_from_rasterio Sample values from rasterio object

sample_randomly Sample randomly from rasterio object or NumPy array

sample_interfaces Sample interfaces from raster

gemgis.raster.sample_randomly

gemgis.raster.**sample_randomly**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *n: int = 1*, *extent: Optional[Sequence[float]] = None*, *seed: int = None*) → tuple

Sampling randomly from a raster (array or rasterio object) using sample_from_array or sample_from_rasterio and a randomly drawn point within the array/raster extent

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy Array or rasterio object containing the raster values
- **n** (*int*) – Number of samples to be drawn, e.g. `n=10`, default `1`
- **extent** (*Optional[Sequence[float]]*) – List containing the values for the extent of the array (minx,maxx,miny,maxy), default is `None`, e.g. `extent=[0, 972, 0, 1069]`
- **seed** (*int*) – Seed for the random variable for reproducibility, e.g. `seed=1`, default is `None`

Returns **sample** – Float of sampled raster value and list containing the x- and y-points of the point where sample was drawn

Return type tuple

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling randomly from an array or rasterio object
>>> value = gg.raster.sample_randomly(raster=raster, n=1)
>>> value
(617.0579833984375, [529.5110732824818, 717.7358438674542])
```

See also:

sample_from_array Sample values from NumPy array

sample_from_rasterio Sample values from rasterio object

sample_orientations Sample orientations from raster

sample_interfaces Sample interfaces from raster

8.2.3 Reading different raster formats

The following methods are used to read different raster formats into Python

gemgis.raster.read_asc(path)	Function to read GoCAD .asc files
gemgis.raster.read_msh(path)	Function to read Leapfrog .msh files - https://help.leapfrog3d.com/Geo/4.3/en-GB/Content/meshes/meshes.htm
gemgis.raster.read_ts(path)	Function to read GoCAD .ts files
gemgis.raster.read_zmap(path)	Reading Petrel ZMAP Files

gemgis.raster.read_asc

gemgis.raster.**read_asc**(*path: Union[str, pathlib.Path]*) → dict

Function to read GoCAD .asc files

Parameters **path** (*Union[str, Path]*) – Path to asc file, e.g. path='raster.asc'

Returns **data** – Dict containing the array data, the extent, resolution and nodata_val of the raster

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and Files
>>> import gemgis as gg
>>> data = gg.raster.read_asc('raster.asc')
```

```
>>> # Inspecting the content of the dict, here we only see the nodata_vals for now
>>> data['Data']
array([[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
...,
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.]])
```

```
>>> data['Extent']
[-42250, 306000, 279000, 867000]
```

```
>>> data['Resolution']
250
```

```
>>> data['Nodata_val']
-99999
```

See also:

read_ts Reading a GoCAD TSurface File

read_msh Reading a Leapfrog Mesh File

read_zmap Reading Petrel ZMAP Files

gemgis.raster.read_msh

gemgis.raster.**read_msh**(*path: Union[str, pathlib.Path]*) → Dict[str, numpy.ndarray]

Function to read Leapfrog .msh files - https://help.leapfrog3d.com/Geo/4.3/en-GB/Content/meshes/meshes.htm

Parameters **path** (*Union[str, Path]*) – Path to msh file, e.g. `path='mesh.msh'`

Returns **data** – Dict containing the mesh data

Return type Dict[str, np.ndarray]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> data = gg.raster.read_msh('mesh.msh')
>>> data
{'Tri': array([[    0,     1,     2],
[    0,     3,     1],
[    4,     3,     0],
...,
[53677, 53672, 53680],
[53679, 53677, 53680],
[53673, 53672, 53677]]),
'Location': array([[ 1.44625109e+06,  5.24854344e+06, -1.12743862e+02],
[ 1.44624766e+06,  5.24854640e+06, -1.15102216e+02],
[ 1.44624808e+06,  5.24854657e+06, -1.15080548e+02],
...,
[ 1.44831008e+06,  5.24896679e+06, -1.24755449e+02],
[ 1.44830385e+06,  5.24896985e+06, -1.33694397e+02],
[ 1.44829874e+06,  5.24897215e+06, -1.42506587e+02]])}
```

See also:

read_ts Reading a GoCAD TSurface File

read_asc Reading ESRI ASC files

read_zmap Reading Petrel ZMAP Files

gemgis.raster.read_ts

gemgis.raster.**read_ts**(*path: Union[str, pathlib.Path]*) → Tuple[list, list]

Function to read GoCAD .ts files

Parameters **path** (*Union[str, Path]*) – Path to ts file, e.g. `path='mesh.ts'`

Returns

- **vertices** (*list*) – Pandas DataFrames containing the vertex data
- **faces** (*list*) – NumPy arrays containing the faces data

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> vertices, faces = gg.raster.read_ts('mesh.ts')
```

```
>>> # Inspecting the vertices
>>> vertices
    id  X           Y           Z
0   0   297077.41   5677487.26  -838.50
1   1   297437.54   5676992.09  -816.61
```

```
>>> # Inspecting the faces
>>> faces
array([[    0,     1,     2],
[    3,     2,     4],
[    1,     5,     6],...,
[40335, 40338, 40336],
[40339, 40340, 40341],
[40341, 40342, 40339]])
```

See also:

read_msh Reading a Leapfrog Mesh File

read_asc Reading ESRI ASC files

read_zmap Reading Petrel ZMAP Files

gemgis.raster.read_zmap

gemgis.raster.**read_zmap**(*path: Union[str, pathlib.Path]*) → dict

Reading Petrel ZMAP Files

Parameters **path** (*Union[str, Path]*) – Path to dat file, e.g. path='raster.dat'

Returns **data** – Dict containing the array data, the extent, array dimension, resolution and nodata_val of the raster

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> data = gg.raster.read_zmap(path='file.dat')
```

```
>>> # Inspecting the content of the dict, here we only see the nodata_vals for now
>>> data
{'Data': array([[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan],
...,
[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan]]),
 'Extent': [-42250.0, 278750.0, 306000.0, 866750.0],
 'Resolution': [250.0, 250.0],
 'Nodata_val': 0.1000000E+31,
 'Dimensions': (2244, 1285),
 'CRS': 'Amersfoort * EPSG-Nld / RD New [28992,1672]',
 'Creation_date': '21/10/2019',
 'Creation_time': '16',
 'File_name': 'TOP_DINANTIAN_TVD_final'}
```

See also:

read_ts Reading a GoCAD TSurface File

read_msh Reading a Leapfrog Mesh File

read_asc Reading ESRI ASC files

8.2.4 Miscellaneous raster data methods

The following methods are further raster data methods used in GemGIS

gemgis.raster.clip_by_bbox(raster, bbox[, ...])	Clipping a rasterio raster or np.ndarray by a given extent
gemgis.raster.clip_by_polygon(raster, polygon)	Clipping/masking a rasterio raster or np.ndarray by a given shapely Polygon
gemgis.raster.create_filepaths(dirpath, ...)	Retrieving the file paths of the tiles to load and to process them later
gemgis.raster.extract_contour_lines_from_raster(ra...	Extracting contour lines from raster with a provided interval.
gemgis.raster.merge_tiles(src_files[, ...])	Merging downloaded tiles to mosaic
gemgis.raster.reproject_raster(path_in, ...)	Reprojecting a raster into different CRS
gemgis.raster.resize_by_array(raster, array)	Rescaling raster to the size of another raster
gemgis.raster.resize_raster(raster, width, ...)	Resizing raster to given dimensions
gemgis.raster.save_as_tiff(raster, path, ...)	Saving a np.array as tif file

gemgis.raster.clip_by_bbox

gemgis.raster.**clip_by_bbox**(*raster: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *bbox: List[Union[int, float]]*, *raster_extent: List[Union[int, float]] = None*, *save_clipped_raster: bool = False*, *path: str = 'raster_clipped.tif'*, *overwrite_file: bool = False*, *create_directory: bool = False*) → numpy.ndarray

Clipping a rasterio raster or np.ndarray by a given extent

Parameters

- **raster** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Array or Rasterio object to be clipped
- **bbox** (*List[Union[int, float]]*) – Bounding box of minx, maxx, miny, maxy values to clip the raster, e.g. `bbox=[0, 972, 0, 1069]`
- **raster_extent** (*List[Union[int, float]]*) – List of float values defining the extent of the raster, default None, e.g. `raster_extent=[0, 972, 0, 1069]`
- **save_clipped_raster** (*bool*) – Variable to save the raster after clipping. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path where the raster is saved, e.g. `path='raster_clipped.tif'`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **raster_clipped** – Clipped array after clipping

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Creating bounding box and defining raster extent
>>> bbox = [250, 500, 250, 500]
>>> raster_extent = [0, 972, 0, 1069]
```

```
>>> # Clipping raster by bounding box
>>> raster_clipped = gg.raster.clip_by_bbox(raster=raster, bbox=bbox, raster_
↪extent=raster_extent)
>>> raster_clipped.shape
(65, 65)
```

See also:

clip_by_polygon Clipping raster by a Shapely Polygon

gemgis.raster.clip_by_polygon

gemgis.raster.**clip_by_polygon**(*raster: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *polygon: shapely.geometry.polygon.Polygon*, *raster_extent: List[Union[int, float]] = None*, *save_clipped_raster: bool = False*, *path: str = 'raster_clipped.tif'*, *overwrite_file: bool = False*, *create_directory: bool = False*) → numpy.ndarray

Clipping/masking a rasterio raster or np.ndarray by a given shapely Polygon

Parameters

- **raster** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Array or Rasterio object to be clipped
- **polygon** (*shapely.geometry.polygon.Polygon*) – Shapely polygon defining the extent of the data, e.g. `polygon = Polygon([(0, 0), (1, 1), (1, 0)])`
- **raster_extent** (*List[Union[int, float]]*) – List of float values defining the extent of the raster, default None, e.g. `raster_extent=[0, 972, 0, 1069]`
- **save_clipped_raster** (*bool*) – Variable to save the raster after clipping, default False. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path where the raster is saved, e.g. `path='raster_clipped.tif`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`

- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **raster_clipped** – Clipped array after clipping

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> from shapely.geometry import Polygon
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Creating Shapely Polygon and defining raster extent
>>> polygon = Polygon([(250, 250), (500, 250), (500, 500), (250, 500)])
>>> raster_extent = [0, 972, 0, 1069]
```

```
>>> # Clipping the raster by a Shapely Polygon
>>> raster_clipped = gg.raster.clip_by_polygon(raster=raster, polygon=polygon,␣
↪raster_extent=raster_extent)
>>> raster_clipped.shape
(65, 65)
```

See also:

clip_by_bbox Clipping raster by a Bounding Box

gemgis.raster.create_filepaths

gemgis.raster.**create_filepaths**(*dirpath: str*, *search_criteria: str*, *create_directory: bool = False*) → List[str]

Retrieving the file paths of the tiles to load and to process them later

Parameters

- **dirpath** (*str*) – Path to the folder where tiles are stored, e.g. `dirpath='Documents/images/'`
- **search_criteria** (*str*) – Name of the files including file ending, use * for autocompletion by Python, e.g. `search_criteria='tile*.tif'`
- **create_directory** (*bool*) – Variable to create a new directory if directory does not exist Options include: `True` or `False`, default set to `False`

Returns **filepaths** – List of file paths

Return type List[str]

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Defining filepath
>>> filepath = 'Documents/images/'
```

```
>>> # Creating list of filepaths based on search criteria
>>> filepaths = gg.raster.create_filepaths(dirpath=filepath, search_criteria='tile*.
↪tif')
>>> filepaths
['Documents/images//tile_292000_294000_5626000_5628000.tif',
'Documents/images//tile_292000_294000_5628000_5630000.tif',
'Documents/images//tile_292000_294000_5630000_5632000.tif',
'Documents/images//tile_294000_296000_5626000_5628000.tif']
```

gemgis.raster.extract_contour_lines_from_raster

gemgis.raster.**extract_contour_lines_from_raster**(*raster: Union[rasterio.io.DatasetReader, numpy.ndarray, str]*, *interval: int*, *extent: Union[Sequence[float], None, Sequence[int]] = None*, *target_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting contour lines from raster with a provided interval.

Parameters

- **raster** (*Union[rasterio.io.DatasetReader, np.ndarray, str]*) – Raster from which contour lines are extracted
- **extent** (*Optional[Sequence[float, int]]*) – If raster given as array: values (minx, maxx, miny, maxy) to define raster extent, e.g. `extent =[0, 972, 0, 1069]`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – If raster given as array: name of the CRS is required to project values to coordinates of GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **interval** (*int*) – Given interval for the extracted contour lines, e.g. `interval=50`

Returns **gdf_lines** – GeoDataFrame containing the extracted contour lines as LineStrings

Return type gpd.GeoDataFrame

New in version 1.0.x.

gemgis.raster.merge_tiles

gemgis.raster.**merge_tiles**(*src_files: List[rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *res: int = None*, *nodata: Union[float, int] = None*, *precision: int = None*, *indices: int = None*, *method: str = 'first'*) → Tuple[numpy.ndarray, affine.Affine]

Merging downloaded tiles to mosaic

Parameters

- **src_files** (*List[rasterio.io.DatasetReader]*) – List of rasterio datasets to be merged
- **extent** (*List[Union[float, int]]*) – Bounds of the output image (left, bottom, right, top). If not set, bounds are determined from bounds of input rasters, e.g. `extent=[0, 972, 0, 1069]`, default is `None`
- **res** (*int*) – Output resolution in units of coordinate reference system. If not set, the resolution of the first raster is used. If a single value is passed, output pixels will be square. e.g. `res=50`, default is `None`
- **nodata** (*Union[float, int]*) – nodata value to use in output file. If not set, uses the nodata value in the first input raster, e.g. `nodata=9999.0`, default is `None`
- **precision** (*int*) – Number of decimal points of precision when computing inverse transform, e.g. `precision=2`, default is `None`
- **indices** (*int*) – Bands to read and merge, e.g. `indices=1`, default is `None`
- **method** (*str*) – Method on how to merge the tiles, e.g. `method='first'`, default is `'first'`

Returns

- **mosaic** (*np.ndarray*) – Array containing the merged tile data
- **transform** (*affine.Affine*) – Affine Transform of the merged tiles

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Creating filepath
>>> filepath = 'Documents/images/'
```

```
>>> # Creating list of filepaths
>>> filepaths = gg.raster.create_filepaths(dirpath=filepath, search_criteria='tile*.
↪tif')
>>> filepaths
['Documents/images//tile_292000_294000_5626000_5628000.tif',
'Documents/images//tile_292000_294000_5628000_5630000.tif',
'Documents/images//tile_292000_294000_5630000_5632000.tif',
'Documents/images//tile_294000_296000_5626000_5628000.tif']
```

```
>>> # Creating list of loaded rasterio objects
>>> src_list = gg.raster.create_src_list(filepaths=filepaths)
>>> src_list
[<open DatasetReader name='Documents/images/tile_292000_294000_5626000_5628000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_292000_294000_5628000_5630000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_292000_294000_5630000_5632000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_294000_296000_5626000_5628000.tif'␣
↪mode='r'>,
```

```
>>> # Merging tiles
>>> mosaic, transform = gg.raster.merge_tiles(src_files=src_list)
```

```
>>> # Inspecting the mosaic data
>>> mosaic
array([[200.72, 200.73, 200.72, ..., 204.42, 204.45, 204.45],
[200.74, 200.74, 200.75, ..., 204.43, 204.44, 204.48]
[200.76, 200.76, 200.76, ..., 204.42, 204.48, 204.5 ],
...,
[329.15, 328.86, 328.74, ..., 242.45, 242.38, 242.28],
[329.29, 329.06, 328.87, ..., 242.45, 242.39, 242.31],
[329.47, 329.3 , 329.09, ..., 242.42, 242.37, 242.32]],
dtype=float32)
```

```
>>> # Inspecting the transform of the mosaic
>>> transform
Affine(1.0, 0.0, 292000.0,
0.0, -1.0, 5632000.0)
```

gemgis.raster.reproject_raster

gemgis.raster.**reproject_raster**(*path_in: str*, *path_out: str*, *dst_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Reprojecting a raster into different CRS

Parameters

- **path_in** (*str*) – Path to the source file, e.g. `path_in='Images/'`
- **path_out** (*str*) – Path for the destination file, e.g. `path_out='Images/'`
- **dst_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – CRS of the destination file, e.g. `dst_crs='EPSG:25832'`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Changed in version 1.1: Fixing an issue where the file would be closed too soon, see https://github.com/cgre-aachen/gemgis/issues/294

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Reprojecting raster
>>> gg.raster.reproject_raster(path_in='raster_in.tif', path_out='raster_out.tif',␣
↪dst_crs='EPSG:4326')
```

gemgis.raster.resize_by_array

gemgis.raster.**resize_by_array**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *array: Union[numpy.ndarray, rasterio.io.DatasetReader]*) → numpy.ndarray

Rescaling raster to the size of another raster

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster that is being resized
- **array** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster with a size that the raster is being resized to

Returns **array_resized** – Resized array

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> import numpy as np
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Creating array
>>> array = np.zeros(100).reshape((10,10))
>>> array.shape
(10, 10)
```

```
>>> # Resizing a raster by an array
>>> raster_resized = gg.raster.resize_by_array(raster=raster, array=array)
>>> raster_resized.shape
(10, 10)
```

See also:

resize_raster Resizing a raster

gemgis.raster.resize_raster

gemgis.raster.**resize_raster**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *width: int*, *height: int*) → numpy.ndarray

Resizing raster to given dimensions

Parameters

- **array** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Array that will be resized
- **width** (*int*) – Width of the resized array, e.g. `width=100`
- **height** (*int*) – Height of the resized array, e.g. `height=100`

Returns **array_resized** – Resized array

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> import numpy as np
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Resizing raster
>>> raster_resized = gg.raster.resize_raster(raster=raster, width=10, height=10)
>>> raster_resized.shape
(10, 10)
```

See also:

resize_by_array Resizing a raster by the shape of another array

gemgis.raster.save_as_tiff

gemgis.raster.**save_as_tiff**(*raster: numpy.ndarray*, *path: str*, *extent: Union[List[Union[int, float]], Tuple[Union[int, float]]]*, *crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*, *nodata: Union[float, int] = None*, *transform=None*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Saving a np.array as tif file

Parameters

- **array** (*np.ndarray*) – Array containing the raster values
- **path** (*string*) – Path and name of the file, e.g. `path='mesh.msh'`

- **extent** (*Union[List[Union[int, float]], Tuple[Union[int, float]]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – CRS of the saved raster, e.g. `crs='EPSG:4647'`
- **nodata** (*Union[float, int]*) – Nodata value of the raster, e.g. `nodata=9999.0`, default `None`
- **transform** – Transform of the data, default is `None`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default is `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Defining raster extent and CRS
>>> extent = [0, 972, 0, 1069]
>>> crs = 'EPSG:4326'
```

```
>>> # Saving raster as tiff
>>> gg.raster.save_as_tiff(raster=raster.read(1), path='raster_saved.tif',␣
↪extent=extent, crs=crs)
Raster successfully saved
```

8.3 Visualization

The following sections provide an overview of the methods implemented in the GemGIS Visualization module.

8.3.1 Creating PolyData and Grid Data from GeoDataFrames, Rasters, and GemPy Models

The following methods are used to create PolyData from various input data formats.

gemgis.visualization.create_depth_map(mesh)	Extracting the depth values of the vertices and add them as scalars to the mesh
gemgis.visualization.create_depth_maps_from_gempy(...)	Creating depth map of model surfaces, adapted from https://github.com/cgre-aachen/gempy/blob/20550fffdd1ccb3c6a9a402bc162e7eed3dd7352/gempy/plot/vista.py#L440-L477
gemgis.visualization.create_thickness_maps(...)	Creating a thickness map using https://docs.pyvista.org/examples/01-filter/distance-between-surfaces.html#sphx-glr-examples-01-filter-distance-between-surfaces-py
gemgis.visualization.create_temperature_map(...)	Creating a temperature map for a surface at depth taking the topography into account
gemgis.visualization.create_delaunay_mesh_from_gdf(gdf)	Creating a delaunay triangulated mesh from surface contour lines
gemgis.visualization.create_dem_3d(dem[, ...])	Plotting the dem in 3D with PyVista
gemgis.visualization.create_lines_3d_linestrings(...)	Creating lines with z-component (LineString Z)
gemgis.visualization.create_lines_3d_polydata(gdf)	Creating lines with z-component for the plotting with PyVista
gemgis.visualization.create_mesh_from_cross_section(...)	Creating a PyVista Mesh from one cross section
gemgis.visualization.create_meshes_from_cross_sections(gdf)	Creating PyVista Meshes from multiple cross section
gemgis.visualization.create_meshes_hypocenters(gdf)	Plotting earthquake hypocenters with PyVista
gemgis.visualization.create_points_3d(gdf)	Plotting points in 3D with PyVista
gemgis.visualization.create_polydata_from_dxf(gdf)	Converting loaded DXF object to PyVista PolyData
gemgis.visualization.create_polydata_from_msh(data)	Converting loaded Leapfrog mesh to PyVista PolyData
gemgis.visualization.create_polydata_from_ts(data)	Converting loaded GoCAD mesh to PyVista PolyData
gemgis.visualization.create_structured_grid_from_asc(data)	Converting loaded ASC object to PyVista StructuredGrid
gemgis.visualization.create_structured_grid_from_zmap(data)	Converting loaded ZMAP object to PyVista StructuredGrid

gemgis.visualization.create_depth_map

gemgis.visualization.**create_depth_map**(*mesh: pyvista.core.pointset.PolyData*, *name: str = 'Depth [m]'*) → pyvista.core.pointset.PolyData

Extracting the depth values of the vertices and add them as scalars to the mesh

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista PolyData dataset
- **name** (*str*) – Name of the data array, e.g. `name='Depth [m]'`, default is `'Depth [m]'`

Returns **mesh** – PyVista PolyData dataset with depth values as data array

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> mesh = pv.read(filename='mesh.vtk')
>>> mesh
PolyData    Information
N Cells     4174
N Points    2303
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    0
```

```
>>> # Creating depth map from surface
>>> mesh = gg.visualization.create_depth_map(mesh=mesh)
>>> mesh
Header
PolyData    Information
N Cells     4174
N Points    2303
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       3.050e+02   7.250e+02
```

See also:

create_depth_maps_from_gempy Creating depth maps from GemPy Model Surfaces

create_thickness_maps Creating thickness map from PolyData datasets

create_temperature_map Creating temperature map from PolyData datasets

gemgis.visualization.create_depth_maps_from_gempy

gemgis.visualization.**create_depth_maps_from_gempy**(*geo_model*, *surfaces: Union[str, List[str]]*) → Dict[str, List[Union[pyvista.core.pointset.PolyData, numpy.ndarray, List[str]]]]

Creating depth map of model surfaces, adapted from https://github.com/cgre-aachen/gempy/blob/20550fffdd1ccb3c6a9a402bc162e7eed3dd7352/gempy/plot/vista.py#L440-L477

Parameters

- **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model
- **surfaces** (*Union[str, List[str]]*) – Name of the surface or list with surface names of which the depth maps are created, e.g. `surfaces=['Layer1', 'Layer2']`

Returns **surfaces_poly** – Dict containing the mesh data, depth data and color data for selected surfaces

Return type Dict[str, List[Union[pv.core.pointset.PolyData, np.ndarray, List[str]]]]

New in version 1.0.x.

Changed in version 1.1.8: Ensure compatibility with GemPy>=3

Example

```
>>> # Loading Libraries and creating depth map
>>> import gemgis as gg
>>> dict_sand1 = gg.visualization.create_depth_maps(geo_model=geo_model, surfaces=
↪'Sand1')
>>> dict_sand1
{'Sand1':   [PolyData (0x2dd0f46c820)
N Cells:    4174
N Points:   2303
X Bounds:   9.720e+00, 9.623e+02
Y Bounds:   1.881e+02, 9.491e+02
Z Bounds:   3.050e+02, 7.250e+02
N Arrays:   1,
'#015482']}
```

See also:

create_depth_map Creating depth map from PolyData dataset

create_thickness_maps Creating thickness map from PolyData datasets

create_temperature_map Creating temperature map from PolyData datasets

gemgis.visualization.create_thickness_maps

gemgis.visualization.**create_thickness_maps**(*top_surface: pyvista.core.pointset.PolyData*, *base_surface: pyvista.core.pointset.PolyData*) → pyvista.core.pointset.PolyData

Creating a thickness map using https://docs.pyvista.org/examples/01-filter/distance-between-surfaces.html#sphx-glr-examples-01-filter-distance-between-surfaces-py

Parameters

- **top_surface** (*pv.core.pointset.PolyData*) – Mesh representing the top of the layer
- **base_surface** (*pv.core.pointset.PolyData*) – Mesh representing the base of the layer

Returns **thickness** – Mesh with scalars representing the thickness of the layer

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating thickness map
>>> import gemgis as gg
>>> dict_all = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model,␣
↪surfaces=['Sand1', 'Ton'])
>>> thickness_map = gg.visualization.create_thickness_maps(top_surface=dict_all[
↪'Sand1'][0], base_surface=dict_all['Ton'][0])
>>> thickness_map
Header
PolyData    Information
N Cells     5111
N Points    2739
X Bounds    9.720e+00, 9.623e+02
Y Bounds    3.578e+02, 1.058e+03
Z Bounds    3.050e+02, 7.265e+02
N Arrays    3
Data Arrays
Name            Field   Type    N Comp  Min             Max
Data            Points  float64 1       3.050e+02   7.265e+02
Normals         Points  float32 3       -9.550e-01  6.656e-01
Thickness [m]   Points  float64 1       4.850e+01   8.761e+01
```

See also:

create_depth_map Creating depth map from PolyData dataset

create_depth_maps_from_gempy Creating depth maps from GemPy Model Surfaces

create_temperature_map Creating temperature map from PolyData datasets

gemgis.visualization.create_temperature_map

gemgis.visualization.**create_temperature_map**(*dem: rasterio.io.DatasetReader*, *mesh: pyvista.core.pointset.PolyData*, *name: str = 'Thickness [m]'*, *apply_threshold: bool = True*, *tsurface: Union[float, int] = 10*, *gradient: Union[float, int] = 0.03*) → pyvista.core.pointset.PolyData

Creating a temperature map for a surface at depth taking the topography into account

Parameters

- **dem** (*rasterio.io.DatasetReader*) – Digital Elevation Model of the area
- **mesh** (*pv.core.pointset.PolyData*) – PolyData dataset for which the temperature at depth will be calculated
- **name** (*str*) – Name of the array to be added to the mesh, e.g. `name='Thickness [m]'`, default is `'Thickness [m]'`
- **apply_threshold** (*bool*) – Variable to apply a threshold to the mesh to remove vertices that were located above the topography. Options include: `True` or `False`, default set to `True`
- **tsurface** (*Union[float, int]*) – Surface temperature in degrees Celsius, e.g. `tsurface=10`, default is `10` degrees C

- **gradient** (*Union[float, int]*) – Geothermal gradient in degrees celsius per meter, e.g. `gradient=0.03`, default is `0.03` degrees C per m

Returns **mesh** – PolyData dataset including a temperature data array

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and Files
>>> import gemgis as gg
>>> import rasterio
>>> import pyvista as pv
>>> dem = rasterio.open(fp='raster.tif')
>>> mesh = pv.read(filename='mesh1.vtk')
>>> mesh
PolyData    Information
N Cells     4174
N Points    2303
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    0
```

```
>>> # Creating temperature map
>>> mesh = gg.visualization.create_temperature_map(dem=dem, mesh=mesh)
>>> mesh
Header
UnstructuredGrid    Information
N Cells     3946
N Points    2130
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    2
Data Arrays
Name                Field   Type    N Comp  Min         Max
Thickness [m]       Points  float64 1       9.321e-02   2.020e+02
Temperature [°C]    Points  float64 1       1.000e+01   1.606e+01
```

See also:

create_depth_map Creating depth map from PolyData dataset

create_depth_maps_from_gempy Creating depth maps from GemPy Model Surfaces

create_thickness_maps Creating thickness map from PolyData datasets

gemgis.visualization.create_delaunay_mesh_from_gdf

gemgis.visualization.**create_delaunay_mesh_from_gdf**(*gdf: geopandas.geodataframe.GeoDataFrame*, *z: str = 'Z'*) → pyvista.core.pointset.PolyData

Creating a delaunay triangulated mesh from surface contour lines

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing LineStrings representing surface contours

Returns **mesh** – Mesh representing the triangulated mesh

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    OBJECTID    Z       EINHEIT         Shape_Leng  geometry
0   1.00        -2450   gg_kru_b_l_Z50m 3924.67     LINESTRING (32403313.109␣
↪5785053.637, 32402917...
1   2.00        -2400   gg_kru_b_l_Z50m 26332.90    LINESTRING (32410198.859␣
↪5781110.785, 32409807...
2   3.00        -2350   gg_kru_b_l_Z50m 31104.28    LINESTRING (32409587.930␣
↪5780538.824, 32408824...
3   4.00        -2300   gg_kru_b_l_Z50m 35631.73    LINESTRING (32408977.008␣
↪5779966.863, 32408808...
4   5.00        -2250   gg_kru_b_l_Z50m 41702.52    LINESTRING (32407319.922␣
↪5779788.672, 32407246...
```

```
>>> # Creating PolyData from isolines
>>> mesh = gg.visualization.create_delaunay_mesh_from_gdf(gdf=gdf)
>>> mesh
Header
PolyData    Information
N Cells     45651
N Points    23009
X Bounds    3.233e+07, 3.250e+07
Y Bounds    5.702e+06, 5.798e+06
Z Bounds    -2.450e+03, 4.000e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -2.450e+03  4.000e+02
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

gemgis.visualization.create_dem_3d

gemgis.visualization.**create_dem_3d**(*dem: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *extent: List[Union[int, float]] = None*, *res: int = 1*) → pyvista.core.pointset.StructuredGrid

Plotting the dem in 3D with PyVista

Parameters

- **dem** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Rasterio object or NumPy array containing the height values
- **extent** (*List[Union[int, float]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **res** (*int*) – Resolution of the meshgrid, e.g. `resolution=1`, default is 1

Returns **grid** – Grid storing the elevation data

Return type pyvista.core.pointset.StructuredGrid

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Defining raster extent
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Creating mesh from raster data
>>> grid = gg.visualization.create_dem_3d(dem=raster.read(1), extent=extent)
>>> grid
Header
             StructuredGrid  Information
N            Cells           1037028
N            Points          1039068
X            Bounds          0.000e+00, 9.710e+02
Y            Bounds          0.000e+00, 1.068e+03
Z            Bounds          2.650e+02, 7.300e+02
Dimensions                   1069, 972, 1
N Arrays                     1
Data Arrays
Name         Field   Type    N Comp  Min         Max
Elevation    Points  float64 1       2.656e+02   7.305e+02
```

See also:

create_lines_3d_polydata Creating a mesh from lines

create_points_3d Creating a mesh from points

gemgis.visualization.create_lines_3d_linestrings

gemgis.visualization.**create_lines_3d_linestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *extent: List[Union[int, float]] = None*) → geopandas.geodataframe.GeoDataFrame

Creating lines with z-component (LineString Z)

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the LineStrings to be converted to linestrings with z-component
- **dem** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Rasterio object or NumPy array containing the height values
- **extent** (*List[Union[int, float]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`

Returns **gdf_3d** – GeoDataFrame containing the LineStrings with Z component (LineString Z)

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
id      formation   geometry
0   None    Unterjura   LINESTRING (32522415.430 5777985.396, 32521520...
1   None    Unterjura   LINESTRING (32479802.616 5782183.163, 32480593...
2   None    Mitteljura  LINESTRING (32522376.263 5779907.729, 32520580...
3   None    Mitteljura  LINESTRING (32463272.196 5788327.350, 32464107...
```

```
>>> # Loading Digital Elevation Model
>>> dem = rasterio.open('raster.tif')
```

```
>>> # Create LineStrings with Z-component
>>> gdf_3d = gg.visualization.create_lines_3d_linestrings(gdf=gdf, dem=dem)
>>> gdf_3d
    id      formation   geometry
0   None    Unterjura   LINESTRING Z (32522415.430 5777985.396 213.000...
1   None    Unterjura   LINESTRING Z (32479802.616 5782183.163 84.000,...
2   None    Mitteljura  LINESTRING Z (32522376.263 5779907.729 116.000...
3   None    Mitteljura  LINESTRING Z (32463272.196 5788327.350 102.000...
```

See also:

create_lines_3d_polydata Creating lines with z-component for the plotting with PyVista

create_dem_3d Creating a mesh from a Digital Elevation Model

create_points_3d Creating a mesh from points

gemgis.visualization.create_lines_3d_polydata

gemgis.visualization.**create_lines_3d_polydata**(*gdf: geopandas.geodataframe.GeoDataFrame*) → pyvista.core.pointset.PolyData

Creating lines with z-component for the plotting with PyVista

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the contour information

Returns **poly** – PyVista Polydata Set containing the lines and vertices

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      Z   geometry
0   None    400 LINESTRING (0.741 475.441, 35.629 429.247, 77....
1   None    300 LINESTRING (645.965 0.525, 685.141 61.866, 724...
2   None    400 LINESTRING (490.292 0.525, 505.756 40.732, 519...
3   None    600 LINESTRING (911.433 1068.585, 908.856 1026.831...
4   None    700 LINESTRING (228.432 1068.585, 239.772 1017.037...
```

```
>>> # Create mesh from LineStrings
>>> polydata = gg.visualization.create_lines_3d_polydata(gdf=gdf)
>>> polydata
    PolyData    Information
N   Cells       7
N   Points      121
X   Bounds      7.409e-01, 9.717e+02
Y   Bounds      5.250e-01, 1.069e+03
Z   Bounds      3.000e+02, 7.000e+02
N   Arrays      0
```

See also:

create_dem_3d Creating a mesh from a Digital Elevation Model

create_points_3d Creating a mesh from points

gemgis.visualization.create_mesh_from_cross_section

gemgis.visualization.**create_mesh_from_cross_section**(*linestring: shapely.geometry.linestring.LineString*, *zmax: Union[float, int]*, *zmin: Union[float, int]*) → pyvista.core.pointset.PolyData

Creating a PyVista Mesh from one cross section

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – LineString representing the trace of the cross section on a geological map, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`
- **zmax** (*Union[float, int]*) – Upper vertical extent of the cross section, e.g. `zmax=1000`
- **zmin** (*Union[float, int]*) – Lower vertical extent of the cross section, e.g. `zmin=0`

Returns **surface** – Mesh defining the cross section in space

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating PolyData from LineStrings
>>> polydata = gg.visualization.create_mesh_from_cross_
↪section(linestring=linestring, zmax=1000, zmin=0)
>>> polydata
Header
PolyData    Information
N Cells     4
N Points    6
X Bounds    0.000e+00, 2.000e+01
Y Bounds    0.000e+00, 2.000e+01
Z Bounds    0.000e+00, 1.000e+03
N Arrays    1
Data Arrays
Name                Field   Type    N Comp  Min         Max
Texture Coordinates Points  float64 2       0.000e+00   1.000e+00
```

See also:

create_meshes_from_cross_sections Creating meshes from cross sections

gemgis.visualization.create_meshes_from_cross_sections

gemgis.visualization.**create_meshes_from_cross_sections**(*gdf: geopandas.geodataframe.GeoDataFrame*) → List[pyvista.core.pointset.PolyData]

Creating PyVista Meshes from multiple cross section

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the traces of the profiles as LineStrings

Returns **meshes_list** – List containing the meshes of all profiles

Return type List[pyvista.core.pointset.PolyData]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      zmax    zmin    name        geometry
0   None    500     -6000   Muenster    LINESTRING (32386148.890 5763304.720,␣
↪32393549...
1   None    500     -2000   Rheine      LINESTRING (32402275.136 5761828.501,␣
↪32431165...
```

```
>>> # Creating list of PolyData datasets from GeoDataFrame
>>> meshes_list = gg.visualization.create_meshes_from_cross_sections(gdf=gdf)
>>> meshes_list
[PolyData (0x2526e543ee0)
N Cells:    20
N Points:   22
X Bounds:   3.239e+07, 3.242e+07
Y Bounds:   5.717e+06, 5.763e+06
Z Bounds:   -6.000e+03, 5.000e+02
N Arrays:   1,
PolyData (0x2526a4687c0)
N Cells:    2
N Points:   4
X Bounds:   3.240e+07, 3.243e+07
Y Bounds:   5.762e+06, 5.814e+06
Z Bounds:   -2.000e+03, 5.000e+02
N Arrays:   1]
```

See also:

create_mesh_from_cross_section Creating a mesh from a cross section

gemgis.visualization.create_meshes_hypocenters

gemgis.visualization.**create_meshes_hypocenters**(*gdf: geopandas.geodataframe.GeoDataFrame*, *magnitude: str = 'Magnitude'*, *magnitude_factor: int = 200*, *year: str = 'Year'*) → pyvista.core.composite.MultiBlock

Plotting earthquake hypocenters with PyVista

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the earthquake hypocenter data
- **magnitude** (*str*) – Name for the column containing the magnitude value, e.g. `magnitude='Magnitude'`, default is `'Magnitude'`
- **magnitude_factor** (*int*) – Scaling factor for the magnitude values defining the size of the spheres, e.g. `magnitude_factor=200`, default is `200`
- **year** (*str*) – Name for the column containing the year of each earthquake event, e.g. `year='Year'`, default to `'Year'`

Returns **spheres** – PyVista MultiBlock object containing the hypocenters stored as spheres

Return type pv.core.composite.MultiBlock

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    Y           X           Z           RASTERVALU  Tiefe [km]  Magnitude  ␣
↪Epizentrum      Year    geometry
0   5645741.63  32322660.15 -8249.25    150.75      8.40        1.50       ␣
↪STETTERNICH     2002    POINT (32322660.151 5645741.630)
1   5645947.18  32323159.51 89.63       89.63       0.00        0.80       ␣
↪SOPHIENHOEHE    2014    POINT (32323159.505 5645947.183)
```

```
>>> # Creating Spheres for hypocenters
>>> spheres = gg.visualization.create_meshes_hypocenters(gdf=gdf)
>>> spheres
Information
MultiBlock  Values
N Blocks    497
X Bounds    32287780.000, 32328260.000
Y Bounds    5620074.000, 5648385.000
Z Bounds    -24317.020, 309.130
Blocks
Index   Name        Type
0       Block-00    PolyData
1       Block-01    PolyData
2       Block-02    PolyData
```

gemgis.visualization.create_points_3d

gemgis.visualization.**create_points_3d**(*gdf: geopandas.geodataframe.GeoDataFrame*) → pyvista.core.pointset.PolyData

Plotting points in 3D with PyVista

Parameters **points** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points including X, Y, and Z columns

Returns **points_mesh** – PyVista PolyData Pointset

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0   None    Ton         POINT (19.150 293.313)
1   None    Ton         POINT (61.934 381.459)
2   None    Ton         POINT (109.358 480.946)
3   None    Ton         POINT (157.812 615.999)
4   None    Ton         POINT (191.318 719.094)
```

```
>>> # Creating PolyData from points
>>> polydata = gg.visualization.create_points_3d(gdf=gdf)
>>> polydata
    PolyData    Information
N Cells     41
N Points        41
X Bounds        8.841e+00, 9.661e+02
Y Bounds        1.650e+02, 1.045e+03
Z Bounds        2.769e+02, 7.220e+02
N Arrays        0
```

See also:

create_lines_3d_polydata Creating a mesh from lines

create_dem_3d Creating a mesh from a Digital Elevation model

gemgis.visualization.create_polydata_from_dxf

gemgis.visualization.**create_polydata_from_dxf**(*gdf: geopandas.geodataframe.GeoDataFrame*) → pyvista.core.pointset.PolyData

Converting loaded DXF object to PyVista PolyData

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the faces/polygons of the loaded DXF object

Returns **polydata** – PyVista PolyData containing the mesh values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.dxf')
>>> gdf
    geometry
0   POLYGON Z ((1.00869 0.92852 1.00000, 0.97744 0...
1   POLYGON Z ((1.00869 0.92852 1.00000, 1.01735 0...
2   POLYGON Z ((0.97744 0.92853 1.00000, 0.94619 0...
3   POLYGON Z ((0.97744 0.92853 1.00000, 0.98610 0...
4   POLYGON Z ((0.94619 0.92853 1.00000, 0.91494 0...
```

```
>>> # Creating PolyData from dxf file
>>> polydata = gg.visualization.create_polydata_from_dxf(gdf=gdf)
>>> polydata
PolyData    Information
N Cells     98304
N Points    393216
X Bounds    -1.576e+00, 2.530e+00
Y Bounds    -9.751e+00, 1.000e+00
Z Bounds    -9.167e-01, 1.000e+00
N Arrays        0
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.create_polydata_from_msh

gemgis.visualization.**create_polydata_from_msh**(*data: Dict[str, numpy.ndarray]*) → pyvista.core.pointset.PolyData

Converting loaded Leapfrog mesh to PyVista PolyData

Parameters **data** (*Dict[str, np.ndarray]*) – Dict containing the data loaded from a Leapfrog mesh with read_msh() of the raster module

Returns **polydata** – PyVista PolyData containing the mesh values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> data = gg.raster.read_msh('mesh.msh')
>>> data
{'Tri': array([[    0,     1,     2],
[    0,     3,     1],
[    4,     3,     0],
...,
[53677, 53672, 53680],
[53679, 53677, 53680],
[53673, 53672, 53677]]),
'Location': array([[ 1.44625109e+06,  5.24854344e+06, -1.12743862e+02],
[ 1.44624766e+06,  5.24854640e+06, -1.15102216e+02],
[ 1.44624808e+06,  5.24854657e+06, -1.15080548e+02],
...,
[ 1.44831008e+06,  5.24896679e+06, -1.24755449e+02],
[ 1.44830385e+06,  5.24896985e+06, -1.33694397e+02],
[ 1.44829874e+06,  5.24897215e+06, -1.42506587e+02]])}
```

```
>>> # Creating PolyData from msh file
>>> polydata = gg.visualization.create_polydata_from_msh(data=data)
>>> polydata
PolyData    Information
N Cells     107358
N Points    53681
X Bounds    1.444e+06, 1.449e+06
Y Bounds    5.246e+06, 5.249e+06
Z Bounds    -2.464e+02, 7.396e+02
N Arrays    0
```

See also:

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.create_polydata_from_ts

gemgis.visualization.**create_polydata_from_ts**(*data: Tuple[list, list]*, *concat: bool = False*) → pyvista.core.pointset.PolyData

Converting loaded GoCAD mesh to PyVista PolyData

Parameters

- **data** (*Tuple[list, list]*) – Tuple containing the data loaded from a GoCAD mesh with read_ts() of the raster module
- **concat** (*bool*) – Boolean defining whether the DataFrames should be concatenated or not

Returns **polydata** – PyVista PolyData containing the mesh values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> vertices, faces = gg.raster.read_ts('mesh.ts')
```

```
>>> # Inspecting vertices
>>> vertices
    id  X           Y           Z
0   0   297077.41   5677487.26  -838.50
1   1   297437.54   5676992.09  -816.61
```

```
>>> # Inspecting
>>> faces
array([[    0,     1,     2],
[    3,     2,     4],
[    1,     5,     6],...,
[40335, 40338, 40336],
[40339, 40340, 40341],
[40341, 40342, 40339]])
```

```
>>> # Creating PolyData from ts file
>>> polydata = gg.visualization.create_polydata_from_ts((vertices, faces))
>>> polydata
PolyData    Information
N Cells     29273
N Points    40343
X Bounds    2.804e+05, 5.161e+05
Y Bounds    5.640e+06, 5.833e+06
Z Bounds    -8.067e+03, 1.457e+02
N Arrays    0
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.create_structured_grid_from_asc

gemgis.visualization.**create_structured_grid_from_asc**(*data: dict*) → pyvista.core.pointset.StructuredGrid

Converting loaded ASC object to PyVista StructuredGrid

Parameters **data** (*dict*) – Dict containing the extracted ASC data using read_asc(...) of the raster module

Returns **grid** – PyVista StructuredGrid created from ASC data

Return type pv.core.pointset.StructuredGrid

New in version 1.0.x.

Example

```
>>> # Loading Libraries and data
>>> import gemgis as gg
>>> data = gg.raster.read_asc('raster.asc')
```

```
>>> # Creating StructuredGrid from data
>>> grid = gg.visualization.create_structured_grid_from_asc(data=data)
>>> grid
Header  Data Arrays
StructuredGrid  Information
N Cells         2880012
N Points        2883540
X Bounds        -4.225e+04, 2.788e+05
Y Bounds        3.060e+05, 8.668e+05
Z Bounds        -1.000e+05, 2.880e+02
Dimensions      2244, 1285, 1
N Arrays        1
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -1.132e+04  2.887e+02
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.create_structured_grid_from_zmap

gemgis.visualization.**create_structured_grid_from_zmap**(*data: dict*) → pyvista.core.pointset.StructuredGrid

Converting loaded ZMAP object to PyVista StructuredGrid

Parameters **data** (*dict*) – Dict containing the extracted ZAMP data using read_zmap(...) of the raster module

Returns **grid** – PyVista StructuredGrid created from zmap data

Return type pv.core.pointset.StructuredGrid

New in version 1.0.x.

Example

```
>>> # Loading Libraries and data
>>> import gemgis as gg
>>> data = gg.raster.read_zmap('raster.dat')
```

```
>>> # Creating StructuredGrid from data
>>> grid = gg.visualization.create_structured_grid_from_zmap(data=data)
>>> grid
Header  Data Arrays
StructuredGrid  Information
N Cells         2880012
N Points        2883540
X Bounds        -4.225e+04, 2.788e+05
Y Bounds        3.060e+05, 8.668e+05
Z Bounds        -1.000e+05, 2.880e+02
Dimensions      2244, 1285, 1
N Arrays        1
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -1.132e+04  2.887e+02
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

8.3.2 Working with Boreholes

The following methods are used to work with boreholes in GemGIS.

gemgis.visualization.add_row_to_boreholes(...)	Adding additional row to each borehole for further processing for 3D visualization
gemgis.visualization.create_borehole_labels(df)	Create labels for borehole plots.
gemgis.visualization.create_borehole_tube(df, ...)	Creating a tube from a line for the 3D visualization of boreholes
gemgis.visualization.create_borehole_tubes(df, ...)	Creating PyVista Tubes for plotting boreholes in 3D
gemgis.visualization.create_boreholes_3d(df, ...)	Plotting boreholes in 3D
gemgis.visualization.create_lines_from_points(df)	Creating a line set from a Pandas DataFrame
gemgis.visualization.create_deviated_borehole_df(...)	Creating Pandas DataFrame containing parameters to create 3D boreholes
gemgis.visualization.create_deviated_boreholes_3d(...)	Plotting boreholes in 3D
gemgis.visualization.group_borehole_dataframe(df)	Grouping Borehole DataFrame by Index
gemgis.visualization.resample_between_well_deviation_points(...)	Resampling between points that define the path of a well
gemgis.visualization.show_well_log_along_well(...)	Function to return a tube representing well log values along a well path

gemgis.visualization.add_row_to_boreholes

gemgis.visualization.**add_row_to_boreholes**(*df_groups: List[pandas.core.frame.DataFrame]*) → List[pandas.core.frame.DataFrame]

Adding additional row to each borehole for further processing for 3D visualization

Parameters **df_groups** (*List[pd.DataFrame]*) – List of Pandas DataFrames containing the borehole data

Returns **df_groups** – List of Pandas DataFrames with additional row

Return type List[pd.DataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                          X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
```

(continues on next page)

(continued from previous page)

```
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Adding row to DataFrames
>>> grouped = df.groupby(['Index'])
>>> df_groups = [grouped.get_group(x) for x in grouped.groups]
>>> list_df = gg.visualization.add_row_to_boreholes(df_groups)
>>> list_df[0]
    Unnamed: 0  Index   Name                            X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   NaN         GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00                  NaN
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

See also:

create_lines_from_points Creating lines from points

create_borehole_tube Creating borehole tube

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.create_borehole_labels

gemgis.visualization.**create_borehole_labels**(*df: Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*) → pyvista.core.pointset.PolyData

Create labels for borehole plots.

Parameters **df** (*Union[pd.DataFrame, gpd.geodataframe.GeoDataFrame]*) – (Geo-)DataFrame containing the borehole data.

Returns **borehole_locations** – Borehole locations with labels.

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Changed in version 1.1.1: Fixed a ValueError that was introduced with pandas>2.0.0.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating borehole labels
>>> labels = gg.visualization.create_borehole_labels(df=df)
>>> labels
Header
PolyData    Information
N Cells     2
N Points    2
X Bounds    3.239e+07, 3.240e+07
Y Bounds    5.753e+06, 5.763e+06
Z Bounds    6.000e+01, 1.070e+02
N Arrays    1
Data Arrays
Name    Field   Type    N Comp  Min Max
Labels  Points          1       nan nan
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing.

create_lines_from_points Creating lines from points.

create_borehole_tube Creating borehole tube.

create_borehole_tubes Creating tubes from lines.

create_boreholes_3d Creating PyVista objects for plotting.

gemgis.visualization.create_borehole_tube

gemgis.visualization.**create_borehole_tube**(*df: pandas.core.frame.DataFrame*, *line: pyvista.core.pointset.PolyData*, *radius: Union[float, int]*) → pyvista.core.pointset.PolyData

Creating a tube from a line for the 3D visualization of boreholes

Parameters

- **df** (*pd.DataFrame*) – DataFrame containing the borehole data
- **line** (*pv.core.pointset.PolyData*) – PyVista line object
- **radius** (*Union[float,int]*) – Radius of the tube, e.g. `'radius=10'`

Returns **tube** – PolyData Object representing the borehole tube

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Adding row to DataFrames
>>> grouped = df.groupby(['Index'])
>>> df_groups = [grouped.get_group(x) for x in grouped.groups]
>>> list_df = gg.visualization.add_row_to_boreholes(df_groups)
>>> list_df[0]
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   NaN         GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00                  NaN
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating Lines from points
>>> line = gg.visualization.create_lines_from_points(df=list_df[0])
>>> line
PolyData    Information
N Cells     39
N Points    20
X Bounds    3.239e+07, 3.239e+07
Y Bounds    5.763e+06, 5.763e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    0
```

```
>>> # Creating Tubes from lines
>>> tube = gg.visualization.create_borehole_tube(df=list_df[0], line=line,␣
↪radius=100)
>>> tube
Header
PolyData    Information
N Cells     418
N Points    1520
X Bounds    3.239e+07, 3.239e+07
```

(continues on next page)

(continued from previous page)

```
Y Bounds    5.762e+06, 5.764e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    2
Data Arrays
Name        Field   Type    N Comp  Min         Max
scalars     Points  int32   1       0.000e+00   1.900e+01
TubeNormals Points  float32 3       -1.000e+00  1.000e+00
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_lines_from_points Creating lines from points

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.create_borehole_tubes

gemgis.visualization.**create_borehole_tubes**(*df: pandas.core.frame.DataFrame*, *min_length: Union[float, int]*, *radius: Union[int, float] = 10*) → Tuple[List[pyvista.core.pointset.PolyData], List[pandas.core.frame.DataFrame]]

Creating PyVista Tubes for plotting boreholes in 3D

Parameters

- **df** (*pd.DataFrame*) – DataFrame containing the extracted borehole data
- **min_length** (*Union[float, int]*) – Length defining the minimum depth of boreholes to be plotted, e.g. `min_length=1000`
- **radius** (*Union[int, float]*) – Radius of the boreholes plotted with PyVista, e.g. `radius=100` default is `10` m

Returns

- **tubes** (*List[pv.core.pointset.PolyData]*) – List of PyVista PolyData Objects
- **df_groups** (*List[pd.DataFrame]*) – List of DataFrames containing the borehole data

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                            X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
```

(continues on next page)

(continued from previous page)

```
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating borehole tubes
>>> tubes, df_groups = gg.visualization.create_borehole_tubes(df=df, min_
↪length=1000, radius=100)
>>> tubes[0]
Header
PolyData    Information
N Cells     418
N Points    1520
X Bounds    3.239e+07, 3.239e+07
Y Bounds    5.762e+06, 5.764e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    2
Data Arrays
Name        Field   Type    N Comp  Min         Max
scalars     Points  int32   1       0.000e+00   1.900e+01
TubeNormals Points  float32 3       -1.000e+00  1.000e+00
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_lines_from_points Creating lines from points

create_borehole_tube Creating borehole tube

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.create_boreholes_3d

gemgis.visualization.**create_boreholes_3d**(*df: pandas.core.frame.DataFrame*, *min_length: Union[float, int]*, *color_dict: dict*, *radius: Union[float, int] = 10*) → Tuple[List[pyvista.core.pointset.PolyData], pyvista.core.pointset.PolyData, List[pandas.core.frame.DataFrame]]

Plotting boreholes in 3D

Parameters

- **df** (*pd.DataFrame*) – DataFrame containing the extracted borehole data
- **min_length** (*Union[float, int]*) – Value defining the minimum depth of boreholes to be plotted, e.g. `min_length=1000`
- **color_dict** (*dict*) – Dict containing the surface colors of the model
- **radius** (*Union[float, int]*) – Values of the radius of the boreholes plotted with PyVista, e.g. `radius=100`, default is `10`

Returns

- **tubes** (*List[pv.core.pointset.PolyData]*) – List of PyVista tubes
- **labels** (*pv.core.pointset.PolyData*) – PyVista PolyData with Borehole Labels
- **df_groups** (*List[pd.DataFrame]*) – List containing DataFrames

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating tubes
>>> tubes, labels, df_groups = gg.visualization.create_boreholes_3d(df=df, min_
↪length=10, color_dict=color_dict, radius=1000)
>>> tubes
Information
MultiBlock  Values
N Blocks    2
X Bounds    32385176.360, 32404939.830
Y Bounds    5751889.550, 5764283.150
Z Bounds    -5849.000, 107.000
Blocks
Index   Name        Type
0       Block-00    PolyData
1       Block-01    PolyData
```

```
>>> # Inspecting labels
>>> labels
        Header
PolyData    Information
N Cells     2
N Points    2
X Bounds    3.239e+07, 3.240e+07
Y Bounds    5.753e+06, 5.763e+06
Z Bounds    6.000e+01, 1.070e+02
N Arrays    1
Data Arrays
Name    Field   Type    N Comp  Min Max
Labels  Points          1       nan nan
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_lines_from_points Creating lines from points

create_borehole_tube Creating borehole tube

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

gemgis.visualization.create_lines_from_points

gemgis.visualization.**create_lines_from_points**(*df: pandas.core.frame.DataFrame*) → pyvista.core.pointset.PolyData

Creating a line set from a Pandas DataFrame

Parameters **df** (*pd.DataFrame*) – Pandas DataFrame containing the data for one borehole

Returns **poly** – Creating borehole traces from points

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Adding row to DataFrames
>>> grouped = df.groupby(['Index'])
>>> df_groups = [grouped.get_group(x) for x in grouped.groups]
>>> list_df = gg.visualization.add_row_to_boreholes(df_groups)
>>> list_df[0]
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   NaN         GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00                  NaN
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating Lines from points
>>> line = gg.visualization.create_lines_from_points(df=list_df[0])
>>> line
PolyData    Information
N Cells     39
N Points    20
```

(continues on next page)

(continued from previous page)

```
X Bounds    3.239e+07, 3.239e+07
Y Bounds    5.763e+06, 5.763e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    0
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_borehole_tube Creating borehole tube

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.create_deviated_borehole_df

gemgis.visualization.**create_deviated_borehole_df**(*df_survey: pandas.core.frame.DataFrame*, *position: Union[numpy.ndarray, shapely.geometry.point.Point]*, *depth: str = 'depth'*, *dip: str = 'dip'*, *azimuth: str = 'azimuth'*) → pandas.core.frame.DataFrame

Creating Pandas DataFrame containing parameters to create 3D boreholes

Parameters

- **df_survey** (*pd.DataFrame*) – Pandas DataFrame containing the survey data of one borehole
- **position** (*np.ndarray*) – NumPy array containing the X, Y, and Z coordinates/the position of the borehole, e.g. `position = np.array([12012.68053 , 30557.53476 , 2325.532416])`
- **depth** (*str*) – Name of the column that contains the depth values, e.g. `depth='depth'`, default is `'depth'`
- **dip** (*str*) – Name of the column that contains the dip values, e.g. `dip='dip'`, default is `'dip'`
- **azimuth** (*str*) – Name of the column that contains the azimuth values, e.g. `azimuth='azimuth'` default is `'azimuth'`

Returns **df_survey** – Pandas DataFrame containing parameters to create 3D boreholes

Return type pd.DataFrame

New in version 1.0.x.

Changed in version 1.1.7.

Replace pandas append with concat.

Example

```
>>> # Loading Libraries and file
>>> import gemgis as gg
>>> import pandas as pd
>>> df_survey = pd.read_csv('survey.csv')
    holeid      depth   dip     azimuth
0   SonicS_006  0       90.00   20
1   SonicS_006  10      89.50   20
2   SonicS_006  20      89.00   20
3   SonicS_006  30      88.50   20
4   SonicS_006  40      88.00   20
```

```
>>> # Defining the position of the borehole at the surface
>>> position = np.array([12012.68053 , 30557.53476 ,  2325.532416])
```

```
>>> # Creating the survey DataFrame with additional parameters
>>> df_survey = gg.visualization.create_deviated_well_df(df_survey=df_survey,
↪position=position)
```

gemgis.visualization.create_deviated_boreholes_3d

gemgis.visualization.**create_deviated_boreholes_3d**(*df_collar: pandas.core.frame.DataFrame*, *df_survey: pandas.core.frame.DataFrame*, *min_length: Union[float, int]*, *radius: Union[float, int] = 10*, *collar_depth: str = 'Depth'*, *survey_depth: str = 'Depth'*, *index: str = 'Index'*, *dip: str = 'dip'*, *azimuth: str = 'azimuth'*) → Tuple[List[pyvista.core.pointset.PolyData], pyvista.core.pointset.PolyData, List[pandas.core.frame.DataFrame]]

Plotting boreholes in 3D

Parameters

- **df_collar** (*pd.DataFrame*) – DataFrame containing the extracted borehole data
- **df_survey** (*pd.DataFrame*) – DataFrame containing the extracted borehole survey data
- **min_length** (*Union[float, int]*) – Value defining the minimum depth of boreholes to be plotted, e.g. `min_length=1000`
- **color_dict** (*dict*) – Dict containing the surface colors of the model
- **radius** (*Union[float, int]*) – Values of the radius of the boreholes plotted with PyVista, e.g. `radius=100`, default is `10`
- **collar_depth** (*str*) – Name of the column that contains the depth values, e.g. `collar_depth='depth'`, default is `'Depth'`
- **survey_depth** (*str*) – Name of the column that contains the depth values, e.g. `survey_depth='depth'`, default is `'Depth'`
- **index** (*str*) – Name of the column that contains the index values, e.g. `index='index'`, default is `'index'`

- **dip** (*str*) – Name of the column that contains the dip values, e.g. `dip='dip'`, default is `'dip'`
- **azimuth** (*str*) – Name of the column that contains the azimuth values, e.g. `azimuth='azimuth'` default is `'azimuth'`

Returns

- **tubes** (*List[pv.core.pointset.PolyData]*) – List of PyVista tubes
- **labels** (*pv.core.pointset.PolyData*) – PyVista PolyData with Borehole Labels
- **df_groups** (*List[pd.DataFrame]*) – List containing DataFrames

New in version 1.0.x.

Example

gemgis.visualization.group_borehole_dataframe

gemgis.visualization.**group_borehole_dataframe**(*df: pandas.core.frame.DataFrame*) → List[pandas.core.frame.DataFrame]

Grouping Borehole DataFrame by Index

Parameters **df** (*pd.DataFrame*) – Pandas DataFrame containing the borehole data

Returns **df_groups**

Return type List[pd.DataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
```

```
>>> # Creating groups
>>> df_groups = gg.visualization.group_borehole_dataframe(df=df)
```

gemgis.visualization.resample_between_well_deviation_points

gemgis.visualization.**resample_between_well_deviation_points**(*coordinates: numpy.ndarray*) → numpy.ndarray

Resampling between points that define the path of a well

Parameters **coordinates** (*np.ndarray*) – Nx3 Numpy array containing the X, Y, and Z coordinates that define the path of a well

Returns **points_resampled** – Resampled points along a well

Return type np.ndarray

New in version 1.0.x.

gemgis.visualization.show_well_log_along_well

gemgis.visualization.**show_well_log_along_well**(*coordinates: numpy.ndarray*, *dist: numpy.ndarray*, *values: numpy.ndarray*, *radius_factor: Union[int, float] = 2*) → pyvista.core.pointset.PolyData

Function to return a tube representing well log values along a well path

Parameters

- **coordinates** (*np.ndarray*) – Nx3 Numpy array containing the X, Y, and Z coordinates that define the path of a well
- **dist** (*np.ndarray*) – np.ndarray containing the measured depths (MD) of values along the well path
- **values** (*np.ndarray*) – np.ndarray containing the measured well log values along the well path
- **radius_factor** (*int, float*) – Radius factor to adjust the diameter of the tube, e.g. `radius_factor=2`, default is `2`

Returns **tube_along_spline** – PyVista PolyData Pointset representing the the measured well log values along the well path

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

8.3.3 Miscellaneous visualization methods

The following methods are further visualization methods used in GemGIS.

gemgis.visualization.calculate_vector(dip, ...)	Calculating the plunge vector of a borehole section
gemgis.visualization.clip_seismic_data(...)	Clipping seismic data loaded with segysak to CDP defined start and end CDP values
gemgis.visualization.convert_to_rgb(array)	Converting array values to RGB values
gemgis.visualization.drape_array_over_dem(...)	Creating grid and texture to drape array over a digital elevation model
gemgis.visualization.get_batlow_cmap()	Returning the Batlow cmap from https://github.com/callumrollo/cmcrameri/blob/master/cmcrameri/cmaps/batlow.txt
gemgis.visualization.get_color_lot(geo_model)	Method to get the right color list depending on the type of plot.
gemgis.visualization.get_mesh_geological_map(...)	Getting the geological map of a GemPy Model draped over the topography as mesh.
gemgis.visualization.get_petrel_cmap()	Returning the Petrel cmap
gemgis.visualization.get_points_along_spline(...)	Returning the closest point on the spline a given a length along a spline.
gemgis.visualization.get_seismic_cmap()	Returning the seismic cmap from https://github.com/lperozzi/Seismic_colormaps/blob/master/colormaps.py
gemgis.visualization.plane_through_hypocenters(spheres)	Fitting a plane through the hypocenters of earthquakes using Eigenvector analysis
gemgis.visualization.plot_data(geo_data[, ...])	Plotting Input Data
gemgis.visualization.plot_orientations(gdf)	Plotting orientation values of a GeoDataFrame with mplstereonet
gemgis.visualization.polyline_from_points(points)	Creating PyVista PolyLine from points
gemgis.visualization.read_raster([path, ...])	Reading a raster and returning a mesh
gemgis.visualization.seismic_to_array(...[, ...])	Converting seismic data loaded with segysak to a NumPy array
gemgis.visualization.seismic_to_mesh(...[, ...])	Converting seismic data loaded with segysak to a PyVista Mesh

gemgis.visualization.calculate_vector

gemgis.visualization.**calculate_vector**(*dip: Union[float, int]*, *azimuth: Union[float, int]*) → numpy.ndarray

Calculating the plunge vector of a borehole section

Parameters

- **dip** (*Union[float, int]*) – Dipping value of a borehole segment, e.g. `dip=90`
- **azimuth** (*Union[float, int]*) – Dipping direction of a borehole segment, e.g. `azimuth=20`

Returns **vector** – Plunging/dipping vector of a borehole segment

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and define dip and azimuth
>>> import gemgis as gg
>>> dip = 90
>>> azimuth = 20
```

```
>>> # Calculating plunging vector
>>> vector = gg.visualization.calculate_vector(dip=dip, azimuth=azimuth)
>>> vector
array([[ 0.364824  ],
[-0.18285081],
[ 0.91294525]])
```

gemgis.visualization.clip_seismic_data

gemgis.visualization.**clip_seismic_data**(*seismic_data*, *cdp_start: Optional[int] = None*, *cdp_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Clipping seismic data loaded with segysak to CDP defined start and end CDP values

Parameters

- **seismic_data** (*xarray.core.dataset.Dataset*) – seismic data loaded with the segysak package
- **cdp_start** (*Union[int, type(None)]*) – Value for the start CDP number, e.g. cdp_start=100, default is 'None'
- **cdp_end** (*Union[int, type(None)]*) – Value for the end CDP number, e.g. cdp_start=200, default is 'None'

Returns **df_seismic_data_selection** – DataFrame containing the clipped seismic data

Return type pd.DataFrame

New in version 1.0.x.

gemgis.visualization.convert_to_rgb

gemgis.visualization.**convert_to_rgb**(*array: numpy.ndarray*) → numpy.ndarray

Converting array values to RGB values

Parameters **array** (*np.ndarray*) – Array containing the different bands of a raster

Returns **array_stacked** – Array with converted array values to RGB values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and showing predefined array
>>> import gemgis as gg
>>> import numpy as np
>>> array
array([[0.3647059 , 0.3647059 , 0.49411765],
[0.40784314, 0.40784314, 0.52156866],
[0.8901961 , 0.8901961 , 0.91764706],
...,
[0.59607846, 0.69803923, 0.8       ],
[0.627451  , 0.7372549 , 0.7882353 ],
[0.80784315, 0.78431374, 0.70980394]], dtype=float32)
```

```
>>> # Inspecting shape of array
>>> array.shape
(2000, 2800, 3)
```

```
>>> # Converting to RGB array
>>> array_stacked = gg.visualization.convert_to_rgb(array=array)
>>> array_stacked
array([[[ 93,  93, 126],
[104, 104, 133],
[227, 227, 234],
...,
[152, 178, 204],
[160, 188, 201],
[206, 200, 181]],
[[247, 246, 248],
[241, 240, 246],
[243, 241, 241],
...,
[150, 177, 205],
[175, 187, 177],
[232, 228, 219]]], dtype=uint8)
```

```
>>> # Inspecting shape of array
>>> array_stacked.shape
(2000, 2800, 3)
```

See also:

read_raster Reading Digital Elevation Model as xarray

drape_array_over_dem Draping an array of the Digital Elevation Model

gemgis.visualization.drape_array_over_dem

gemgis.visualization.**drape_array_over_dem**(*array: numpy.ndarray*, *dem: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *extent: List[Union[int, float]] = None*, *zmax: Union[float, int] = 10000*, *resize_array: bool = True*)

Creating grid and texture to drape array over a digital elevation model

Parameters

- **array** (*np.ndarray*) – Array containing the map data such as a WMS Map
- **dem** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Digital elevation model where the array data is being draped over
- **extent** (*List[Union[float, int]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **zmax** (*Union[float, int]*) – Maximum z value to limit the elevation data, e.g. `zmax=1000`
- **resize_array** (*bool*) – Whether to resize the array or the dem if the shape of the dem does not match the shape of the array Options include: `True` or `False`, default set to `True`

Returns

- **mesh** (*pyvista.core.pointset.PolyData*) – Mesh containing the Digital elevation model data
- **texture** (*pyvista.core.objects.Texture*) – PyVista Texture containing the map data

New in version 1.0.x.

Changed in version 1.1: Function now allows rasters with different sizes and resizes one of the rasters automatically

Changed in version 1.1.2: Edit zmax value and fixing a bug with the scikit-image resize function, see https://github.com/cgre-aachen/gemgis/issues/303

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> array
array([[[ 93,  93, 126],
[104, 104, 133],
[227, 227, 234],
...,
[152, 178, 204],
[160, 188, 201],
[206, 200, 181]],
[[247, 246, 248],
[241, 240, 246],
[243, 241, 241],
...,
[150, 177, 205],
[175, 187, 177],
[232, 228, 219]]], dtype=uint8)
```

```
>>> # Inspecting Digital Elevation Model values
>>> dem
array([[  0.  ,   0.  ,   0.  , ...,  40.1 ,  40.09,  44.58],
[  0.  ,   0.  ,   0.  , ...,  40.08,  40.07,  44.21],
[  0.  ,   0.  ,   0.  , ...,  40.14,  44.21,  43.98],
...,
[100.56, 102.14, 102.17, ...,   0.  ,   0.  ,   0.  ],
[ 99.44,  99.85,  99.77, ...,   0.  ,   0.  ,   0.  ],
[ 88.32,  91.76,  98.68, ...,   0.  ,   0.  ,   0.  ]],
dtype=float32)
```

```
>>> # Draping mesh over array
>>> mesh, texture = gg.visualization.drape_array_over_dem(array=array, dem=dem)
>>> mesh
Header
StructuredGrid  Information
N Cells         5595201
N Points        5600000
X Bounds        3.236e+07, 3.250e+07
Y Bounds        5.700e+06, 5.800e+06
Z Bounds        0.000e+00, 5.038e+02
Dimensions      2000, 2800, 1
N Arrays        1
Data Arrays
Name                Field   Type    N Comp  Min         Max
Texture Coordinates Points  float32 2       -7.077e-06  1.000e+00
```

```
>>> # Inspecting the texture
>>> texture
(Texture)00000151B91F3AC0
```

See also:

read_raster Reading Digital Elevation Model as xarray

convert_to_rgb Converting bands to RGB values for plotting

gemgis.visualization.get_batlow_cmap

gemgis.visualization.**get_batlow_cmap**() → matplotlib.colors.ListedColormap

Returning the Batlow cmap from https://github.com/callumrollo/cmcrameri/blob/master/cmcrameri/cmaps/batlow.txt

Returns **cmap_batlow** – Batlow color map

Return type matplotlib.colors.ListedColormap

New in version 1.0.x.

gemgis.visualization.get_color_lot

gemgis.visualization.**get_color_lot**(*geo_model*, *lith_c: pandas.core.frame.DataFrame = None*, *index='surface'*, *is_faults: bool = True*, *is_basement: bool = False*) → pandas.core.series.Series

Method to get the right color list depending on the type of plot. Borrowed from https://github.com/cgre-aachen/gempy/blob/6aed72a4dfa26830df142a0461294bd9d21a4fa4/gempy/plot/vista.py#L133-L167

Parameters

- **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model
- **lith_c** (*pd.DataFrame*) – Pandas Series with index surface names and values hex strings with the colors
- **index** (*str*) – Index provided as string, e.g. `index='surface'`, default is `'surface'`
- **is_faults** (*bool*) – Return the colors of the faults. This should be true for surfaces and input data and false for scalar values. Options include `True` and `False`, default is `True`
- **is_basement** (*bool*) – Return or not the basement. This should be true for the lith block and false for surfaces and input data. Options include `True` and `False`, default is `False`

New in version 1.0.x.

gemgis.visualization.get_mesh_geological_map

gemgis.visualization.**get_mesh_geological_map**(*geo_model*) → Tuple[pyvista.core.pointset.PolyData, matplotlib.colors.ListedColormap, bool]

Getting the geological map of a GemPy Model draped over the topography as mesh. Borrowed from https://github.com/cgre-aachen/gempy/blob/6aed72a4dfa26830df142a0461294bd9d21a4fa4/gempy/plot/vista.py#L512-L604

Parameters **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model

Returns

- **polydata** (*pv.core.PolyData*) – PyVista Mesh containing the geological map draped over the topography
- **cm** (*matplotlib.colors.ListedColormap*) – Colormap for plotting
- **rgb** (*bool*) – Boolean to use `rgb=True` when plotting

New in version 1.0.x.

gemgis.visualization.get_petrel_cmap

gemgis.visualization.**get_petrel_cmap**() → matplotlib.colors.ListedColormap

Returning the Petrel cmap

Returns **cmap_seismic** – Seismic color map

Return type matplotlib.colors.ListedColormap

New in version 1.0.x.

gemgis.visualization.get_points_along_spline

gemgis.visualization.**get_points_along_spline**(*spline: pyvista.core.pointset.PolyData*, *dist: numpy.ndarray*)

Returning the closest point on the spline a given a length along a spline.

Parameters

- **spline** (*pv.core.pointset.PolyData*) – Spline with the resampled vertices
- **dist** (*np.ndarray*) – np.ndarray containing the measured depths (MD) of values along the well path

Returns **spline.points[idx_list]** – PyVista Array containing the selected points

Return type pv.core.pyvista_ndarray.pyvista_ndarray

New in version 1.0.x.

gemgis.visualization.get_seismic_cmap

gemgis.visualization.**get_seismic_cmap**() → matplotlib.colors.ListedColormap

Returning the seismic cmap from https://github.com/lperozzi/Seismic_colormaps/blob/master/colormaps.py

Returns **cmap_seismic** – Seismic color map

Return type matplotlib.colors.ListedColormap

New in version 1.0.x.

gemgis.visualization.plane_through_hypocenters

gemgis.visualization.**plane_through_hypocenters**(*spheres: pyvista.core.composite.MultiBlock*) → pyvista.core.pointset.PolyData

Fitting a plane through the hypocenters of earthquakes using Eigenvector analysis

Parameters **spheres** (*pv.core.composite.MultiBlock*) – PyVista MultiBlock object containing the hypocenters stored as spheres

Returns **plane** – Plane fitting through the hypocenters using Eigenvector analysis

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> spheres = pv.read(filename='spheres.vtk')
```

```
>>> # Fitting plane through spheres
>>> plane = gg.visualization.plane_through_hypocenters(spheres=spheres)
>>> plane
Header
```

(continues on next page)

(continued from previous page)

```
PolyData    Information
N Cells     100
N Points    121
X Bounds    3.230e+07, 3.231e+07
Y Bounds    5.618e+06, 5.620e+06
Z Bounds    -1.113e+04, -8.471e+03
N Arrays    2
Data Arrays
Name                Field   Type    N Comp  Min         Max
Normals             Points  float32 3       0.000e+00   1.000e+00
TextureCoordinates  Points  float32 2       0.000e+00   1.000e+00
```

gemgis.visualization.plot_data

gemgis.visualization.**plot_data**(*geo_data*, *show_basemap: bool = False*, *show_geolmap: bool = False*, *show_topo: bool = False*, *show_interfaces: bool = False*, *show_orientations: bool = False*, *show_customsections: bool = False*, *show_wms: bool = False*, *show_legend: bool = True*, *show_hillshades: bool = False*, *show_slope: bool = False*, *show_aspect: bool = False*, *show_contours: bool = False*, *add_to_extent: float = 0*, *hide_topo_left: bool = False*, ***kwargs*)

Plotting Input Data

Parameters

- **geo_data** – GemPy Geo Data Class containing the raw data
- **show_basemap** (*bool*) – Showing the basemap. Options include `True` and `False`, default is `False`
- **show_geolmap** (*bool*) – Showing the geological map. Options include `True` and `False`, default is `False`
- **show_topo** (*bool*) – Showing the topography/digital elevation model. Options include `True` and `False`, default is `False`
- **show_interfaces** (*bool*) – Showing the interfaces. Options include `True` and `False`, default is `False`
- **show_orientations** (*bool*) – Showing orientations. Options include `True` and `False`, default is `False`
- **show_customsections** (*bool*) – Showing custom sections. Options include `True` and `False`, default is `False`
- **show_wms** (*bool*) – Showing a WMS layer. Options include `True` and `False`, default is `False`
- **show_legend** (*bool*) – Showing the legend of interfaces. Options include `True` and `False`, default is `False`
- **show_hillshades** (*bool*) – Showing hillshades. Options include `True` and `False`, default is `False`
- **show_slope** (*bool*) – Showing the slope of the DEM. Options include `True` and `False`, default is `False`

- **show_aspect** (*bool*) – Showing the aspect of the DEM. Options include `True` and `False`, default is `False`
- **show_contours** (*bool*) – Showing the contours of the DEM
- **add_to_extent** (*float*) – Number of meters to add to the extent of the plot in each direction, e.g. `add_to_extent=10`, default is `0`
- **hide_topo_left** (*bool*) – If set to `True`, the topography will not be shown in the left plot. Options include `True` and `False`, default is `False`
- **cmap_basemap** (*str*) – Cmap for basemap
- **cmap_geolmap** (*str*) – Cmap for geological map
- **cmap_topo** (*str*) – Cmap for topography
- **cmap_hillshades** (*str*) – Cmap for hillshades
- **cmap_slope** (*str*) – Cmap for slope
- **cmap_aspect** (*str*) – Cmap for aspect
- **cmap_interfaces** (*str*) – Cmap for interfaces
- **cmap_orientations** (*str*) – Cmap for orientations
- **cmap_wms** (*str*) – Cmap for WMS Service
- **cmap_contours** (*str*) – Cmap for contour lines

New in version 1.0.x.

gemgis.visualization.plot_orientations

gemgis.visualization.**plot_orientations**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, pandas.core.frame.DataFrame]*, *show_planes: bool = True*, *show_density_contours: bool = True*, *show_density_contourf: bool = False*, *formation: str = None*, *method: str = 'exponential_kamb'*, *sigma: Union[float, int] = 1*, *cmap: str = 'Blues_r'*)

Plotting orientation values of a GeoDataFrame with mplstereonet

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing columns with orientations values
- **show_planes** (*bool*) – Variable to show planes of orientation values. Options include: `True` or `False`, default set to `True`
- **show_density_contours** (*bool*) – Variable to display density contours. Options include: `True` or `False`, default set to `True`
- **show_density_contourf** (*bool*) – Variable to display density contourf. Options include: `True` or `False`, default set to `False`
- **formation** (*str*) – Name of the formation for which the contourf plot is shown, e.g. `formation='Layer1'`, default is `None`
- **method** (*str*) – Method to estimate the orientation density distribution, `method='exponential_kamb'`, default is `'exponential_kamb'`

- **sigma** (*Union[float, int]*) – Expected count in standard deviations, e.g. `sigma=1`, default is 1
- **cmap** (*str*) – Name of the colormap for plotting the orientation densities, e.g. `cmap='Blues_r'`, default is `'Blues_r'`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   dip     azimuth geometry
0   None    Sand        25.00   310     POINT (49.249 1033.893)
1   None    Sand        30.00   315     POINT (355.212 947.557)
2   None    Sand        15.00   330     POINT (720.248 880.912)
3   None    Clay        10.00   135     POINT (526.370 611.300)
4   None    Clay        25.00   140     POINT (497.591 876.368)
5   None    Clay        35.00   50      POINT (394.593 481.039)
```

```
>>> # Creating plot
>>> gg.visualization.plot_orientations(gdf=gdf, show_planes=True, show_density_
↪contours=False, show_density_contourf=False)
```

gemgis.visualization.polyline_from_points

gemgis.visualization.**polyline_from_points**(*points: numpy.ndarray*) → pyvista.core.pointset.PolyData

Creating PyVista PolyLine from points

Parameters **points** (*np.ndarray*) – Points defining the PolyLine

Returns **poly**

Return type pv.core.pointset.PolyData

New in version 1.0.x.

gemgis.visualization.read_raster

gemgis.visualization.**read_raster**(*path=<class 'str'>*, *nodata_val: typing.Union[float, int] = None*, *name: str = 'Elevation [m]'*) → pyvista.core.pointset.PolyData

Reading a raster and returning a mesh

Parameters

- **path** (*str*) – Path to the raster, e.g. `path='raster.tif'`
- **nodata_val** (*Union[float, int]*) – Nodata value of the raster, e.g. `nodata_val=9999.0`
- **name** (*str*) – Name of the data array, e.g. `name='Elevation [m]`, default is `'Elevation [m]'`

Returns **mesh** – PyVista mesh containing the raster values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Changed in version 1.1.1: Set nodata value manually if no data value is provided and raster does not contain nodata values

Example

```
>>> # Loading Libraries and outputting mesh
>>> import gemgis as gg
>>> polydata = gg.visualization.read_raster(path='raster.tif', nodata_val=9999.0,␣
↪name='Elevation [m]')
>>> polydata
Header
StructuredGrid  Information
N Cells         5595201
N Points        5600000
X Bounds        3.236e+07, 3.250e+07
Y Bounds        5.700e+06, 5.800e+06
Z Bounds        0.000e+00, 0.000e+00
Dimensions      2000, 2800, 1
N Arrays        1
Data Arrays
Name            Field   Type    N Comp  Min         Max
Elevation [m]   Points  float32 1       0.000e+00   5.038e+02
```

See also:

convert_to_rgb Converting bands to RGB values for plotting

drape_array_over_dem Draping an array of the Digital Elevation Model

gemgis.visualization.seismic_to_array

gemgis.visualization.**seismic_to_array**(*seismic_data*, *cdp_start: Optional[int] = None*, *cdp_end: Optional[int] = None*, *max_depth: Optional[Union[int, float]] = None*) → numpy.ndarray

Converting seismic data loaded with segysak to a NumPy array

Parameters

- **seismic_data** (*xarray.core.dataset.Dataset*) – seismic data loaded with the segysak package
- **cdp_start** (*Union[int, type(None)]*) – Value for the start CDP number, e.g. cdp_start=100, default is None
- **cdp_end** (*Union[int, type(None)]*) – Value for the end CDP number, e.g. cdp_start=200, default is None
- **max_depth** (*Union[int, float, type(None)]*) – Maximum depth of the seismic, e.g. max_depth=200, default is None

Returns **df_seismic_data_values_reshaped_selected** – NumPy array containing the seismic data

Return type np.ndarray

New in version 1.0.x.

gemgis.visualization.seismic_to_mesh

gemgis.visualization.**seismic_to_mesh**(*seismic_data*, *cdp_start: Optional[int] = None*, *cdp_end: Optional[int] = None*, *max_depth: Union[int, float] = None*, *sampling_rate: Optional[int] = None*, *shift: Union[int, float] = 0*, *source_crs: Union[str, pyproj.crs.crs.CRS] = None*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *cdp_coords=None*) → pyvista.core.pointset.StructuredGrid

Converting seismic data loaded with segysak to a PyVista Mesh

Parameters

- **seismic_data** (*xarray.core.dataset.Dataset*) – seismic data loaded with the segysak package
- **cdp_start** (*Union[int, type(None)]*) – Value for the start CDP number, e.g. `cdp_start=100`, default is `None`
- **cdp_end** (*Union[int, type(None)]*) – Value for the end CDP number, e.g. `cdp_start=200`, default is `None`
- **max_depth** (*Union[int, float, type(None)]*) – Maximum depth of the seismic, e.g. `max_depth=200`, default is `None`
- **sampling_rate** (*Union[int, type(None)]*) – Sampling rate of the seismic, e.g. `sampling_rate=2`, default is `None`
- **shift** (*Union[int, float]*) – Shift of the seismic, e.g. `shift=100`, default is `0`
- **source_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Source CRS of the seismic, e.g. `source_crs='EPSG:25832'`, default is `None`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Target CRS of the seismic, e.g. `target_crs='EPSG:3034'`, default is `None`
- **cdp_coords** (*Union[int, float, type(None)]*) – CDP coordinates of the seismic if no CDP columns are present in the segysak object, default is `None`

Returns **grid** – PyVista Structured grid containing the seismic data

Return type pv.core.pointset.StructuredGrid

New in version 1.0.x.

8.4 Utils

The following sections provide an overview of the methods implemented in the GemGIS Utils module.

8.4.1 Miscellaneous utils methods

The following methods are further visualization methods used in GemGIS.

gemgis.utils.assign_properties(lith_block, ...)	Replacing lith block IDs with physical properties
gemgis.utils.build_style_dict(classes)	Building a style dict based on extracted style classes
gemgis.utils.parse_categorized_qml(qml_name)	Parsing a QGIS style file to retrieve surface color values
gemgis.utils.load_surface_colors(path, gdf)	Loading surface colors from a QML file and storing the color values as list to be displayed with GeoPandas plots
gemgis.utils.create_surface_color_dict(path)	Creating GemPy surface color dict from a QML file
gemgis.utils.calculate_lines(gdf, increment)	Function to interpolate strike lines
gemgis.utils.calculate_number_of_isopoints(...)	Creating the number of isopoints to further interpolate strike lines
gemgis.utils.convert_location_dict_to_gdf(...)	Converting a location dict to a GeoDataFrame
gemgis.utils.convert_to_gempy_df(gdf[, dem, ...])	Converting a GeoDataFrame into a Pandas DataFrame ready to be read in for GemPy
gemgis.utils.convert_to_petrel_points_with_attributes(...)	Function to convert vertices of a PyVista Mesh to Petrel Points with Attributes
gemgis.utils.create_polygon_from_location(...)	Creating Shapely Polygon from bounding box coordinates
gemgis.utils.create_virtual_profile(...[, ...])	Function to filter and sort the resulting well tops
gemgis.utils.create_zmap_grid(surface, ...)	Function to write data to ZMAP Grid, This code is heavily inspired by https://github.com/abduhbm/zmapio
gemgis.utils.extract_zmap_data(surface, ...)	Function to extract a meshgrid of values from a PyVista mesh
gemgis.utils.get_location_coordinate(name)	Obtaining coordinates of a given city
gemgis.utils.get_locations(names[, crs])	Obtaining coordinates for one city or a list of given cities.
gemgis.utils.get_nearest_neighbor(x, y)	Function to return the index of the nearest neighbor for a given point Y
gemgis.utils.getfeatures(extent, crs_raster, ...)	Creating a list containing a dict with keys and values to clip a raster
gemgis.utils.interpolate_strike_lines(gdf, ...)	Interpolating strike lines to calculate orientations
gemgis.utils.ray_trace_multiple_surfaces(...)	Function to return the depth of multiple surfaces in one well using PyVista ray tracing
gemgis.utils.ray_trace_one_surface(surface, ...)	Function to return the depth of one surface in one well using PyVista ray tracing
gemgis.utils.read_csv_as_gdf(path, crs[, x, ...])	Reading CSV files as GeoDataFrame
gemgis.utils.save_zmap_grid(zmap_grid[, path])	Function to save ZMAP Grid information to file
gemgis.utils.set_extent([minx, maxx, miny, ...])	Setting the extent for a model
gemgis.utils.set_resolution(x, y, z)	Setting the resolution for a model
gemgis.utils.show_number_of_data_points(...)	Adding the number of Interfaces and Orientations to the GemPy Surface dataframe
gemgis.utils.to_section_dict(gdf[, ...])	Converting custom sections stored in Shape files to GemPy section_dicts
gemgis.utils.transform_location_coordinate(...)	Transforming coordinates of GeoPy Location

gemgis.utils.assign_properties

gemgis.utils.**assign_properties**(*lith_block: numpy.ndarray*, *property_dict: dict*) → numpy.ndarray

Replacing lith block IDs with physical properties

Parameters

- **lith_block** (*np.ndarray*) – GemPy lith block array containing the surface IDs
- **property_array** (*dict*) – Dict containing the property values mapped to a surface ID

Returns **property_block** – Array containing the properties

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and lith_block plus reshaping
>>> import gemgis as gg
>>> import numpy as np
>>> lith_block = np.load('lith_block.npy').reshape(50,50,50)
```

```
>>> # Defining properties
>>> density_values = [0.1, 2.5, 3.0]
```

```
>>> # Creating dict
>>> density_dict = {k: v for k,v in zip(np.unique(np.round(lith_block)), density_
↪values)}
>>> density_dict
{1.0: 0.1, 2.0: 2.5, 3.0: 3.0}
```

```
>>> # Assign properties
>>> property_block = gg.utils.assign_properties(lith_block=lith_block, property_
↪dict=property_dict)
```

gemgis.utils.build_style_dict

gemgis.utils.**build_style_dict**(*classes: dict*) → dict

Building a style dict based on extracted style classes

Parameters **classes** (*dict*) – Dict containing the styles of objects

Returns **styles** – Dict containing styles for different objects

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> column, classes = gg.utils.parse_categorized_qml(qml_name='style.qml')
>>>column
'formation'
```

```
>>> # Inspecting classes
>>> classes
{'Sand1': {'border_width_map_unit_scale': '3x:0,0,0,0,0,0',
'color': '179,90,42,255',
'joinstyle': 'bevel',
'offset': '0,0',
'offset_map_unit_scale': '3x:0,0,0,0,0,0',
'offset_unit': 'MM',
'outline_color': '102,51,24,255',
'outline_style': 'solid',
'outline_width': '0.26',
'outline_width_unit': 'MM',
'style': 'solid'},....}
```

```
>>> # Creating Style Dict
>>> style_dict = gg.utils.build_style_dict(classes=classes)
>>> style_dict
{'Sand1': {'color': '#b35a2a',
'color_rgb': [179, 90, 42],
'opacity': 1.0,
'weight': 0.26,
'fillColor': '#b35a2a',
'fillOpacity': 1.0},...}
```

See also:

parse_categorized_qml Reading the contents of a QGIS Style file (qml)

load_surface_colors Loading surface colors as list

create_surface_color_dict Creating dict with colors for each formation

gemgis.utils.parse_categorized_qml

gemgis.utils.**parse_categorized_qml**(*qml_name: str*) → tuple

Parsing a QGIS style file to retrieve surface color values

Parameters **qml_name** (*str*) – Path to the QML file, e.g. `qml_name='style.qml'`

Returns

- **column** (*str*) – Variable indicating after which formation the objects were colored (i.e. `'formation'`)
- **classes** (*dict*) – Dict containing the style attributes for all available objects

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> column, classes = gg.utils.parse_categorized_qml(qml_name='style.qml')
>>> column
'formation'
```

```
>>> # Inspecting classes
>>> classes
{'Sand1': {'border_width_map_unit_scale': '3x:0,0,0,0,0,0',
'color': '179,90,42,255',
'joinstyle': 'bevel',
'offset': '0,0',
'offset_map_unit_scale': '3x:0,0,0,0,0,0',
'offset_unit': 'MM',
'outline_color': '102,51,24,255',
'outline_style': 'solid',
'outline_width': '0.26',
'outline_width_unit': 'MM',
'style': 'solid'},....}
```

See also:

build_style_dict Building style dictionairy from loaded style file

load_surface_colors Loading surface colors as list

create_surface_color_dict Creating dict with colors for each formation

gemgis.utils.load_surface_colors

gemgis.utils.**load_surface_colors**(*path: str*, *gdf: geopandas.geodataframe.GeoDataFrame*) → List[str]

Loading surface colors from a QML file and storing the color values as list to be displayed with GeoPandas plots

Parameters

- **path** (*str*) – Path to the qml file, e.g. `qml_name='style.qml'`
- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame of which objects are supposed to be plotted, usually loaded from a polygon/line shape file

Returns **cols** – List of color values for each surface

Return type List[str]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
```

```
>>> # Loading surface colors
>>> colors = gg.utils.load_surface_colors(path='style.qml', gdf=gdf)
>>> colors
['#b35a2a', '#b35a2a', '#525252']
```

See also:

build_style_dict Building style dictionairy from loaded style file

parse_categorized_qml Reading the contents of a QGIS Style file (qml)

create_surface_color_dict Creating dict with colors for each formation

gemgis.utils.create_surface_color_dict

gemgis.utils.**create_surface_color_dict**(*path: str*) → dict

Creating GemPy surface color dict from a QML file

Parameters **path** (*str*) – Path to the qml file, e.g. `qml_name='style.qml'`

Returns **surface_color_dict** – Dict containing the surface color values for GemPy

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> surface_colors_dict = gg.utils.create_surface_color_dict(path='style.qml')
>>> surface_colors_dict
{'Sand1': '#b35a2a', 'Sand2': '#b35a2a', 'Ton': '#525252'}
```

See also:

build_style_dict Building style dictionairy from loaded style file

parse_categorized_qml Reading the contents of a QGIS Style file (qml)

load_surface_colors Loading surface colors as list

gemgis.utils.calculate_lines

gemgis.utils.**calculate_lines**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, pandas.core.frame.DataFrame]*, *increment: Union[float, int]*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*) → geopandas.geodataframe.GeoDataFrame

Function to interpolate strike lines

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing existing strike lines
- **increment** (*Union[float, int]*) – Increment between the strike lines, e.g. `increment=50`
- **xcol** (*str*) – Name of X column, e.g. `x='X'`
- **ycol** (*str*) – Name of X column, e.g. `y='Y'`
- **zcol** (*str*) – Name of Z column, e.g. `z='Z'`

Returns **lines** – GeoDataFrame with interpolated strike lines

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='lines5_strike.shp')
>>> gdf
    id  Z   formation   geometry
0   7   0   Coal1   LINESTRING (1642.839 2582.579, 2829.348 2205.937)
1   6   150 Coal1   LINESTRING (1705.332 1759.201, 2875.795 1406.768)
2   5   200 Coal1   LINESTRING (1017.766 1722.234, 2979.938 1137.003)
3   4   250 Coal1   LINESTRING (99.956 1763.424, 765.837 1620.705,...
4   3   200 Coal1   LINESTRING (1078.147 1313.501, 2963.048 752.760)
```

```
>>> gdf_interpolated = gg.utils.calculate_lines(gdf=gdf, increment=50)
```

gemgis.utils.calculate_number_of_isopoints

gemgis.utils.**calculate_number_of_isopoints**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, pandas.core.frame.DataFrame]*, *increment: Union[float, int]*, *zcol: str = 'Z'*) → int

Creating the number of isopoints to further interpolate strike lines

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing existing strike lines
- **increment** (*Union[float, int]*) – Increment between the strike lines, e.g. `increment=50`

- **zcol** (*string*) – Name of z column, e.g. z='Z', default is 'Z'

Returns **number** – Number of isopoints

Return type int

New in version 1.0.x.

Example

```
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='lines5_strike.shp')
>>> gdf
    id  Z   formation   geometry
0   7   0   Coal1   LINESTRING (1642.839 2582.579, 2829.348 2205.937)
1   6   150 Coal1   LINESTRING (1705.332 1759.201, 2875.795 1406.768)
2   5   200 Coal1   LINESTRING (1017.766 1722.234, 2979.938 1137.003)
3   4   250 Coal1   LINESTRING (99.956 1763.424, 765.837 1620.705,...
4   3   200 Coal1   LINESTRING (1078.147 1313.501, 2963.048 752.760)
```

```
>>> number = gg.utils.calculate_number_of_isopoints(gdf=gdf, increment=50)
>>> number
2
```

See also:

get_nearest_neighbor Getting the nearest neighbor to a point

gemgis.utils.convert_location_dict_to_gdf

gemgis.utils.**convert_location_dict_to_gdf**(*location_dict: dict*) → geopandas.geodataframe.GeoDataFrame

Converting a location dict to a GeoDataFrame

Parameters **location_dict** (*dict*) – Dict containing the name of the location and the coordinates

Returns **gdf** – GeoDataFrame containing the location name and the coordinates of the location

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Creating a dict with coordinates
>>> coordinates_dict = gg.utils.get_locations(names = ['Aachen', 'Berlin', 'München
↪', 'Hamburg', 'Köln'], crs='EPSG:4647')
```

```
>>> # Converting dict to GeoDataFrame
>>> gdf = gg.utils.convert_location_dict_to_gdf(location_dict=coordinates_dict)
>>> gdf
    City    X           Y           geometry
0       Aachen  32294411.33 5629009.36  POINT (32294411.335 5629009.357)
1       Berlin  32797738.56 5827603.74  POINT (32797738.561 5827603.740)
2       München 32691595.36 5334747.27  POINT (32691595.356 5334747.274)
3       Hamburg 32566296.25 5933959.96  POINT (32566296.251 5933959.965)
4       Köln    32356668.82 5644952.10  POINT (32356668.818 5644952.100)
```

gemgis.utils.convert_to_gempy_df

gemgis.utils.**convert_to_gempy_df**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: Union[rasterio.io.DatasetReader, numpy.ndarray] = None*, *extent: List[Union[int, float]] = None*) → pandas.core.frame.DataFrame

Converting a GeoDataFrame into a Pandas DataFrame ready to be read in for GemPy

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing spatial information, formation names and orientation values
- **dem** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy ndarray or rasterio object containing the height values
- **extent** (*List[Union[float,int]*) – List containing the extent of the np.ndarray, must be provided in the same CRS as the gdf, e.g. `extent=[0, 972, 0, 1069]`

Returns **df** – Interface or orientations DataFrame ready to be read in for GemPy

Return type pd.DataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0   None    Ton         POINT (19.150 293.313)
1   None    Ton         POINT (61.934 381.459)
2   None    Ton         POINT (109.358 480.946)
3   None    Ton         POINT (157.812 615.999)
4   None    Ton         POINT (191.318 719.094)
```

```
>>> # Loading Digital Elevation Model
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Defining extent
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Converting GeoDataFrame to DataFrame
>>> df = gg.utils.convert_to_gempy_df(gdf=gdf, dem=dem, extent=extent)
>>> df
    formation   X       Y       Z
0   Ton         19.15   293.31  364.99
1   Ton         61.93   381.46  400.34
2   Ton         109.36  480.95  459.55
3   Ton         157.81  616.00  525.69
4   Ton         191.32  719.09  597.63
```

gemgis.utils.convert_to_petrel_points_with_attributes

gemgis.utils.**convert_to_petrel_points_with_attributes**(*mesh: pyvista.core.pointset.PolyData*, *path: str*, *crs: Optional[Union[str, pyproj.crs.crs.CRS]] = None*, *target_crs: Optional[Union[str, pyproj.crs.crs.CRS]] = None*)

Function to convert vertices of a PyVista Mesh to Petrel Points with Attributes

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista Mesh to be converted to points
- **path** (*str*) – Path to store the converted points, e.g. `path='project/'`
- **crs** (*str, pyproj.crs.crs.CRS, type(None)*) – Coordinate reference system for the GeoDataFrame, e.g. `crs='EPSG:4326'`, default is `None`
- **target_crs** (*str, pyproj.crs.crs.CRS, type(None)*) – Target coordinate reference system if coordinates of points should be reprojected, e.g. `crs='EPSG:4326'`, default is `None`

New in version 1.0.x.

gemgis.utils.create_polygon_from_location

gemgis.utils.**create_polygon_from_location**(*coordinates*) → shapely.geometry.polygon.Polygon

Creating Shapely Polygon from bounding box coordinates

Parameters **coordinates** (*geopy.location.Location*) – GeoPy location object

Returns **polygon** – Shapely Polygon marking the bounding box of the coordinate object

Return type shapely.geometry.polygon.Polygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries and get location object
>>> import gemgis as gg
>>> location = gg.utils.get_location_coordinate(name='Aachen')
>>> location
Location(Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland, (50.776351,␣
↪6.083862, 0.0))
```

```
>>> # Creating polygon from location bounds
>>> polygon = gg.utils.create_polygon_from_location(coordinates=location)
>>> polygon.wkt
'POLYGON ((50.8572449 5.9748624, 50.8572449 6.2180747, 50.6621373 6.2180747, 50.
↪6621373 5.9748624, 50.8572449 5.9748624))'
```

See also:

transform_location_coordinate Transforming location coordinate to another CRS

get_location_coordinate Get GeoPy Location Object

get_locations Get location information for a list of city names

gemgis.utils.create_virtual_profile

gemgis.utils.**create_virtual_profile**(*names_surfaces: list*, *surfaces: list*, *borehole: pyvista.core.pointset.PolyData*, *first_point: bool = False*) → pandas.core.frame.DataFrame

Function to filter and sort the resulting well tops

Parameters

- **names_surfaces** (*list*) – List of the names of the calculated GemPy surfaces, e.g. `names_surfaces=['Layer1', 'Layer2']`
- **surfaces** (*list*) – List of calculated GemPy surfaces, e.g. `surfaces=['Layer1', 'Layer2']`
- **borehole** (*pv.core.pointset.PolyData*) – Coordinates of the bottom of the well
- **first_point** (*bool*) – Returns intersection of first point only. Options include: `True` or `False`, default set to `False`

Returns **df** – DataFrame containing the well tops

Return type pd.DataFrame

New in version 1.0.x.

gemgis.utils.create_zmap_grid

gemgis.utils.**create_zmap_grid**(*surface: pyvista.core.pointset.PolyData*, *cell_width: int*, *comments: str = ''*, *name: str = 'ZMAP_Grid'*, *z_type: str = 'GRID'*, *nodes_per_line: int = 5*, *field_width: int = 15*, *nodata: Union[int, float] = - 9999.0*, *nodata2: Union[int, float, str] = ''*, *decimal_places: int = 5*, *start_column: int = 1*)

Function to write data to ZMAP Grid, This code is heavily inspired by https://github.com/abduhbm/zmapio

Parameters

- **surface** (*pv.core.pointset.PolyData*) – PyVista mesh
- **cell_width** (*int*) – Width of grid cell, e.g. `cell_width=50`
- **comments** (*str*) – Comments written to the ZMAP File, e.g. `comments='Project: Einstein'`, default is `''`
- **name** (*str*) – Name of the ZMAP File, e.g. `name='ZMAP_Grid'`, default is `'ZMAP_Grid'`
- **z_type** (*str*) – ZMAP Grid Type, e.g. `z_type='GRID'`, default is `'GRID'`
- **nodes_per_lines** (*int*) – Number of values per line, e.g. `nodes_per_line=5`, default is `5`
- **field_width** (*int*) – Width of each field, e.g. `field_width=15`, default is `15`
- **nodata** (*Union[int, float]*) – No data value, e.g. `nodata=-9999`, default is `-9999`
- **nodata2** (*Union[int, float, str]*) – No data value, e.g. `nodata2=-9999`, default is `''`
- **decimal_places** (*int*) – Number of Decimal Places, e.g. `decimal_places=5`, default is `5`
- **start_column** (*int*) – Number of the start column, e.g. `start_column=1`, default is `1`

Returns **lines** – String containing the ZMAP Grid Data

Return type str

New in version 1.0.x.

gemgis.utils.extract_zmap_data

gemgis.utils.**extract_zmap_data**(*surface: pyvista.core.pointset.PolyData*, *cell_width: int*, *nodata: Union[float, int] = - 9999*)

Function to extract a meshgrid of values from a PyVista mesh

Parameters

- **surface** (*pv.core.pointset.PolyData*) – PyVista mesh
- **cell_width** (*int*) – Width of grid cell, e.g. `cell_width=50`
- **nodata** (*Union[float, int]*) – No data value, e.g. `nodata=-9999`, default is `-9999`

New in version 1.0.x.

gemgis.utils.get_location_coordinate

gemgis.utils.**get_location_coordinate**(*name: str*)

Obtaining coordinates of a given city

Parameters **name** (*str*) – Name of the location, e.g. name='Aachen'

Returns **coordinates** – GeoPy Location object

Return type geopy.location.Location

New in version 1.0.x.

Example

```
>>> # Loading Libraries and get location object
>>> import gemgis as gg
>>> location = gg.utils.get_location_coordinate(name='Aachen')
>>> location
Location(Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland, (50.776351,␣
↪6.083862, 0.0))
```

See also:

transform_location_coordinate Transforming location coordinate to another CRS

create_polygon_from_location Create Shapely Polygon from GeoPy Location Object bounds

get_locations Get location information for a list of city names

gemgis.utils.get_locations

gemgis.utils.**get_locations**(*names: Union[list, str]*, *crs: Union[str, pyproj.crs.crs.CRS] = 'EPSG:4326'*) → dict

Obtaining coordinates for one city or a list of given cities. A CRS other than 'EPSG:4326' can be passed to transform the coordinates

Parameters

- **names** (*Union[list, str]*) – List of cities or single city name, e.g. names=['Aachen', 'Cologne', 'Munich', 'Berlin']
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – CRS that coordinates will be transformed to, e.g. crs='EPSG:4647', default is the GeoPy crs 'EPSG:4326'

Returns **location_dict** – Dict containing the addresses and coordinates of the selected cities

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and get location objects
>>> import gemgis as gg
>>> names = ['Aachen', 'Cologne', 'Munich', 'Berlin']
>>> location_dict = gg.utils.get_locations(names=names, crs='EPSG:4647')
>>> location_dict
{'Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland': (32294411.
↪33488576,  5629009.357074926),
'Köln, Nordrhein-Westfalen, Deutschland': (32356668.818424627, 5644952.099932303),
'München, Bayern, Deutschland': (32691595.356409974, 5334747.274305081),
'Berlin, 10117, Deutschland': (32797738.56053437, 5827603.740024588)}
```

See also:

transform_location_coordinate Transforming location coordinate to another CRS

get_location_coordinate Get GeoPy Location Object

create_polygon_from_location Create Shapely Polygon from GeoPy Location Object bounds

gemgis.utils.get_nearest_neighbor

gemgis.utils.**get_nearest_neighbor**(*x: numpy.ndarray*, *y: numpy.ndarray*) → numpy.int64

Function to return the index of the nearest neighbor for a given point Y

Parameters

- **x** (*np.ndarray*) – Array with coordinates of a set of points, e.g. `x=np.array([(0,0), (10,10)])`
- **y** (*np.ndarray*) – Array with coordinates for point y, e.g. `y=np.array([2,2])`

Returns **index** – Index of the nearest neighbor of point set X to point Y

Return type np.int64

New in version 1.0.x.

Example

```
>>> import gemgis as gg
>>> import numpy as np
>>> x = np.array([(0,0), (10,10)])
>>> x
array([[ 0,  0],
       [10, 10]])
```

```
>>> y = np.array([2,2])
>>> y
array([2, 2])
```

```
>>> index = gg.utils.get_nearest_neighbor(x=x, y=y)
>>> index
0
```

See also:

calculate_number_of_isopoints Calculating the number of isopoints that are necessary to interpolate lines

gemgis.utils.getfeatures

gemgis.utils.**getfeatures**(*extent: Optional[List[Union[int, float]]]*, *crs_raster: Union[str, dict]*, *crs_bbox: Union[str, dict]*, *bbox: shapely.geometry.polygon.Polygon = None*) → list

Creating a list containing a dict with keys and values to clip a raster

Parameters

- **extent** (*Union[List[Union[int, float]]*) – List of bounds (minx,maxx, miny, maxy), e.g. `extent=[0, 972, 0, 1069]`
- **crs_raster** (*Union[str, dict]*) – String or dict containing the raster crs, e.g. `crs='EPSG:4647'`
- **crs_bbox** (*Union[str, dict]*) – String or dict containing the bbox crs, e.g. `crs='EPSG:4647'`
- **bbox** (*shapely.geometry.polygon.Polygon*) – Shapely polygon defining the bbox used to get the coordinates, , e.g. `polygon = Polygon([(0, 0), (0, 10), (10, 10), (10, 0)])`

Returns **data** – List containing a dict with keys and values to clip raster

Return type list

New in version 1.0.x.

gemgis.utils.interpolate_strike_lines

gemgis.utils.**interpolate_strike_lines**(*gdf: geopandas.geodataframe.GeoDataFrame*, *increment: Union[float, int]*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*) → geopandas.geodataframe.GeoDataFrame

Interpolating strike lines to calculate orientations

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing existing strike lines
- **increment** (*Union[float, int]*) – Increment between the strike lines, e.g. `increment=50`
- **xcol** (*str*) – Name of X column, e.g. `x='X'`
- **ycol** (*str*) – Name of X column, e.g. `y='Y'`
- **zcol** (*str*) – Name of Z column, e.g. `z='Z'`

Returns **gdf_out** – GeoDataFrame containing the existing and interpolated strike lines

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

gemgis.utils.ray_trace_multiple_surfaces

gemgis.utils.**ray_trace_multiple_surfaces**(*surfaces: list*, *borehole_top: Union[numpy.ndarray, list]*, *borehole_bottom: Union[numpy.ndarray, list]*, *first_point: bool = False*) → list

Function to return the depth of multiple surfaces in one well using PyVista ray tracing

Parameters

- **surfaces** (*list*) – List of calculated GemPy surfaces
- **borehole_top** – Coordinates of the top of the well
- **borehole_bottom** – Coordinates of the bottom of the well
- **first_point** (*bool*) – Returns intersection of first point only

Returns **intersections** – List of intersections

Return type list

New in version 1.0.x.

gemgis.utils.ray_trace_one_surface

gemgis.utils.**ray_trace_one_surface**(*surface: Union[pyvista.core.pointset.PolyData, pyvista.core.pointset.UnstructuredGrid]*, *origin: Union[numpy.ndarray, list]*, *end_point: Union[numpy.ndarray, list]*, *first_point: bool = False*) → tuple

Function to return the depth of one surface in one well using PyVista ray tracing

Parameters

- **surface** (*Union[pv.core.pointset.PolyData, pv.core.pointset.UnstructuredGrid]*) – Calculated or clipped GemPy surface
- **origin** – Coordinates of the top of the well
- **end_point** – Coordinates of the bottom of the well
- **first_point** (*bool*) – Returns intersection of first point only

Returns **intersection_points, intersection_cells** – Location of the intersection points, Indices of the intersection cells

Return type tuple

New in version 1.0.x.

gemgis.utils.read_csv_as_gdf

gemgis.utils.**read_csv_as_gdf**(*path: str*, *crs: Union[str, pyproj.crs.crs.CRS]*, *x: str = 'X'*, *y: str = 'Y'*, *z: str = None*, *delimiter: str = ','*) → geopandas.geodataframe.GeoDataFrame

Reading CSV files as GeoDataFrame

Parameters

- **path** (*str*) – Path of the CSV files, e.g. `path='file.csv'`
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – CRS of the spatial data, e.g. `crs='EPSG:4647'`

- **x** (*str*) – Name of the X column, e.g. `x='X'`, default is `'X'`
- **y** (*str*) – Name of the Y column, e.g. `y='Y'`, default is `'Y'`
- **z** (*str*) – Name of the Z column, e.g. `z='Z'`, default is `'Z'`
- **delimiter** (*str*) – Delimiter of CSV file, e.g. `delimiter=','`, default is ‘,’

Returns **gdf** – GeoDataFrame of the CSV data

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File as GeoDataFrame
>>> import gemgis as gg
>>> gdf = gg.utils.read_csv_as_gdf(path='file.csv')
>>> gdf
    id      formation   geometry
0   None    Ton         POINT (19.150 293.313)
1   None    Ton         POINT (61.934 381.459)
2   None    Ton         POINT (109.358 480.946)
3   None    Ton         POINT (157.812 615.999)
4   None    Ton         POINT (191.318 719.094)
```

gemgis.utils.save_zmap_grid

gemgis.utils.**save_zmap_grid**(*zmap_grid: list*, *path: str = 'ZMAP_Grid.dat'*)

Function to save ZMAP Grid information to file

Parameters

- **zmap_grid** (*list*) – List of strings containing the ZMAP Data
- **path** (*str*) – Path and filename to store the ZMAP Grid, e.g. `path='ZMAP_Grid.dat'`, default is `'ZMAP_Grid.dat'`

New in version 1.0.x.

gemgis.utils.set_extent

gemgis.utils.**set_extent**(*minx: Union[int, float] = 0*, *maxx: Union[int, float] = 0*, *miny: Union[int, float] = 0*, *maxy: Union[int, float] = 0*, *minz: Union[int, float] = 0*, *maxz: Union[int, float] = 0*, *gdf: geopandas.geodataframe.GeoDataFrame = None*) → List[Union[int, float]]

Setting the extent for a model

Parameters

- **minx** (*Union[int, float]*) – Value defining the left border of the model, e.g. `minx=0`, default is `0`
- **maxx** (*Union[int, float]*) – Value defining the right border of the model, e.g. `max=972`, default is `0`
- **miny** (*Union[int, float]*) – Value defining the upper border of the model, e.g. `miny=0`, default is `0`

- **maxy** (*Union[int, float]*) – Value defining the lower border of the model, e.g. maxy=1069, default is 0
- **minz** (*Union[int, float]*) – Value defining the top border of the model, e.g. minz=0, default is 0
- **maxz** (*Union[int, float]*) – Value defining the bottom border of the model, e.g. maxz=1000, default is 0
- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame from which bounds the extent will be set, default is None

Returns **extent** – List containing extent values

Return type List[Union[int, float]]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and setting the extent
>>> import gemgis as gg
>>> extent = gg.utils.set_extent(minx=0, maxx=972, miny=0, maxy=1069, minz=0,␣
↪maxz=1000)
>>> extent
[0, 972, 0, 1069, 0, 1000]
```

gemgis.utils.set_resolution

gemgis.utils.**set_resolution**(*x: int*, *y: int*, *z: int*) → List[int]

Setting the resolution for a model

Parameters

- **x** (*int*) – Value defining the resolution in X direction, e.g. x=50
- **y** (*int*) – Value defining the resolution in Y direction, e.g. y=50
- **z** (*int*) – Value defining the resolution in Z direction, e.g. z=50

Returns **resolution** – List containing resolution values

Return type List[int]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and setting the resolution
>>> import gemgis as gg
>>> res = gg.utils.set_resolution(x=50, y=50, z=50)
>>> res
[50, 50, 50]
```

gemgis.utils.show_number_of_data_points

gemgis.utils.**show_number_of_data_points**(*geo_model*)

Adding the number of Interfaces and Orientations to the GemPy Surface dataframe

Parameters **geo_model** (*gp.core.model.Project*) – GemPy geo_model object

Returns **geo_model.surfaces** – DataFrame-like object containing surface information and number of data points

Return type gempy.core.model.RestrictingWrapper

New in version 1.0.x.

Example

```
>>> # Loading Libraries and displaying surface DataFrame of a GemPy geo_model
>>> import gemgis as gg
>>> geo_model.surfaces
    surface     series          order_surfaces  color   id
0   Sand1       Strat_Series    1               #015482 1
1   Ton         Strat_Series    2               #9f0052 2
2   basement    Strat_Series    3               #ffbe00 3
```

```
>>> # Adding number of data points to DataFrame
>>> gg.utils.show_number_of_data_points(geo_model=geo_model)
    surface     series          order_surfaces  color   id  No. of Interfaces   No.␣
↪of Orientations
0   Sand1       Strat_Series    1               #015482 1   95                  0
1   Ton         Strat_Series    2               #9f0052 2   36                  8
2   basement    Strat_Series    3               #ffbe00 3   0                   0
```

gemgis.utils.to_section_dict

gemgis.utils.**to_section_dict**(*gdf: geopandas.geodataframe.GeoDataFrame*, *section_column: str = 'section_name'*, *resolution: List[int] = None*) → dict

Converting custom sections stored in Shape files to GemPy section_dicts

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points or lines of custom sections
- **section_column** (*str*) – String containing the name of the column containing the section names, e.g. `section_column='section_name'`, default is `'section_name'`
- **List[int]** (*resolution* –) – List containing the x,y resolution of the custom section, e.g. `resolution=[80,80]`

Returns **section_dict** – Dict containing the section names, coordinates and resolution

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      geometry                    Section
0       None    POINT (695.467 3.226)       Section1
1       None    POINT (669.284 1060.822)        Section1
```

```
>>> # Creating Section dict
>>> section_dict = gg.utils.to_section_dict(gdf=gdf, section_column='Section')
>>> section_dict
{'Section1': ([695.4667461080886, 3.2262250771374283],
[669.2840030245482, 1060.822026058724], [100, 80])}
```

gemgis.utils.transform_location_coordinate

gemgis.utils.**transform_location_coordinate**(*coordinates*, *crs: Union[str, pyproj.crs.crs.CRS]*) → dict

Transforming coordinates of GeoPy Location

Parameters

- **coordinates** (*geopy.location.Location*) – GeoPy location object
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the target crs, e.g. `crs='EPSG:4647'`

Returns **result_dict** – Dict containing the location address and transformed coordinates

Return type dict

New in version 1.0.x.

Changed in version 1.1.7.

Updated to use the latest pyproj transformer

Example

```
>>> # Loading Libraries and get location object
>>> import gemgis as gg
>>> location = gg.utils.get_location_coordinate(name='Aachen')
>>> location
Location(Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland, (50.776351,␣
↪6.083862, 0.0))
```

```
>>> # Transforming location coordinates
>>> result_dict = gg.utils.transform_location_coordinate(coordinates=location, crs=
↪'EPSG:4647')
>>> result_dict
{'Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland': (32294411.
↪33488576, 5629009.357074926)}
```

See also:

get_location_coordinate Get GeoPy Location Object

create_polygon_from_location Create Shapely Polygon from GeoPy Location Object bounds

get_locations Get location information for a list of city names

8.5 Web

The following sections provide an overview of the methods implemented in the GemGIS Web module.

gemgis.web.create_request(wcs_url, version, ...)	Creating URL to request data from WCS Server
gemgis.web.load_as_array(url, layer, style, ...)	Loading a portion of a WMS as array
gemgis.web.load_as_file(url, path[, ...])	Executing WCS request and downloading file into specified folder
gemgis.web.load_as_files(wcs_url, version, ...)	Executing WCS requests and downloading files into specified folder
gemgis.web.load_as_gpd(url[, typename, ...])	Requesting data from a WFS Service
gemgis.web.load_as_map(url, layer, style, ...)	Loading a portion of a WMS as array
gemgis.web.load_wcs(url)	Loading Web Coverage Service
gemgis.web.load_wfs(url)	Loading a WFS Service by URL
gemgis.web.load_wms(url)	Loading a WMS Service by URL

8.5.1 gemgis.web.create_request

gemgis.web.**create_request**(*wcs_url: str*, *version: str*, *identifier: str*, *form: str*, *extent: List[Union[int, float]]*, *name: str = 'test.tif'*) → str

Creating URL to request data from WCS Server

Parameters

- **wcs_url** (*str*) – Url of the WCS server, e.g. `url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'`
- **version** (*str*) – Version number of the WCS as string, e.g. `version='2.0.1'`
- **identifier** (*str*) – Name of the layer, e.g. `identifier='nw_dgm'`
- **form** (*str*) – Format of the layer, e.g. `form='image/tiff'`
- **extent** (*List[Union[float,int]]*) – Extent of the tile to be downloaded, size may be restricted by server, e.g. `extent=[0, 972, 0, 1069]`
- **name** (*str*) – Name of file, e.g. `name='tile1.tif'`, default is `'test.tif'`

Returns **url** – Url for the WCS request

Return type str

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

```
>>> # Creating Request for WCS Service
>>> url = gg.web.create_request(url=wcs.url, version=wcs.version, identifier='nw_dgm
↪', form='image/tiff', extent=[0, 1000, 0, 1000], name='test.tif'])
```

See also:

load_wcs Load WCS Service

load_as_file Download WCS data file

load_as_files Download WCS data files

8.5.2 gemgis.web.load_as_array

gemgis.web.**load_as_array**(*url: str*, *layer: str*, *style: str*, *crs: Union[str, dict]*, *bbox: List[Union[int, float]]*, *size: List[int]*, *filetype: str*, *transparent: bool = True*, *save_image: bool = False*, *path: str = None*, *overwrite_file: bool = False*, *create_directory: bool = False*) → numpy.ndarray

Loading a portion of a WMS as array

Parameters

- **url** (*str*) – Link of the WMS Service, e.g. `url='https://ows.terrestris.de/osm/service?'`
- **layer** (*str*) – Name of layer to be requested, e.g. `layer='OSM-WMS'`
- **style** (*str*) – Name of style of the layer, e.g. `style='default'`
- **crs** (*str*) – String or dict containing the CRS, e.g. `crs='EPSG:4647'`
- **bbox** (*List[Union[float,int]]*) – List of bounding box coordinates, e.g. `bbox=[0, 972, 0, 1069]`
- **size** (*List[int]*) – List of x and y values defining the size of the image, e.g. `size=[1000, 1000]`
- **filetype** (*str*) – String of the image type to be downloaded, e.g. 'filetype='image/png'```
- **transparent** (*bool*) – Variable if layer is transparent. Options include: `True` or `False`, default set to `True`
- **save_image** (*bool*) – Variable to save image. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path and file name of the file to be saved, e.g. `path=map.tif`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **wms_array** – OWSlib map object loaded as np.ndarray

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WMS Service as array
>>> import gemgis as gg
>>> wms_map = gg.web.load_as_array(url='https://ows.terrestris.de/osm/service?',␣
↪layer='OSM-WMS', style='default', crs='EPSG:4647', bbox=[32286000,32328000,␣
↪5620000,5648000], size=[4200, 2800], filetype='image/png')
>>> wms_map
array([[[0.8039216 , 0.7647059 , 0.65882355],
[0.85882354, 0.8784314 , 0.6627451 ],
[0.87058824, 0.91764706, 0.6666667 ],
...,
[0.78431374, 0.7647059 , 0.65882355],
[0.8862745 , 0.9019608 , 0.81960785],
[0.9529412 , 0.93333334, 0.9019608 ]]], dtype=float32)
```

See also:

load_wms Load WMS Service

load_as_map Load Map from WMS Service

8.5.3 gemgis.web.load_as_file

gemgis.web.**load_as_file**(*url: str*, *path: str*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Executing WCS request and downloading file into specified folder

Parameters

- **url** (*str*) – Url for request
- **path** (*str*) – Path where file is saved, e.g. `path='tile.tif'`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

```
>>> # Creating Request for WCS Service
>>> url = gg.web.create_request(url=wcs.url, version=wcs.version, identifier='nw_dgm
↪', form='image/tiff', extent=[0, 1000, 0, 1000], name='test.tif'])
```

```
>>> # Downloading file from WCS Service
>>> gg.web.load_as_file(url=url, path='tile.tif')
```

See also:

load_wcs Load WCS Service

create_request Create request for WCS

load_as_files Download WCS data files

8.5.4 gemgis.web.load_as_files

gemgis.web.**load_as_files**(*wcs_url: str*, *version: str*, *identifier: str*, *form: str*, *extent: List[Union[int, float]]*, *size: int*, *path: str = ''*, *create_directory: bool = False*)

Executing WCS requests and downloading files into specified folder

Parameters

- **wcs_url** (*str*) – Url of the WCS server, e.g. `url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'`
- **version** (*str*) – Version number of the WCS as string, e.g. `version='2.0.1'`
- **identifier** (*str*) – Name of the layer, e.g. `identifier='nw_dgm'`
- **form** (*str*) – Format of the layer, e.g. `form='image/tiff'`
- **extent** (*List[Union[float,int]]*) – Extent of the tile to be downloaded, size may be restricted by server, e.g. `extent=[0, 972, 0, 1069]`
- **size** (*int*) – Size of the quadratic tile that is downloaded, e.g. `size=2000`
- **path** (*str*) – Path where the file is going to be downloaded, e.g. `name='tile1'`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

```
>>> # Downloading files from WCS Service
>>> gg.web.load_as_files(wcs_url=wcs.url, version=wcs.version, form='image/tiff',␣
↪extent=[0, 10000, 0, 10000], size=2000, path='tile.tif')
```

See also:

load_wcs Load WCS Service

create_request Create request for WCS

load_as_file Download WCS data file

8.5.5 gemgis.web.load_as_gpd

gemgis.web.**load_as_gpd**(*url: str*, *typename: str = None*, *outputformat: str = None*) → geopandas.geodataframe.GeoDataFrame

Requesting data from a WFS Service

Parameters

- **url** (*str*) – URL of the Web Feature Service, e.g. `url="https://nibis.lbeg.de/net3/public/ogc.ashx?NodeId=476&Service=WFS&"`
- **typename** (*str*) – Name of the feature layer, e.g. `typename='iwan:L383'`, default is `None`
- **outputformat** (*str*) – Output format of the feature layer, e.g. `outputformat='xml/gml2'`, default is `None`

Returns **feature** – GeoDataFrame containing the feature data of the WFS Service

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WFS Service as GeoDataFrame
>>> import gemgis as gg
>>> wfs = gg.web.load_as_gpd(url="https://nibis.lbeg.de/net3/public/ogc.ashx?
↪NodeId=476&Service=WFS&")
>>> wfs
    gml_id      OBJECTID    ID  SURVEYNAME      ARCHIV  MESSJAHR        OPERATOR  ␣
↪                                    OP_NACHFOL                          MESSFIRMA␣
↪                                MESSPUNKTE  UP_DATE                     geometry
0       1541    1541        112 Jemgum 2007     0127494 2007        GdF Produktion␣
↪Exploration Deutschland GmbH Neptune Energy Deutschland GmbH Geophysik und␣
↪Geotechnik Leipzig GmbH   1340        2020-01-20T00:00:00+01:00   MULTIPOLYGON␣
↪(((32395246.839 5907777.660, 3239...
```

(continues on next page)

(continued from previous page)

See also:

load_wfs Load WFS Service

8.5.6 gemgis.web.load_as_map

gemgis.web.**load_as_map**(*url: str*, *layer: str*, *style: str*, *crs: Union[str, dict]*, *bbox: List[Union[int, float]]*, *size: List[int]*, *filetype: str*, *transparent: bool = True*, *save_image: bool = False*, *path: str = None*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Loading a portion of a WMS as array

Parameters

- **url** (*str*) – Link of the WMS Service, e.g. `url='https://ows.terrestris.de/osm/service?'`
- **layer** (*str*) – Name of layer to be requested, e.g. `layer='OSM-WMS'`
- **style** (*str*) – Name of style of the layer, e.g. `style='default'`
- **crs** (*str*) – String or dict containing the CRS, e.g. `crs='EPSG:4647'`
- **bbox** (*List[Union[float,int]]*) – List of bounding box coordinates, e.g. `bbox=[0, 972, 0, 1069]`
- **size** (*List[int]*) – List of x and y values defining the size of the image, e.g. `size=[1000, 1000]`
- **filetype** (*str*) – String of the image type to be downloaded, e.g. `filetype='image/png'`
- **transparent** (*bool*) – Variable if layer is transparent. Options include: `True` or `False`, default set to `True`
- **save_image** (*bool*) – Variable to save image. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path and file name of the file to be saved, e.g. `path=map.tif`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **wms_map** – OWSlib map object

Return type owslib.util.ResponseWrapper

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WMS Service as Map
>>> import gemgis as gg
>>> wms_map = gg.web.load_as_map(url='https://ows.terrestris.de/osm/service?',␣
↪layer='OSM-WMS', style='default', crs='EPSG:4647', bbox=[32286000,32328000,␣
↪5620000,5648000], size=[4200, 2800], filetype='image/png')
>>> wms_map
<owslib.util.ResponseWrapper at 0x261d348cc10>
```

See also:

load_wms Load WMS Service

load_as_array Load Map as array from WMS Service

8.5.7 gemgis.web.load_wcs

gemgis.web.**load_wcs**(*url: str*)

Loading Web Coverage Service

Parameters **url** (*str*) – Link of the Web Coverage Service, e.g. url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'

Returns **wcs** – OWSLib Web Coverage Object

Return type owslib.coverage.wcs201.WebCoverageService_2_0_1

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

See also:

create_request Create request for WCS

load_as_file Download WCS data file

load_as_files Download WCS data files

8.5.8 gemgis.web.load_wfs

gemgis.web.**load_wfs**(*url: str*)

Loading a WFS Service by URL

Parameters **url** (*str*) – Link of the WFS Service, e.g. url="https://nibis.lbeg.de/net3/public/ogc.ashx?NodeId=476&Service=WFS&"

Returns **wfs** – OWSLib Feature object

Return type owslib.feature.wfs100.WebFeatureService_1_0_0

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WFS Service
>>> import gemgis as gg
>>> wfs = gg.web.load_wfs(url="https://nibis.lbeg.de/net3/public/ogc.ashx?
↪NodeId=476&Service=WFS&")
>>> wfs
<owslib.feature.wfs100.WebFeatureService_1_0_0 at 0x19260e21340>
```

See also:

load_as_gpd Load information of a WFS Service as GeoDataFrame

8.5.9 gemgis.web.load_wms

gemgis.web.**load_wms**(*url: str*)

Loading a WMS Service by URL

Parameters **url** (*str*) – Link of the WMS Service, e.g. url='https://ows.terrestris.de/osm/service?'

Returns **wms** – OWSLib WebMapService Object

Return type owslib.map.wms111.WebMapService

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WMS Service
>>> import gemgis as gg
>>> wms = gg.web.load_wms(url='https://ows.terrestris.de/osm/service?')
>>> wms
<owslib.map.wms111.WebMapService_1_1_1 at 0x1c434eb6370>
```

See also:

load_as_map Load Map from WMS Service

load_as_array Load Map as array from WMS Service

8.6 Miscellaneous

The following sections provide an overview of the methods implemented in the GemGIS Misc module.

gemgis.misc.get_meta_data(page)	This function is used to extract the name, coordinates and depths, of one page with one well provided by the Geological Survey NRW.
gemgis.misc.get_meta_data_df(data[, name, ...])	Function to create a dataframe with coordinates and meta data of the different boreholes
gemgis.misc.get_stratigraphic_data(text, ...)	Function to retrieve the stratigraphic data from borehole logs
gemgis.misc.get_stratigraphic_data_df(data, ...)	Function to create a dataframe with coordinates and the stratigraphy of the different boreholes
gemgis.misc.load_formations(path)	Loading formations for extraction of borehole data
gemgis.misc.load_pdf(path[, save_as_txt])	Load PDF file containing borehole data.
gemgis.misc.load_symbols(path)	Loading symbols for extraction of borehole data

8.6.1 gemgis.misc.get_meta_data

gemgis.misc.**get_meta_data**(*page: List[str]*) → list

This function is used to extract the name, coordinates and depths, of one page with one well provided by the Geological Survey NRW. It is using the extracted page as string as input data and returns floats of the coordination data and the well name

Parameters **page** (*List[str]*) – List containing the strings of the borehole pdf

Returns **data** – List containing the extracted data values

Return type list

New in version 1.0.x.

Changed in version 1.1.7.

Adapting positions of coordinate values.

Example

```
>>> # Loading Libraries and split data
>>> import gemgis as gg
>>> # Split Data - from get_meta_data_df(...)
>>> data = data.split()
>>> data = '#'.join(data)
>>> data = data.split('-Stammdaten')
>>> data = [item.split('|')[0] for item in data]
>>> data = [item.split('#') for item in data]
```

```
>>> # Filter out wells without Stratigraphic Column
>>> data = [item for item in data if 'Beschreibung' in item]
```

```
>>> # Get Coordinates of data
>>> coordinates = [get_meta_data(page=item) for item in data]
>>> coordinates[0]
['DABO_196747', 'B.19ESCHWEILER', '19', 70.3, 32310019.32, 5633520.32, 130.0,
2521370.0, 5631910.0, 'Karbon', 'Eschweiler [TK 5103]', 'Eschweiler/Weißweiler',
'ungeprüfte Angabe aus dem Bohrarchiv', 'ungeprüfte Angabe aus dem Bohrarchiv',
'Exploration, Lagerstättenerkundung', 'Bohrung', '', 'vertraulich, offen nach␣
↪Einzelfallprüfung;',
'Übertragung eines alten Archivbestandes', '1', 'Schichtdaten von guter Qualität;␣
↪genaue stratigrafische
Einstufung aufgestellt', '', '', 'Original-Schichtenverzeichnis liegt vor']
```

See also:

load_pdf Loading PDF data as string

get_meta_data_df Getting the meta data of wells as DataFrame

get_stratigraphic_data Getting the stratigraphic data of a well

get_stratigraphic_data_df Getting the stratigraphic data of wells as DataFrame

8.6.2 gemgis.misc.get_meta_data_df

gemgis.misc.**get_meta_data_df**(*data: str*, *name: str = 'GD'*, *return_gdf: bool = True*) → Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]

Function to create a dataframe with coordinates and meta data of the different boreholes

Parameters

- **data** (*str*) – String containing the borehole data
- **name** (*str*) – Prefix for custom index for boreholes, default 'GD', e.g. `name='GD'`
- **return_gdf** (*bool*) – Variable to return GeoDataFrame. Options include: `True` or `False`, default set to `True`

Returns **coordinates_dataframe_new** – (Geo-)DataFrame containing the coordinates and meta data of the boreholes

Return type Union[pd.DataFrame, gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Changed in version 1.1.7.

Fixed bug in parsing PDF.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> content = gg.misc.load_pdf(path='file.pdf')
>>> content
'Stammdaten    -     2521/ 5631/ 1         -          Bnum: 196747  .  .  Objekt /␣
↪Name :B. 19  ESCHWEILER
Bohrungs- / Aufschluß-Nr. :19  Archiv-Nr. :  Endteufe [m] :70.30  Stratigraphie der␣
↪Endteufe :Karbon
.  TK 25 :Eschweiler [TK 5103]  Ort / Gemarkung :Eschweiler/Weißweiler  GK   R...'
```

```
>>> # Creating meta data DataFrame
>>> gdf = gg.misc.get_meta_data_df(data=content, name='GD', return_gdf=True)
>>> gdf
    Index   DABO No.    Name            Number  Depth   X           Y           Z  ␣
↪    X_GK        Y_GK        ... Kind    Procedure   Confidentiality              ␣
↪              Record Type                             Lithlog Version Quality    ␣
↪                                    Drilling Period Remarks Availability␣
↪Lithlog                    geometry
0   GD0001  DABO_196747 B.19ESCHWEILER  19      70.30   32310019.32 5633520.32  130.
↪00  2521370.00  5631910.00  ... Bohrung             vertraulich, offen nach␣
↪Einzelfallprüfung;  Übertragung eines alten Archivbestandes 1              ␣
↪Schichtdaten von guter Qualität; genaue strati...                          ␣
↪Original-Schichtenverzeichnis liegt vor POINT (32310019.320 5633520.320)
1   GD0002  DABO_196748 B.16ESCHWEILER  16      37.61   2310327.14  5632967.35  122.
↪00  2521700.00  5631370.00  ... Bohrung             vertraulich, offen nach␣
↪Einzelfallprüfung;  Übertragung eines alten Archivbestandes 1              ␣
↪Schichtdaten von guter Qualität; genaue strati...                          ␣
↪Original-Schichtenverzeichnis liegt vor POINT (32310327.140 5632967.350)
```

See also:

load_pdf Loading PDF data as string

get_meta_data Getting the meta data of a well

get_stratigraphic_data Getting the stratigraphic data of a well

get_stratigraphic_data_df Getting the stratigraphic data of wells as DataFrame

8.6.3 gemgis.misc.get_stratigraphic_data

gemgis.misc.**get_stratigraphic_data**(*text: list*, *symbols: List[Tuple[str, str]]*, *formations: List[Tuple[str, str]]*) → list

Function to retrieve the stratigraphic data from borehole logs

Parameters

- **text** (*list*) – String containing the borehole data
- **symbols** (*List[Tuple[str, str]]*) – List of symbols to be removed from list of strings
- **formations** (*List[Tuple[str, str]]*) – List of categorized formations

Returns **data** – List of extracted data values

Return type list

New in version 1.0.x.

Changed in version 1.1.7.

Fixed bug in parsing PDF.

Example

```
>>> # Loading Libraries and getting the stratigraphic data of borehole
>>> import gemgis as gg
>>> data = gg.misc.get_stratigraphic_data(text=text, symbols=symbols,␣
↪formations=formations)
```

See also:

load_pdf Loading PDF data as string

get_meta_data Getting the meta data of a well

get_meta_data_df Getting the meta data of wells as DataFrame

get_stratigraphic_data_df Getting the stratigraphic data of wells as DataFrame

8.6.4 gemgis.misc.get_stratigraphic_data_df

gemgis.misc.**get_stratigraphic_data_df**(*data: str*, *name: str*, *symbols: List[Tuple[str, str]]*, *formations: List[Tuple[str, str]]*, *remove_last: bool = False*, *return_gdf: bool = True*) → Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]

Function to create a dataframe with coordinates and the stratigraphy of the different boreholes

Parameters

- **data** (*list*) – List containing the strings of the borehole log
- **name** (*str*) – Name for index reference, e.g. `name='GD'`
- **symbols** (*List[Tuple[str, str]]*) – List of tuples with symbols to be filtered out
- **formations** (*List[Tuple[str, str]]*) – List of tuples with formation names to be replaced
- **remove_last** – Variable to remove the last value of each well. Options include: `True` or `False`, default set to `False`

Returns **strata** – (Geo-)DataFrame containing the coordinates and the stratigraphy of the boreholes

Return type Union[pd.DataFrame, gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Changed in version 1.1.7.

Fixed bug in parsing PDF.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> content = gg.misc.load_pdf(path='file.pdf')
>>> content
'Stammdaten    -     2521/ 5631/ 1         -          Bnum: 196747  .  .  Objekt /␣
↪Name :B. 19  ESCHWEILER
Bohrungs- / Aufschluß-Nr. :19  Archiv-Nr. :  Endteufe [m] :70.30  Stratigraphie der␣
↪Endteufe :Karbon
.  TK 25 :Eschweiler [TK 5103]  Ort / Gemarkung :Eschweiler/Weißweiler  GK   R...'
```

```
>>> # Getting stratigraphic data DataFrame
>>> gdf = gg.misc.get_stratigraphic_data_df(data=data, name='GD', symbols=symbols,␣
↪formations=formations)
>>> gdf
    Index   Name            X           Y           Z       Altitude    Depth  ␣
↪formation   geometry
0   GD0001  B.19ESCHWEILER  32310019.32 5633520.32  125.30  130.00      70.30  ␣
↪Quaternary  POINT (32310019.320 5633520.320)
1   GD0001  B.19ESCHWEILER  32310019.32 5633520.32  66.50   130.00      70.30  ␣
↪Miocene     POINT (32310019.320 5633520.320)
2   GD0001  B.19ESCHWEILER  32310019.32 5633520.32  60.90   130.00      70.30  ␣
↪Oligocene   POINT (32310019.320 5633520.320)
```

See also:

load_pdf Loading PDF data as string

get_meta_data Getting the meta data of a well

get_meta_data_df Getting the meta data of wells as DataFrame

get_stratigraphic_data Getting the stratigraphic data of a well

8.6.5 gemgis.misc.load_formations

gemgis.misc.**load_formations**(*path: str*) → list

Loading formations for extraction of borehole data

Parameters **path** (*str*) – Path to the file containing the symbols for extracting the borehole data, e.g. `path='boreholes.txt'`

Returns **formations** – List of tuples with formations to be extracted

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> formations = gg.misc.load_formations(paths='formations.txt')
```

```
>>> # Inspecting the formations
>>> formations
[('UnterdevonKalltalFormation', 'KalltalFM'),
('Bölling', 'Quaternary'),
('AtlantikumAuenterrassen[TalterrasseInselterrasse]', 'Quaternary'),
('nullLöss', 'Quaternary'),
('Waal', 'Quaternary')]
```

8.6.6 gemgis.misc.load_pdf

gemgis.misc.**load_pdf**(*path: str*, *save_as_txt: bool = True*) → str

Load PDF file containing borehole data.

Parameters

- **path** (*str*) – Name of the PDF file, e.g. `path='file.pdf'`.
- **save_as_txt** (bool, default: `True`) – Variable to save the extracted data as txt file. Options include: `True` or `False`.

Returns Extracted page content from borehole data.

Return type str

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> content = gg.misc.load_pdf(path='file.pdf')
>>> content
'Stammdaten    -     2521/ 5631/ 1         -          Bnum: 196747  .  .  Objekt /␣
↪Name :B. 19  ESCHWEILER
Bohrungs- / Aufschluß-Nr. :19  Archiv-Nr. :  Endteufe [m] :70.30  Stratigraphie der␣
↪Endteufe :Karbon
.  TK 25 :Eschweiler [TK 5103]  Ort / Gemarkung :Eschweiler/Weißweiler  GK   R...'
```

See also:

get_meta_data Get the meta data of a well.

get_meta_data_df Get the meta data of wells as DataFrame.

get_stratigraphic_data Get the stratigraphic data of a well.

get_stratigraphic_data_df Get the stratigraphic data of wells as DataFrame.

8.6.7 gemgis.misc.load_symbols

gemgis.misc.**load_symbols**(*path: str*) → list

Loading symbols for extraction of borehole data

Parameters **path** (*str*) – Path to the file containing the symbols for extracting the borehole data, e.g. path='boreholes.txt'

Returns **symbols** – List of tuples with symbols to be removed

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> symbols = gg.misc.load_symbols(paths='symbols.txt')
```

```
>>> # Inspecting the symbols
>>> symbols
[('.m ', ''),
(', ', ''),
('; ', ''),
(': ', ''),
('/ ', ''),
('? ', ''),
('! ', ''),
('-"- ', ''),
('" ', ''),
('% ', ''),
('< ', ''),
('> ', ''),
('= ', ''),
('~ ', ''),
('_ ', ''),
('^A° ', ''),
("' ", '')]
```

8.7 Postprocessing

The following section provides an overview of the methods implemented in the GemGIS postprocessing module.

gemgis.postprocessing.calculate_dip_and_azimuth_from_mesh(mesh)	Calculating dip and azimuth values for a mesh and setting them as scalars for subsequent plotting
gemgis.postprocessing.create_attributes(...)	Creating a list of attribute dicts
gemgis.postprocessing.create_subelement(...)	Creating Subelement
gemgis.postprocessing.create_symbol(parent, ...)	Creating symbol entry
gemgis.postprocessing.crop_block_to_topography(...)	Cropping GemPy solutions block to topography
gemgis.postprocessing.extract_borehole(...)	Extracting a borehole at a provided location from a recalculated GemPy Model
gemgis.postprocessing.extract_lithologies(...)	Extracting the geological map as GeoDataFrame
gemgis.postprocessing.extract_orientations_from_mesh(...)	Extracting orientations (dip and azimuth) from PyVista Mesh
gemgis.postprocessing.save_model(geo_model, path)	Function to save the model parameters to files
gemgis.postprocessing.save_qgis_qml_file(gdf)	Creating and saving a QGIS Style File/QML File based on a GeoDataFrame
gemgis.postprocessing.clip_fault_of_gempy_model(...)	Clip fault of a GemPy model.
gemgis.postprocessing.create_plane_from_interface_and_orientation_dfs(...)	Create PyVista plane from GemPy interface and orientations DataFrames.
gemgis.postprocessing.translate_clipping_plane(...)	Translate clipping plane.

8.7.1 gemgis.postprocessing.calculate_dip_and_azimuth_from_mesh

gemgis.postprocessing.**calculate_dip_and_azimuth_from_mesh**(*mesh: pyvista.core.pointset.PolyData*) → pyvista.core.pointset.PolyData

Calculating dip and azimuth values for a mesh and setting them as scalars for subsequent plotting

Parameters **mesh** (*pv.core.pointset.PolyData*) – PyVista Mesh for which the dip and the azimuth will be calculated

Returns **mesh** – PyVista Mesh with appended dips and azimuths

Return type pv.core.pointset.PolyData

New in version 1.0.x.

8.7.2 gemgis.postprocessing.create_attributes

gemgis.postprocessing.**create_attributes**(*keys: list*, *values: list*) → list

Creating a list of attribute dicts

Parameters

- **key** (*list*) – List of keys to create the attributes with
- **values** (*list*) – List of values for the dicts

Returns **dicts** – List containing the attribute dicts

Return type list

New in version 1.0.x.

8.7.3 gemgis.postprocessing.create_subelement

gemgis.postprocessing.**create_subelement**(*parent: xml.etree.ElementTree.Element*, *name: str*, *attrib: dict*)

Creating Subelement

Parameters

- **parent** (*xml.etree.ElementTree.Element*) – Parent Element
- **name** (*str*) – Name of the Element
- **attrib** (*dict*) – Dict containing the attributes of the element

New in version 1.0.x.

8.7.4 gemgis.postprocessing.create_symbol

gemgis.postprocessing.**create_symbol**(*parent: xml.etree.ElementTree.Element*, *color: str*, *symbol_text: str*, *outline_width: str = '0.26'*, *alpha: str = '1'*)

Creating symbol entry

Parameters

- **parent** (*xml.etree.ElementTree.Element*) – Parent Element
- **color** (*str*) – RGBA values provided as string
- **outline_width** (*str*) – Outline width of the polygons
- **alpha** (*str*) – Opacity value
- **symbol_text** (*str*) – Number of the symbol

New in version 1.0.x.

8.7.5 gemgis.postprocessing.crop_block_to_topography

gemgis.postprocessing.**crop_block_to_topography**(*geo_model*) → pyvista.core.pointset.UnstructuredGrid

Cropping GemPy solutions block to topography

Parameters **geo_model** (*gp.core.model.Project*) –

Returns **grid**

Return type pv.core.pointset.UnstructuredGrid

New in version 1.0.x.

8.7.6 gemgis.postprocessing.extract_borehole

gemgis.postprocessing.**extract_borehole**(*geo_model*, *geo_data: gemgis.gemgis.GemPyData*, *loc: List[Union[int, float]]*, ***kwargs*)

Extracting a borehole at a provided location from a recalculated GemPy Model

Parameters

- **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model
- **geo_data** (*gemgis.GemPyData*) – GemGIS GemPy Data class used to calculate the previous model
- **loc** (*list*) – List of X and Y point pairs representing the well location
- **zmax** (*Union[int, float]*) – Value indicating the maximum depth of the well, default is minz of the previous model
- **res** (*int*) – Value indicating the resolution of the model in z-direction

Returns

- **sol** (*np.ndarray*)
- **well_model** (*gp.core.model.Project*)
- **depth_dict** (*dict*)

New in version 1.0.x.

8.7.7 gemgis.postprocessing.extract_lithologies

gemgis.postprocessing.**extract_lithologies**(*geo_model*, *extent: list*, *crs: Union[str, pyproj.crs.crs.CRS]*) → geopandas.geodataframe.GeoDataFrame

Extracting the geological map as GeoDataFrame

Parameters

- **geo_model** (*gp.core.model.Project*) – GemPy geo_model
- **extent** (*list*) – Extent of geo_model
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Coordinate References System

Returns **lith** – Lithologies of the geological map

Return type gpd.geodataFrame.GeoDataFrame

New in version 1.0.x.

8.7.8 gemgis.postprocessing.extract_orientations_from_mesh

gemgis.postprocessing.**extract_orientations_from_mesh**(*mesh: pyvista.core.pointset.PolyData*, *crs: Union[str, pyproj.crs.crs.CRS]*) → geopandas.geodataframe.GeoDataFrame

Extracting orientations (dip and azimuth) from PyVista Mesh

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista Mesh from which the orientations will be extracted

- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Coordinate reference system of the returned GeoDataFrame, `crs='EPSG:4326'`

Returns **gdf_orientations** – GeoDataFrame consisting of the orientations

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

8.7.9 gemgis.postprocessing.save_model

gemgis.postprocessing.**save_model**(*geo_model*, *path*)

Function to save the model parameters to files

Parameters

- **geo_model** (*gp.core.model.Project*) – GemPy model to be saved
- **path** (*str*) – Path/folder where data is stored, e.g. `path='model/'`

New in version 1.0.x.

8.7.10 gemgis.postprocessing.save_qgis_qml_file

gemgis.postprocessing.**save_qgis_qml_file**(*gdf: geopandas.geodataframe.GeoDataFrame*, *value: str = 'formation'*, *color: str = 'color'*, *outline_width: Union[int, float] = 0.26*, *alpha: Union[int, float] = 1*, *path: str = ''*)

Creating and saving a QGIS Style File/QML File based on a GeoDataFrame

Parameters

- **gdf** (*gpd.geoDataFrame.GeoDataFrame*) – GeoDataFrame containing the Polygons, formation names and color values
- **value** (*str*) – Name of the column used to categorize the layer
- **color** (*str*) – Name of the column containing the color values
- **outline_width** (*Union[int, float]*) – Outline width of the polygons
- **path** (*str*) – Path where the QML file will be stored

New in version 1.0.x.

8.7.11 gemgis.postprocessing.clip_fault_of_gempy_model

gemgis.postprocessing.**clip_fault_of_gempy_model**(*geo_model*, *fault: str*, *which: str = 'first'*, *buffer_first: Union[int, float] = None*, *buffer_last: Union[int, float] = None*, *i_size: Union[int, float] = 1000*, *j_size: Union[int, float] = 1000*, *invert_first: bool = True*, *invert_last: bool = False*) → Union[pyvista.core.pointset.PolyData, List[pyvista.core.pointset.PolyData]]

Clip fault of a GemPy model.

Parameters

- **geo_model** (*gp.core.model.Project*) – GemPy Model containing the faults.

- **fault** (*str*) – String or list of strings containing the name of faults to be clipped, e.g. `faults='Fault1'`.
- **which** (str, default: `'first'`) – Parameter to decide which end of the faults to clip. Options include `'first'`, `'last'`, or both, e.g. `'which='first'`.
- **buffer_first** (*Union[int, float]*) – Int or float value or list of values to clip the fault/s behind the first interface point, e.g. `'buffer_first=500'`.
- **buffer_last** (*Union[int, float]*) – Int or float value or list of values to clip the fault/s behind the last interface point, e.g. `'buffer_last=500'`.
- **i_size** (*Union[int, float]*) – Size of the plane in the i direction.
- **j_size** (*Union[int, float]*) – Size of the plane in the j direction.
- **invert_first** (bool, default: `'True'`) – Invert clipping for first plane.
- **invert_last** (bool, default: `'False'`) – Invert clipping for second plane.

Returns Clipped faults.

Return type pv.core.pointset.PolyData

New in version 1.1.

See also:

create_plane_from_interface_and_orientation Create PyVista plane from GemPy interface and orientations DataFrames.

translate_clipping_plane Translate clipping plane.

Example

8.7.12 gemgis.postprocessing.create_plane_from_interface_and_orientation_dfs

gemgis.postprocessing.**create_plane_from_interface_and_orientation_dfs**(*df_interface: pandas.core.frame.DataFrame*, *df_orientations: pandas.core.frame.DataFrame*, *i_size: Union[int, float] = 1000*, *j_size: Union[int, float] = 1000*) → pyvista.core.pointset.PolyData

Create PyVista plane from GemPy interface and orientations DataFrames.

Parameters

- **df_interface** (*pd.DataFrame*) – GemPy Pandas DataFrame containing the interface point for the plane creation.
- **df_orientations** (*pd.DataFrame*) – GemPy Pandas Dataframe containing the orientations for the plane creation.
- **i_size** (*Union[int, float]*) – Size of the plane in the i direction.
- **j_size** (*Union[int, float]*) – Size of the plane in the j direction.

Returns

- **plane** (*pv.core.pointset.PolyData*) – Plane for clipping the fault.
- **azimuth** (*Union[int, float]*) – Azimuth of the fault.

New in version 1.1.

See also:

clip_fault_of_gempy_model Clip fault of a GemPy model.

translate_clipping_plane Translate clipping plane.

Example

8.7.13 gemgis.postprocessing.translate_clipping_plane

gemgis.postprocessing.**translate_clipping_plane**(*plane: pyvista.core.pointset.PolyData*, *azimuth: Union[int, float, numpy.int64]*, *buffer: Union[int, float]*) → pyvista.core.pointset.PolyData

Translate clipping plane.

Parameters

- **plane** (*pv.core.pointset.PolyData*) – Clipping Plane.
- **azimuth** (*Union[int, float, np.int64]*) – Orientation of the Fault.
- **buffer** (*Union[int, float, type(None)]*) – Buffer to translate the clipping plane along the strike of the fault.

Returns Translated clipping plane.

Return type pv.core.pointset.PolyData

New in version 1.1.

See also:

create_plane_from_interface_and_orientation Create PyVista plane from GemPy interface and orientations DataFrames.

clip_fault_of_gempy_model Clip fault of a GemPy model.

Example

8.8 Download GemGIS Data

The following section provides an overview of the methods implemented in the GemGIS module that downloads the tutorial data.

gemgis.download_gemgis_data.create_pooch(...)	Create pooch class to fetch files from a website.
gemgis.download_gemgis_data.download_tutorial_data(...)	Download the GemGIS data for each tutorial.

8.8.1 gemgis.download_gemgis_data.create_pooch

gemgis.download_gemgis_data.**create_pooch**(*storage_url: str*, *files: List[str]*, *target: str*)

Create pooch class to fetch files from a website.

Parameters

- **storage_url** (*str*) – Base URL for the remote data source.
- **files** (*List[str]*) – A record of the files that are managed by this Pooch.
- **target** (str, default: '') – The path to the local data storage folder, e.g. target='Documents/gemgis/'.

Returns Pooch class.

Return type pooch.core.Pooch

See also:

download_tutorial_data Download the GemGIS data for each tutorial.

8.8.2 gemgis.download_gemgis_data.download_tutorial_data

gemgis.download_gemgis_data.**download_tutorial_data**(*filename: str*, *dirpath: str = ''*, *storage_url: str = 'https://rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F'*)

Download the GemGIS data for each tutorial.

Parameters

- **filename** (*str*) – File name to be downloaded by pooch, e.g. filename='file.zip'.
- **dirpath** (str, default: '') – Path to the directory where the data is being stored, default to the directory where the notebook is located, e.g. dirpath='Documents/gemgis/'.
- **storage_url** (str, default 'https://rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F') – URL to the GemGIS data storage, default is the RWTH Aachen University Sciebo Cloud Storage.

See also:

create_pooch Create pooch class to fetch files from a website.

CHAPTER

NINE

GEMGIS API REFERENCE

9.1 GemGIS API Reference

The API reference provides an overview of all functions and methods implemented in GemGIS.

9.1.1 gemgis.download_gemgis_data module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke.

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

gemgis.download_gemgis_data.**create_pooch**(*storage_url: str*, *files: List[str]*, *target: str*)

Create pooch class to fetch files from a website.

Parameters

- **storage_url** (*str*) – Base URL for the remote data source.
- **files** (*List[str]*) – A record of the files that are managed by this Pooch.
- **target** (str, default: `''`) – The path to the local data storage folder, e.g. `target='Documents/gemgis/'`.

Returns Pooch class.

Return type pooch.core.Pooch

See also:

download_tutorial_data Download the GemGIS data for each tutorial.

gemgis.download_gemgis_data.**download_tutorial_data**(*filename: str*, *dirpath: str = ''*, *storage_url: str = 'https://rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F'*)

Download the GemGIS data for each tutorial.

Parameters

- **filename** (*str*) – File name to be downloaded by pooch, e.g. `filename='file.zip'`.
- **dirpath** (str, default: `''`) – Path to the directory where the data is being stored, default to the directory where the notebook is located, e.g. `dirpath='Documents/gemgis/'`.
- **storage_url** (str, default 'https://rwth-aachen.sciebo.de/s/AfXRsZywYDbUF34/download?path=%2F') – URL to the GemGIS data storage, default is the RWTH Aachen University Sciebo Cloud Storage.

See also:

create_pooch Create pooch class to fetch files from a website.

9.1.2 gemgis.gemgis module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

class gemgis.gemgis.**GemPyData**(*model_name=None*, *crs=None*, *extent=None*, *resolution=None*, *interfaces=None*, *orientations=None*, *section_dict=None*, *customsections=None*, *dem=None*, *stack=None*, *surface_colors=None*, *is_fault=None*, *geolmap=None*, *basemap=None*, *faults=None*, *tectonics=None*, *raw_i=None*, *raw_o=None*, *raw_dem=None*, *wms=None*, *slope=None*, *hillshades=None*, *aspect=None*, *contours=None*)

Bases: `object`

This class creates an object with attributes containing i.e. the interfaces or orientations that can directly be passed to a GemPy Model

The following attributes are available: - model_name: string - the name of the model - crs: string - the coordinate reference system of the model - interfaces: pd DataFrame - DataFrame containing the interfaces for the GemPy model - orientations: pd DataFrame - DataFrame containing the orientations for the GemPy model - extent: list - List containing the minx, maxx, miny, maxy, minz and maxz values - section_dict: dict - Dictionary containing the section_dict for custom sections for the GemPy model - customsections: GeoDataFrame containing the Linestrings or Endpoints of custom sections - resolution: list - List containing the x,y and z resolution of the model - dem: Union[string, array] - String containing the path to the DEM or array containing DEM values - stack: dict - Dictionary containing the layer stack associated with the model - surface_colors: dict - Dictionary containing the surface colors for the model - is_fault: list - list of surface that are classified as faults - geolmap: Union[GeoDataFrame,np.ndarray rasterio.io.Datasetreader] - GeoDataFrame or array containing the geological map either as vector or raster data set - basemap: Union[np.ndarray rasterio.io.Datasetreader] - Array or rasterio object containing a base map of the area - tectonics: GeoDataFrame - GeoDataFrame containing the LineStrings of fault traces - raw_i: GeoDataFrame - GeoDataFrame containing the raw interfaces point data - raw_o: GeoDataFrame - GeoDataFrame containing the raw orientations data - raw_dem: GeoDataFrame or np.ndarray - Raw dem data such as topographic lines or (gdf) or raster (array) - slope: np.ndarray - array containing the slope values of the DEM - hillshades: np.ndarray - array containing the color values of the hillshades - aspect:

np.ndarray - array containing the aspect values of the DEM - faults: GeoDataFrame containing the Linestrings or vertices of faults - wms: np.ndarray containing data obtained from a WMS layer - contours: GeoDataFrame containing the contour lines of the model area

set_extent(*minx: Union[int, float] = 0*, *maxx: Union[int, float] = 0*, *miny: Union[int, float] = 0*, *maxy: Union[int, float] = 0*, *minz: Union[int, float] = 0*, *maxz: Union[int, float] = 0*, ***kwargs*)

Setting the extent for a model :param minx - float defining the left border of the model: :param maxx - float defining the right border of the model: :param miny - float defining the upper border of the model: :param maxy - float defining the lower border of the model: :param minz - float defining the top border of the model: :param maxz - float defining the bottom border of the model:

Kwargs: gdf - GeoDataFrame from which bounds the extent will be set

Returns extent - list with resolution values

set_resolution(*x: int*, *y: int*, *z: int*)

Setting the resolution for a model :param x - int defining the resolution in X direction: :param y - int defining the resolution in Y direction: :param z - int defining the resolution in Z direction:

Returns [x, y, z] - list with resolution values

to_gempy_df(*gdf: geopandas.geodataframe.GeoDataFrame*, *cat: str*, ***kwargs*)

Converting a GeoDataFrame into a Pandas DataFrame ready to be read in for GemPy :param gdf - gpd.geodataframe.GeoDataFrame containing spatial information: :param formation names and orientation values: :param cat - str/type of point data: :type cat - str/type of point data: interfaces or orientations

Kwargs: dem -

Returns df - interface or orientations DataFrame ready to be read in for GemPy

to_section_dict(*gdf: geopandas.geodataframe.GeoDataFrame*, *section_column: str = 'section_name'*, *resolution=None*)

Converting custom sections stored in shape files to GemPy section_dicts :param gdf - gpd.geodataframe.GeoDataFrame containing the points or lines of custom sections: :param section_column - string containing the name of the column containing the section names: :param resolution - list containing the x: :param y resolution of the custom section:

Returns section_dict containing the section names, coordinates and resolution

to_surface_color_dict(*path: str*, ***kwargs*)

Create GemPy surface color dict from a qml file :param path: str/path to the qml file

Returns dict containing the surface color values for GemPy

Return type surface_color_dict

9.1.3 gemgis.misc module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke.

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

gemgis.misc.**get_meta_data**(*page: List[str]*) → list

This function is used to extract the name, coordinates and depths, of one page with one well provided by the Geological Survey NRW. It is using the extracted page as string as input data and returns floats of the coordination data and the well name

Parameters **page** (*List[str]*) – List containing the strings of the borehole pdf

Returns **data** – List containing the extracted data values

Return type list

New in version 1.0.x.

Changed in version 1.1.7.

Adapting positions of coordinate values.

Example

```
>>> # Loading Libraries and split data
>>> import gemgis as gg
>>> # Split Data - from get_meta_data_df(...)
>>> data = data.split()
>>> data = '#'.join(data)
>>> data = data.split('-Stammdaten')
>>> data = [item.split('|')[0] for item in data]
>>> data = [item.split('#') for item in data]
```

```
>>> # Filter out wells without Stratigraphic Column
>>> data = [item for item in data if 'Beschreibung' in item]
```

```
>>> # Get Coordinates of data
>>> coordinates = [get_meta_data(page=item) for item in data]
>>> coordinates[0]
['DABO_196747', 'B.19ESCHWEILER', '19', 70.3, 32310019.32, 5633520.32, 130.0,
2521370.0, 5631910.0, 'Karbon', 'Eschweiler [TK 5103]', 'Eschweiler/Weißweiler',
'ungeprüfte Angabe aus dem Bohrarchiv', 'ungeprüfte Angabe aus dem Bohrarchiv',
'Exploration, Lagerstättenerkundung', 'Bohrung', '', 'vertraulich, offen nach␣
↪Einzelfallprüfung;',
'Übertragung eines alten Archivbestandes', '1', 'Schichtdaten von guter Qualität;␣
↪genaue stratigrafische
Einstufung aufgestellt', '', '', 'Original-Schichtenverzeichnis liegt vor']
```

See also:

load_pdf Loading PDF data as string

get_meta_data_df Getting the meta data of wells as DataFrame

get_stratigraphic_data Getting the stratigraphic data of a well

get_stratigraphic_data_df Getting the stratigraphic data of wells as DataFrame

gemgis.misc.**get_meta_data_df**(*data: str*, *name: str = 'GD'*, *return_gdf: bool = True*) → Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]

Function to create a dataframe with coordinates and meta data of the different boreholes

Parameters

- **data** (*str*) – String containing the borehole data
- **name** (*str*) – Prefix for custom index for boreholes, default 'GD', e.g. `name='GD'`
- **return_gdf** (*bool*) – Variable to return GeoDataFrame. Options include: `True` or `False`, default set to `True`

Returns **coordinates_dataframe_new** – (Geo-)DataFrame containing the coordinates and meta data of the boreholes

Return type Union[pd.DataFrame, gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Changed in version 1.1.7.

Fixed bug in parsing PDF.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> content = gg.misc.load_pdf(path='file.pdf')
>>> content
'Stammdaten    -     2521/ 5631/ 1         -          Bnum: 196747  .  .  Objekt /␣
↪Name :B. 19  ESCHWEILER
Bohrungs- / Aufschluß-Nr. :19  Archiv-Nr. :  Endteufe [m] :70.30  Stratigraphie der␣
↪Endteufe :Karbon
.  TK 25 :Eschweiler [TK 5103]  Ort / Gemarkung :Eschweiler/Weißweiler  GK   R...'
```

```
>>> # Creating meta data DataFrame
>>> gdf = gg.misc.get_meta_data_df(data=content, name='GD', return_gdf=True)
>>> gdf
    Index   DABO No.    Name            Number  Depth   X           Y           Z  ␣
↪    X_GK        Y_GK        ... Kind    Procedure   Confidentiality             ␣
↪              Record Type                             Lithlog Version Quality    ␣
↪                                  Drilling Period Remarks Availability␣
↪Lithlog                    geometry
0   GD0001  DABO_196747 B.19ESCHWEILER  19      70.30   32310019.32 5633520.32  130.
↪00  2521370.00  5631910.00  ... Bohrung             vertraulich, offen nach␣
↪Einzelfallprüfung;  Übertragung eines alten Archivbestandes 1              ␣
↪Schichtdaten von guter Qualität; genaue strati...                          ␣
↪Original-Schichtenverzeichnis liegt vor POINT (32310019.320 5633520.320)
1   GD0002  DABO_196748 B.16ESCHWEILER  16      37.61   2310327.14  5632967.35  122.
↪00  2521700.00  5631370.00  ... Bohrung             vertraulich, offen nach␣
↪Einzelfallprüfung;  Übertragung eines alten Archivbestandes 1              ␣
↪Schichtdaten von guter Qualität; genaue strati...                          ␣
↪Original-Schichtenverzeichnis liegt vor POINT (32310327.140 5632967.350)
```

See also:

load_pdf Loading PDF data as string

get_meta_data Getting the meta data of a well

get_stratigraphic_data Getting the stratigraphic data of a well

get_stratigraphic_data_df Getting the stratigraphic data of wells as DataFrame

gemgis.misc.**get_stratigraphic_data**(*text: list*, *symbols: List[Tuple[str, str]]*, *formations: List[Tuple[str, str]]*) → list

Function to retrieve the stratigraphic data from borehole logs

Parameters

- **text** (*list*) – String containing the borehole data
- **symbols** (*List[Tuple[str, str]]*) – List of symbols to be removed from list of strings
- **formations** (*List[Tuple[str, str]]*) – List of categorized formations

Returns **data** – List of extracted data values

Return type list

New in version 1.0.x.

Changed in version 1.1.7.

Fixed bug in parsing PDF.

Example

```
>>> # Loading Libraries and getting the stratigraphic data of borehole
>>> import gemgis as gg
>>> data = gg.misc.get_stratigraphic_data(text=text, symbols=symbols,␣
↪formations=formations)
```

See also:

load_pdf Loading PDF data as string

get_meta_data Getting the meta data of a well

get_meta_data_df Getting the meta data of wells as DataFrame

get_stratigraphic_data_df Getting the stratigraphic data of wells as DataFrame

gemgis.misc.**get_stratigraphic_data_df**(*data: str*, *name: str*, *symbols: List[Tuple[str, str]]*, *formations: List[Tuple[str, str]]*, *remove_last: bool = False*, *return_gdf: bool = True*) → Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]

Function to create a dataframe with coordinates and the stratigraphy of the different boreholes

Parameters

- **data** (*list*) – List containing the strings of the borehole log
- **name** (*str*) – Name for index reference, e.g. `name='GD'`
- **symbols** (*List[Tuple[str, str]]*) – List of tuples with symbols to be filtered out

- **formations** (*List[Tuple[str, str]]*) – List of tuples with formation names to be replaced
- **remove_last** – Variable to remove the last value of each well. Options include: `True` or `False`, default set to `False`

Returns **strata** – (Geo-)DataFrame containing the coordinates and the stratigraphy of the boreholes

Return type Union[pd.DataFrame, gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Changed in version 1.1.7.

Fixed bug in parsing PDF.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> content = gg.misc.load_pdf(path='file.pdf')
>>> content
'Stammdaten    -     2521/ 5631/ 1         -          Bnum: 196747  .  .  Objekt /␣
↪Name :B. 19  ESCHWEILER
Bohrungs- / Aufschluß-Nr. :19  Archiv-Nr. :  Endteufe [m] :70.30  Stratigraphie der␣
↪Endteufe :Karbon
.  TK 25 :Eschweiler [TK 5103]  Ort / Gemarkung :Eschweiler/Weißweiler  GK   R...'
```

```
>>> # Getting stratigraphic data DataFrame
>>> gdf = gg.misc.get_stratigraphic_data_df(data=data, name='GD', symbols=symbols,␣
↪formations=formations)
>>> gdf
    Index   Name            X           Y           Z       Altitude    Depth  ␣
↪formation   geometry
0   GD0001  B.19ESCHWEILER  32310019.32 5633520.32  125.30  130.00      70.30  ␣
↪Quaternary  POINT (32310019.320 5633520.320)
1   GD0001  B.19ESCHWEILER  32310019.32 5633520.32  66.50   130.00      70.30  ␣
↪Miocene     POINT (32310019.320 5633520.320)
2   GD0001  B.19ESCHWEILER  32310019.32 5633520.32  60.90   130.00      70.30  ␣
↪Oligocene   POINT (32310019.320 5633520.320)
```

See also:

load_pdf Loading PDF data as string

get_meta_data Getting the meta data of a well

get_meta_data_df Getting the meta data of wells as DataFrame

get_stratigraphic_data Getting the stratigraphic data of a well

gemgis.misc.**load_formations**(*path: str*) → list

Loading formations for extraction of borehole data

Parameters **path** (*str*) – Path to the file containing the symbols for extracting the borehole data, e.g. `path='boreholes.txt'`

Returns **formations** – List of tuples with formations to be extracted

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> formations = gg.misc.load_formations(paths='formations.txt')
```

```
>>> # Inspecting the formations
>>> formations
[('UnterdevonKalltalFormation', 'KalltalFM'),
('Bölling', 'Quaternary'),
('AtlantikumAuenterrassen[TalterrasseInselterrasse]', 'Quaternary'),
('nullLöss', 'Quaternary'),
('Waal', 'Quaternary')]
```

gemgis.misc.**load_pdf**(*path: str*, *save_as_txt: bool = True*) → str

Load PDF file containing borehole data.

Parameters

- **path** (*str*) – Name of the PDF file, e.g. `path='file.pdf'`.
- **save_as_txt** (bool, default: `True`) – Variable to save the extracted data as txt file. Options include: `True` or `False`.

Returns Extracted page content from borehole data.

Return type str

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> content = gg.misc.load_pdf(path='file.pdf')
>>> content
'Stammdaten    -     2521/ 5631/ 1         -          Bnum: 196747  .  .  Objekt /␣
↪Name :B. 19  ESCHWEILER
Bohrungs- / Aufschluß-Nr. :19  Archiv-Nr. :  Endteufe [m] :70.30  Stratigraphie der␣
↪Endteufe :Karbon
.  TK 25 :Eschweiler [TK 5103]  Ort / Gemarkung :Eschweiler/Weißweiler  GK   R...'
```

See also:

get_meta_data Get the meta data of a well.

get_meta_data_df Get the meta data of wells as DataFrame.

get_stratigraphic_data Get the stratigraphic data of a well.

get_stratigraphic_data_df Get the stratigraphic data of wells as DataFrame.

gemgis.misc.**load_symbols**(*path: str*) → list

Loading symbols for extraction of borehole data

Parameters **path** (*str*) – Path to the file containing the symbols for extracting the borehole data, e.g. path='boreholes.txt'

Returns **symbols** – List of tuples with symbols to be removed

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> symbols = gg.misc.load_symbols(paths='symbols.txt')
```

```
>>> # Inspecting the symbols
>>> symbols
[('.m ', ''),
(', ', ''),
('; ', ''),
(': ', ''),
('/ ', ''),
('? ', ''),
('! ', ''),
('-"- ', ''),
('" ', ''),
('% ', ''),
('< ', ''),
('> ', ''),
('= ', ''),
('~ ', ''),
('_ ', ''),
('^A° ', ''),
("' ", '')]
```

9.1.4 gemgis.postprocessing module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

gemgis.postprocessing.**calculate_dip_and_azimuth_from_mesh**(*mesh: pyvista.core.pointset.PolyData*) → pyvista.core.pointset.PolyData

Calculating dip and azimuth values for a mesh and setting them as scalars for subsequent plotting

Parameters **mesh** (*pv.core.pointset.PolyData*) – PyVista Mesh for which the dip and the azimuth will be calculated

Returns **mesh** – PyVista Mesh with appended dips and azimuths

Return type pv.core.pointset.PolyData

New in version 1.0.x.

gemgis.postprocessing.**clip_fault_of_gempy_model**(*geo_model*, *fault: str*, *which: str = 'first'*, *buffer_first: Union[int, float] = None*, *buffer_last: Union[int, float] = None*, *i_size: Union[int, float] = 1000*, *j_size: Union[int, float] = 1000*, *invert_first: bool = True*, *invert_last: bool = False*) → Union[pyvista.core.pointset.PolyData, List[pyvista.core.pointset.PolyData]]

Clip fault of a GemPy model.

Parameters

- **geo_model** (*gp.core.model.Project*) – GemPy Model containing the faults.
- **fault** (*str*) – String or list of strings containing the name of faults to be clipped, e.g. `faults='Fault1'`.
- **which** (str, default: `'first'`) – Parameter to decide which end of the faults to clip. Options include `'first'`, `'last'`, or both, e.g. `'which='first'`.
- **buffer_first** (*Union[int, float]*) – Int or float value or list of values to clip the fault/s behind the first interface point, e.g. `'buffer_first=500'`.
- **buffer_last** (*Union[int, float]*) – Int or float value or list of values to clip the fault/s behind the last interface point, e.g. `'buffer_last=500'`.
- **i_size** (*Union[int, float]*) – Size of the plane in the i direction.
- **j_size** (*Union[int, float]*) – Size of the plane in the j direction.
- **invert_first** (bool, default: `'True'`) – Invert clipping for first plane.
- **invert_last** (bool, default: `'False'`) – Invert clipping for second plane.

Returns Clipped faults.

Return type pv.core.pointset.PolyData

New in version 1.1.

See also:

create_plane_from_interface_and_orientation Create PyVista plane from GemPy interface and orientations DataFrames.

translate_clipping_plane Translate clipping plane.

Example

gemgis.postprocessing.**create_attributes**(*keys: list*, *values: list*) → list

Creating a list of attribute dicts

Parameters

- **key** (*list*) – List of keys to create the attributes with
- **values** (*list*) – List of values for the dicts

Returns **dicts** – List containing the attribute dicts

Return type list

New in version 1.0.x.

gemgis.postprocessing.**create_plane_from_interface_and_orientation_dfs**(*df_interface: pandas.core.frame.DataFrame*, *df_orientations: pandas.core.frame.DataFrame*, *i_size: Union[int, float] = 1000*, *j_size: Union[int, float] = 1000*) → pyvista.core.pointset.PolyData

Create PyVista plane from GemPy interface and orientations DataFrames.

Parameters

- **df_interface** (*pd.DataFrame*) – GemPy Pandas DataFrame containing the interface point for the plane creation.
- **df_orientations** (*pd.DataFrame*) – GemPy Pandas Dataframe containing the orientations for the plane creation.
- **i_size** (*Union[int, float]*) – Size of the plane in the i direction.
- **j_size** (*Union[int, float]*) – Size of the plane in the j direction.

Returns

- **plane** (*pv.core.pointset.PolyData*) – Plane for clipping the fault.
- **azimuth** (*Union[int, float]*) – Azimuth of the fault.

New in version 1.1.

See also:

clip_fault_of_gempy_model Clip fault of a GemPy model.

translate_clipping_plane Translate clipping plane.

Example

gemgis.postprocessing.**create_subelement**(*parent: xml.etree.ElementTree.Element*, *name: str*, *attrib: dict*)

Creating Subelement

Parameters

- **parent** (*xml.etree.ElementTree.Element*) – Parent Element
- **name** (*str*) – Name of the Element
- **attrib** (*dict*) – Dict containing the attributes of the element

New in version 1.0.x.

gemgis.postprocessing.**create_symbol**(*parent: xml.etree.ElementTree.Element*, *color: str*, *symbol_text: str*, *outline_width: str = '0.26'*, *alpha: str = '1'*)

Creating symbol entry

Parameters

- **parent** (*xml.etree.ElementTree.Element*) – Parent Element
- **color** (*str*) – RGBA values provided as string
- **outline_width** (*str*) – Outline width of the polygons
- **alpha** (*str*) – Opacity value
- **symbol_text** (*str*) – Number of the symbol

New in version 1.0.x.

gemgis.postprocessing.**crop_block_to_topography**(*geo_model*) → pyvista.core.pointset.UnstructuredGrid

Cropping GemPy solutions block to topography

Parameters **geo_model** (*gp.core.model.Project*) –

Returns **grid**

Return type pv.core.pointset.UnstructuredGrid

New in version 1.0.x.

gemgis.postprocessing.**extract_borehole**(*geo_model*, *geo_data: gemgis.gemgis.GemPyData*, *loc: List[Union[int, float]]*, ***kwargs*)

Extracting a borehole at a provided location from a recalculated GemPy Model

Parameters

- **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model
- **geo_data** (*gemgis.GemPyData*) – GemGIS GemPy Data class used to calculate the previous model
- **loc** (*list*) – List of X and Y point pairs representing the well location
- **zmax** (*Union[int, float]*) – Value indicating the maximum depth of the well, default is minz of the previous model
- **res** (*int*) – Value indicating the resolution of the model in z-direction

Returns

- **sol** (*np.ndarray*)
- **well_model** (*gp.core.model.Project*)

- **depth_dict** (*dict*)

New in version 1.0.x.

gemgis.postprocessing.**extract_lithologies**(*geo_model*, *extent: list*, *crs: Union[str, pyproj.crs.crs.CRS]*) → geopandas.geodataframe.GeoDataFrame

Extracting the geological map as GeoDataFrame

Parameters

- **geo_model** (*gp.core.model.Project*) – GemPy geo_model
- **extent** (*list*) – Extent of geo_model
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Coordinate References System

Returns **lith** – Lithologies of the geological map

Return type gpd.geodataFrame.GeoDataFrame

New in version 1.0.x.

gemgis.postprocessing.**extract_orientations_from_mesh**(*mesh: pyvista.core.pointset.PolyData*, *crs: Union[str, pyproj.crs.crs.CRS]*) → geopandas.geodataframe.GeoDataFrame

Extracting orientations (dip and azimuth) from PyVista Mesh

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista Mesh from which the orientations will be extracted
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Coordinate reference system of the returned GeoDataFrame, `crs='EPSG:4326'`

Returns **gdf_orientations** – GeoDataFrame consisting of the orientations

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

gemgis.postprocessing.**save_model**(*geo_model*, *path*)

Function to save the model parameters to files

Parameters

- **geo_model** (*gp.core.model.Project*) – GemPy model to be saved
- **path** (*str*) – Path/folder where data is stored, e.g. `path='model/'`

New in version 1.0.x.

gemgis.postprocessing.**save_qgis_qml_file**(*gdf: geopandas.geodataframe.GeoDataFrame*, *value: str = 'formation'*, *color: str = 'color'*, *outline_width: Union[int, float] = 0.26*, *alpha: Union[int, float] = 1*, *path: str = ''*)

Creating and saving a QGIS Style File/QML File based on a GeoDataFrame

Parameters

- **gdf** (*gpd.geoDataFrame.GeoDataFrame*) – GeoDataFrame containing the Polygons, formation names and color values
- **value** (*str*) – Name of the column used to categorize the layer
- **color** (*str*) – Name of the column containing the color values

- **outline_width** (*Union[int, float]*) – Outline width of the polygons
- **path** (*str*) – Path where the QML file will be stored

New in version 1.0.x.

gemgis.postprocessing.**translate_clipping_plane**(*plane: pyvista.core.pointset.PolyData*, *azimuth: Union[int, float, numpy.int64]*, *buffer: Union[int, float]*) → pyvista.core.pointset.PolyData

Translate clipping plane.

Parameters

- **plane** (*pv.core.pointset.PolyData*) – Clipping Plane.
- **azimuth** (*Union[int, float, np.int64]*) – Orientation of the Fault.
- **buffer** (*Union[int, float, type(None)]*) – Buffer to translate the clipping plane along the strike of the fault.

Returns Translated clipping plane.

Return type pv.core.pointset.PolyData

New in version 1.1.

See also:

create_plane_from_interface_and_orientation Create PyVista plane from GemPy interface and orientations DataFrames.

clip_fault_of_gempy_model Clip fault of a GemPy model.

Example

9.1.5 gemgis.raster module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann

GemGIS is a Python-based, open-source geographic information processing library. It is capable of preprocessing spatial data such as vector data (shape files, geojson files, geopackages), raster data, data obtained from WMS services or XML/KML files. Preprocessed data can be stored in a dedicated Data Class to be passed to the geomodeling package GemPy in order to accelerate to model building process.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

gemgis.raster.**calculate_aspect**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *band_no: int = 1*) → numpy.ndarray

Calculating the aspect based on a digital elevation model/raster

Parameters

- **raster** (*np.ndarray, rasterio.io.DatasetReader*) – NumPy array or rasterio object containing the elevation data
- **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny and maxy coordinates representing the raster extent if raster is passed as array, e.g. `extent=[0, 972, 0, 1069]`

- **band_no** (*int*) – Band number of the raster to be used for calculating the hillshades, e.g. band_no=1, default is 1

Returns **aspect** – NumPy array containing the aspect values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Calculating the aspect of a raster
>>> aspect = gg.raster.calculate_aspect(raster=raster)
>>> aspect
array([[246.37328, 245.80156, 245.04022, ..., 269.87958, 270.11377,
270.32904],....], dtype=float32)
```

See also:

calculate_hillshades Calculating the hillshades of a raster

calculate_slope Calculating the slope of a raster

calculate_difference Calculating the difference between two rasters

gemgis.raster.**calculate_difference**(*raster1: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *raster2: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *flip_array: bool = False*) → numpy.ndarray

Calculating the difference between two rasters

Parameters

- **raster1** (*np.ndarray*) – First array
- **raster2** (*np.ndarray*) – Second array
- **flip_array** (*bool*) – Variable to flip the array. Options include: `True` or `False`, default set to `False`

Returns **array_diff** – Array containing the difference between array1 and array2

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and Files
>>> import gemgis as gg
>>> import rasterio
>>> raster1 = rasterio.open(fp='raster1.tif')
>>> raster2 = rasterio.open(fp='raster2.tif')
```

```
>>> # Calculate difference between two rasters
>>> difference = gg.raster.calculate_difference(raster1=raster1, raster2=raster2)
>>> difference
array([[-10., -10., -10., ..., -10., -10., -10.],.....], dtype=float32)
```

See also:

calculate_hillshades Calculating the hillshades of a raster

calculate_slope Calculating the slope of a raster

calculate_aspect Calculating the aspect of a raster

gemgis.raster.**calculate_hillshades**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *azdeg: Union[int, float] = 225*, *altdeg: Union[int, float] = 45*, *band_no: int = 1*) → numpy.ndarray

Calculating Hillshades based on digital elevation model/raster

Parameters

- **raster** (*np.ndarray, rasterio.io.DatasetReader*) – NumPy array or rasterio object containing the elevation data
- **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny and maxy points representing the extent of the raster if raster is passed as array, e.g. `extent=[0, 972, 0, 1069]`
- **azdeg** (*Union[int, float]*) – Azimuth value for the light source direction, e.g. `azdeg=225`, default is `225` degrees
- **altdeg** (*Union[int, float]*) – Altitude value for the light source, e.g. `altdeg=45`, default is `45` degrees
- **band_no** (*int*) – Band number of the raster to be used for calculating the hillshades, e.g. `band_no=1`, default is `1`

Returns **hillshades** – NumPy array containing the hillshade color values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Calculating hillshades from raster
>>> hillshades = gg.raster.calculate_hillshades(raster=raster)
>>> hillshades
array([[250.04817, 250.21147, 250.38988, ..., 235.01764, 235.0847 ,
235.0842 ], ....], dtype=float32)
```

See also:

calculate_slope Calculating the slope of a raster

calculate_aspect Calculating the aspect of a raster

calculate_difference Calculating the difference between two rasters

gemgis.raster.**calculate_slope**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *band_no: int = 1*) → numpy.ndarray

Calculating the slope based on digital elevation model/raster

Parameters

- **raster** (*np.ndarray, rasterio.io.DatasetReader*) – NumPy array or rasterio object containing the elevation data
- **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny and maxy coordinates representing the raster extent if raster is passed as array, e.g. `extent=[0, 972, 0, 1069]`
- **band_no** (*int*) – Band number of the raster to be used for calculating the hillshades, e.g. band_no=1, default is 1

Returns **slope** – NumPy array containing the slope values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Calculating the slope of a raster
>>> slope = gg.raster.calculate_slope(raster=raster)
>>> slope
array([[37.092472, 36.95191 , 36.649662, ..., 21.988844, 22.367924,
22.584248],....], dtype=float32)
```

See also:

calculate_hillshades Calculating the hillshades of a raster

calculate_aspect Calculating the aspect of a raster

calculate_difference Calculating the difference between two rasters

gemgis.raster.**clip_by_bbox**(*raster: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *bbox: List[Union[int, float]]*, *raster_extent: List[Union[int, float]] = None*, *save_clipped_raster: bool = False*, *path: str = 'raster_clipped.tif'*, *overwrite_file: bool = False*, *create_directory: bool = False*) → numpy.ndarray

Clipping a rasterio raster or np.ndarray by a given extent

Parameters

- **raster** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Array or Rasterio object to be clipped
- **bbox** (*List[Union[int, float]]*) – Bounding box of minx, maxx, miny, maxy values to clip the raster, e.g. `bbox=[0, 972, 0, 1069]`
- **raster_extent** (*List[Union[int, float]]*) – List of float values defining the extent of the raster, default None, e.g. `raster_extent=[0, 972, 0, 1069]`
- **save_clipped_raster** (*bool*) – Variable to save the raster after clipping. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path where the raster is saved, e.g. `path='raster_clipped.tif'`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **raster_clipped** – Clipped array after clipping

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Creating bounding box and defining raster extent
>>> bbox = [250, 500, 250, 500]
>>> raster_extent = [0, 972, 0, 1069]
```

```
>>> # Clipping raster by bounding box
>>> raster_clipped = gg.raster.clip_by_bbox(raster=raster, bbox=bbox, raster_
↪extent=raster_extent)
>>> raster_clipped.shape
(65, 65)
```

See also:

clip_by_polygon Clipping raster by a Shapely Polygon

gemgis.raster.**clip_by_polygon**(*raster: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *polygon: shapely.geometry.polygon.Polygon*, *raster_extent: List[Union[int, float]] = None*, *save_clipped_raster: bool = False*, *path: str = 'raster_clipped.tif'*, *overwrite_file: bool = False*, *create_directory: bool = False*) → numpy.ndarray

Clipping/masking a rasterio raster or np.ndarray by a given shapely Polygon

Parameters

- **raster** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Array or Rasterio object to be clipped
- **polygon** (*shapely.geometry.polygon.Polygon*) – Shapely polygon defining the extent of the data, e.g. `polygon = Polygon([(0, 0), (1, 1), (1, 0)])`
- **raster_extent** (*List[Union[int, float]]*) – List of float values defining the extent of the raster, default None, e.g. `raster_extent=[0, 972, 0, 1069]`
- **save_clipped_raster** (*bool*) – Variable to save the raster after clipping, default False. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path where the raster is saved, e.g. `path='raster_clipped.tif`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **raster_clipped** – Clipped array after clipping

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> from shapely.geometry import Polygon
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Creating Shapely Polygon and defining raster extent
>>> polygon = Polygon([(250, 250), (500, 250), (500, 500), (250, 500)])
>>> raster_extent = [0, 972, 0, 1069]
```

```
>>> # Clipping the raster by a Shapely Polygon
>>> raster_clipped = gg.raster.clip_by_polygon(raster=raster, polygon=polygon,␣
↪raster_extent=raster_extent)
```

(continues on next page)

(continued from previous page)

```
>>> raster_clipped.shape
(65, 65)
```

See also:

clip_by_bbox Clipping raster by a Bounding Box

gemgis.raster.**create_filepaths**(*dirpath: str*, *search_criteria: str*, *create_directory: bool = False*) → List[str]

Retrieving the file paths of the tiles to load and to process them later

Parameters

- **dirpath** (*str*) – Path to the folder where tiles are stored, e.g. `dirpath='Documents/images/'`
- **search_criteria** (*str*) – Name of the files including file ending, use * for autocompletion by Python, e.g. `search_criteria='tile*.tif'`
- **create_directory** (*bool*) – Variable to create a new directory if directory does not exist Options include: `True` or `False`, default set to `False`

Returns **filepaths** – List of file paths

Return type List[str]

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Defining filepath
>>> filepath = 'Documents/images/'
```

```
>>> # Creating list of filepaths based on search criteria
>>> filepaths = gg.raster.create_filepaths(dirpath=filepath, search_criteria='tile*.
↪tif')
>>> filepaths
['Documents/images//tile_292000_294000_5626000_5628000.tif',
'Documents/images//tile_292000_294000_5628000_5630000.tif',
'Documents/images//tile_292000_294000_5630000_5632000.tif',
'Documents/images//tile_294000_296000_5626000_5628000.tif']
```

gemgis.raster.**create_src_list**(*dirpath: str = ''*, *search_criteria: str = ''*, *filepaths: List[str] = None*, *create_directory: bool = False*) → List[rasterio.io.DatasetReader]

Creating a list of source files

Parameters

- **dirpath** (*str*) – Path to the folder where tiles are stored, e.g. `dirpath='Documents/images/'`
- **search_criteria** (*str*) – Name of the files including file ending, use * for autocompletion by Python, e.g. `search_criteria='tile*.tif'`

- **filepaths** (*List[str]*) – List of strings containing file paths
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **src_files** – List containing the loaded rasterio datasets

Return type List[rasterio.io.DatasetReader]

New in version 1.0.x.

Example

```
>>> # Importing Libraries
>>> import gemgis as gg
```

```
>>> # Defining filepath
>>> filepath = 'Documents/images/'
```

```
>>> # Creating List of filepaths
>>> filepaths = gg.raster.create_filepaths(dirpath=filepath, search_criteria='tile*.
↪tif')
>>> filepaths
['Documents/images//tile_292000_294000_5626000_5628000.tif',
'Documents/images//tile_292000_294000_5628000_5630000.tif',
'Documents/images//tile_292000_294000_5630000_5632000.tif',
'Documents/images//tile_294000_296000_5626000_5628000.tif']
```

```
>>> # Creating list of loaded rasterio objects
>>> src_list = gg.raster.create_src_list(filepaths=filepaths)
>>> src_list
[<open DatasetReader name='Documents/images/tile_292000_294000_5626000_5628000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_292000_294000_5628000_5630000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_292000_294000_5630000_5632000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_294000_296000_5626000_5628000.tif'␣
↪mode='r'>,
```

gemgis.raster.**extract_contour_lines_from_raster**(*raster: Union[rasterio.io.DatasetReader, numpy.ndarray, str]*, *interval: int*, *extent: Union[Sequence[float], None, Sequence[int]] = None*, *target_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting contour lines from raster with a provided interval.

Parameters

- **raster** (*Union[rasterio.io.DatasetReader, np.ndarray, str]*) – Raster from which contour lines are extracted
- **extent** (*Optional[Sequence[float, int]]*) – If raster given as array: values (minx, maxx, miny, maxy) to define raster extent, e.g. `extent =[0, 972, 0, 1069]`

- **target_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – If raster given as array: name of the CRS is required to project values to coordinates of GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **interval** (*int*) – Given interval for the extracted contour lines, e.g. `interval=50`

Returns **gdf_lines** – GeoDataFrame containing the extracted contour lines as LineStrings

Return type gpd.GeoDataFrame

New in version 1.0.x.

gemgis.raster.**merge_tiles**(*src_files: List[rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *res: int = None*, *nodata: Union[float, int] = None*, *precision: int = None*, *indices: int = None*, *method: str = 'first'*) → Tuple[numpy.ndarray, affine.Affine]

Merging downloaded tiles to mosaic

Parameters

- **src_files** (*List[rasterio.io.DatasetReader]*) – List of rasterio datasets to be merged
- **extent** (*List[Union[float, int]]*) – Bounds of the output image (left, bottom, right, top). If not set, bounds are determined from bounds of input rasters, e.g. `extent=[0, 972, 0, 1069]`, default is `None`
- **res** (*int*) – Output resolution in units of coordinate reference system. If not set, the resolution of the first raster is used. If a single value is passed, output pixels will be square. e.g. `res=50`, default is `None`
- **nodata** (*Union[float, int]*) – nodata value to use in output file. If not set, uses the nodata value in the first input raster, e.g. `nodata=9999.0`, default is `None`
- **precision** (*int*) – Number of decimal points of precision when computing inverse transform, e.g. `precision=2`, default is `None`
- **indices** (*int*) – Bands to read and merge, e.g. `indices=1`, default is `None`
- **method** (*str*) – Method on how to merge the tiles, e.g. `method='first'`, default is `'first'`

Returns

- **mosaic** (*np.ndarray*) – Array containing the merged tile data
- **transform** (*affine.Affine*) – Affine Transform of the merged tiles

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Creating filepath
>>> filepath = 'Documents/images/'
```

```
>>> # Creating list of filepaths
>>> filepaths = gg.raster.create_filepaths(dirpath=filepath, search_criteria='tile*.
↪tif')
```

(continues on next page)

(continued from previous page)

```
>>> filepaths
['Documents/images//tile_292000_294000_5626000_5628000.tif',
'Documents/images//tile_292000_294000_5628000_5630000.tif',
'Documents/images//tile_292000_294000_5630000_5632000.tif',
'Documents/images//tile_294000_296000_5626000_5628000.tif']
```

```
>>> # Creating list of loaded rasterio objects
>>> src_list = gg.raster.create_src_list(filepaths=filepaths)
>>> src_list
[<open DatasetReader name='Documents/images/tile_292000_294000_5626000_5628000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_292000_294000_5628000_5630000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_292000_294000_5630000_5632000.tif'␣
↪mode='r'>,
<open DatasetReader name='Documents/images/tile_294000_296000_5626000_5628000.tif'␣
↪mode='r'>,
```

```
>>> # Merging tiles
>>> mosaic, transform = gg.raster.merge_tiles(src_files=src_list)
```

```
>>> # Inspecting the mosaic data
>>> mosaic
array([[200.72, 200.73, 200.72, ..., 204.42, 204.45, 204.45],
[200.74, 200.74, 200.75, ..., 204.43, 204.44, 204.48]
[200.76, 200.76, 200.76, ..., 204.42, 204.48, 204.5 ],
...,
[329.15, 328.86, 328.74, ..., 242.45, 242.38, 242.28],
[329.29, 329.06, 328.87, ..., 242.45, 242.39, 242.31],
[329.47, 329.3 , 329.09, ..., 242.42, 242.37, 242.32]],
dtype=float32)
```

```
>>> # Inspecting the transform of the mosaic
>>> transform
Affine(1.0, 0.0, 292000.0,
0.0, -1.0, 5632000.0)
```

gemgis.raster.**read_asc**(*path: Union[str, pathlib.Path]*) → dict

Function to read GoCAD .asc files

Parameters **path** (*Union[str, Path]*) – Path to asc file, e.g. `path='raster.asc'`

Returns **data** – Dict containing the array data, the extent, resolution and nodata_val of the raster

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and Files
>>> import gemgis as gg
>>> data = gg.raster.read_asc('raster.asc')
```

```
>>> # Inspecting the content of the dict, here we only see the nodata_vals for now
>>> data['Data']
array([[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
...,
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.],
[-99999., -99999., -99999., ..., -99999., -99999., -99999.]])
```

```
>>> data['Extent']
[-42250, 306000, 279000, 867000]
```

```
>>> data['Resolution']
250
```

```
>>> data['Nodata_val']
-99999
```

See also:

read_ts Reading a GoCAD TSurface File

read_msh Reading a Leapfrog Mesh File

read_zmap Reading Petrel ZMAP Files

gemgis.raster.**read_msh**(*path: Union[str, pathlib.Path]*) → Dict[str, numpy.ndarray]

Function to read Leapfrog .msh files - https://help.leapfrog3d.com/Geo/4.3/en-GB/Content/meshes/meshes.htm

Parameters **path** (*Union[str, Path]*) – Path to msh file, e.g. `path='mesh.msh'`

Returns **data** – Dict containing the mesh data

Return type Dict[str, np.ndarray]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> data = gg.raster.read_msh('mesh.msh')
>>> data
{'Tri': array([[    0,     1,     2],
[    0,     3,     1],
[    4,     3,     0],
```

(continues on next page)

(continued from previous page)

```
...,
[53677, 53672, 53680],
[53679, 53677, 53680],
[53673, 53672, 53677]]),
'Location': array([[ 1.44625109e+06,  5.24854344e+06, -1.12743862e+02],
[ 1.44624766e+06,  5.24854640e+06, -1.15102216e+02],
[ 1.44624808e+06,  5.24854657e+06, -1.15080548e+02],
...,
[ 1.44831008e+06,  5.24896679e+06, -1.24755449e+02],
[ 1.44830385e+06,  5.24896985e+06, -1.33694397e+02],
[ 1.44829874e+06,  5.24897215e+06, -1.42506587e+02]])}
```

See also:

read_ts Reading a GoCAD TSurface File

read_asc Reading ESRI ASC files

read_zmap Reading Petrel ZMAP Files

gemgis.raster.**read_raster_gdb**(*path: str*, *crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*, *path_out: str = ''*)

Read Raster from OpenFileGDB.

Parameters

- **path** (*str*) – Path to the OpenFileGDB.
- **crs** (*str, pyproj.crs.crs.CRS, rasterio.crs.CRS*) – Coordinate Reference System of the dataset.
- **path_out** (*str*) – Output folder path

New in version 1.1.1.

gemgis.raster.**read_ts**(*path: Union[str, pathlib.Path]*) → Tuple[list, list]

Function to read GoCAD .ts files

Parameters **path** (*Union[str, Path]*) – Path to ts file, e.g. `path='mesh.ts'`

Returns

- **vertices** (*list*) – Pandas DataFrames containing the vertex data
- **faces** (*list*) – NumPy arrays containing the faces data

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> vertices, faces = gg.raster.read_ts('mesh.ts')
```

```
>>> # Inspecting the vertices
>>> vertices
    id  X           Y           Z
0   0   297077.41   5677487.26  -838.50
1   1   297437.54   5676992.09  -816.61
```

```
>>> # Inspecting the faces
>>> faces
array([[    0,     1,     2],
[    3,     2,     4],
[    1,     5,     6],...,
[40335, 40338, 40336],
[40339, 40340, 40341],
[40341, 40342, 40339]])
```

See also:

read_msh Reading a Leapfrog Mesh File

read_asc Reading ESRI ASC files

read_zmap Reading Petrel ZMAP Files

gemgis.raster.**read_zmap**(*path: Union[str, pathlib.Path]*) → dict

Reading Petrel ZMAP Files

Parameters **path** (*Union[str, Path]*) – Path to dat file, e.g. `path='raster.dat'`

Returns **data** – Dict containing the array data, the extent, array dimension, resolution and nodata_val of the raster

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> data = gg.raster.read_zmap(path='file.dat')
```

```
>>> # Inspecting the content of the dict, here we only see the nodata_vals for now
>>> data
{'Data': array([[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan],
...,
[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan],
[nan, nan, nan, ..., nan, nan, nan]]),
 'Extent': [-42250.0, 278750.0, 306000.0, 866750.0],
 'Resolution': [250.0, 250.0],
 'Nodata_val': 0.1000000E+31,
 'Dimensions': (2244, 1285),
```

(continues on next page)

(continued from previous page)

```
'CRS': 'Amersfoort * EPSG-Nld / RD New [28992,1672]',
'Creation_date': '21/10/2019',
'Creation_time': '16',
'File_name': 'TOP_DINANTIAN_TVD_final'}
```

See also:

read_ts Reading a GoCAD TSurface File

read_msh Reading a Leapfrog Mesh File

read_asc Reading ESRI ASC files

gemgis.raster.**reproject_raster**(*path_in: str*, *path_out: str*, *dst_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Reprojecting a raster into different CRS

Parameters

- **path_in** (*str*) – Path to the source file, e.g. `path_in='Images/'`
- **path_out** (*str*) – Path for the destination file, e.g. `path_out='Images/'`
- **dst_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – CRS of the destination file, e.g. `dst_crs='EPSG:25832'`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Changed in version 1.1: Fixing an issue where the file would be closed too soon, see https://github.com/cgre-aachen/gemgis/issues/294

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Reprojecting raster
>>> gg.raster.reproject_raster(path_in='raster_in.tif', path_out='raster_out.tif',␣
↪dst_crs='EPSG:4326')
```

gemgis.raster.**resize_by_array**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *array: Union[numpy.ndarray, rasterio.io.DatasetReader]*) → numpy.ndarray

Rescaling raster to the size of another raster

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster that is being resized

- **array** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster with a size that the raster is being resized to

Returns **array_resized** – Resized array

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> import numpy as np
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Creating array
>>> array = np.zeros(100).reshape((10,10))
>>> array.shape
(10, 10)
```

```
>>> # Resizing a raster by an array
>>> raster_resized = gg.raster.resize_by_array(raster=raster, array=array)
>>> raster_resized.shape
(10, 10)
```

See also:

resize_raster Resizing a raster

gemgis.raster.**resize_raster**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *width: int*, *height: int*) → numpy.ndarray

Resizing raster to given dimensions

Parameters

- **array** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Array that will be resized
- **width** (*int*) – Width of the resized array, e.g. `width=100`
- **height** (*int*) – Height of the resized array, e.g. `height=100`

Returns **array_resized** – Resized array

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> import numpy as np
>>> raster = rasterio.open(fp='raster.tif')
>>> raster.read(1).shape
(275, 250)
```

```
>>> # Resizing raster
>>> raster_resized = gg.raster.resize_raster(raster=raster, width=10, height=10)
>>> raster_resized.shape
(10, 10)
```

See also:

resize_by_array Resizing a raster by the shape of another array

gemgis.raster.**sample_from_array**(*array: numpy.ndarray*, *extent: Sequence[float]*, *point_x: Union[float, int, list, numpy.ndarray]*, *point_y: Union[float, int, list, numpy.ndarray]*) → Union[numpy.ndarray, float]

Sampling the value of a np.ndarray at a given point and given the arrays true extent

Parameters

- **array** (*np.ndarray*) – Array containing the raster values
- **extent** (*list*) – List containing the values for the extent of the array (minx,maxx,miny,maxy), e.g. `extent=[0, 972, 0, 1069]`
- **point_x** (*Union[float, int, list, np.ndarray]*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`
- **point_y** (*Union[float, int, list, np.ndarray]*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`

Returns **sample** – Value/s of the raster at the provided position/s

Return type Union[np.ndarray, float]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Getting array data
>>> array = raster.read()
```

```
>>> # Sampling values from an array
>>> value = gg.raster.sample_from_array(array=array, extent=[0, 972, 0, 1069],␣
↪point_x=500, point_y=500)
>>> value
562.0227
```

See also:

sample_from_rasterio Sample values from rasterio object

sample_randomly Sample randomly from rasterio object or NumPy array

sample_orientations Sample orientations from raster

sample_interfaces Sample interfaces from raster

gemgis.raster.**sample_from_rasterio**(*raster: rasterio.io.DatasetReader*, *point_x: Union[float, int, list, numpy.ndarray]*, *point_y: Union[float, int, list, numpy.ndarray]*, *sample_outside_extent: bool = True*, *sample_all_bands: bool = False*) → Union[list, float]

Sampling the value of a rasterio object at a given point within the extent of the raster

Parameters

- **raster** (*rasterio.io.DatasetReader*) – Rasterio Object containing the height information
- **point_x** (*list, np.ndarray, float, int*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`
- **point_y** (*list, np.ndarray, float, int*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`
- **sample_outside_extent** (*bool*) – Allow sampling outside the extent of the rasterio object. Options include: `True` or `False`, default set to `True`
- **sample_all_bands** (*bool*) – Allow sampling from all bands returning Options include: `True` or `False`, default set to `False`

Returns **sample** – Value/s of the raster at the provided position/s

Return type list, float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling values from a rasterio object
>>> value = gg.raster.sample_from_rasterio(raster=raster, point_x=500, point_y=500)
>>> value
561.646728515625
```

See also:

sample_from_array Sample values from NumPy array

sample_randomly Sample randomly from rasterio object or NumPy array

sample_orientations Sample orientations from raster

sample_interfaces Sample interfaces from raster

gemgis.raster.**sample_interfaces**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *point_x: Union[float, int, list, numpy.ndarray] = None*, *point_y: Union[float, int, list, numpy.ndarray] = None*, *random_samples: int = None*, *formation: str = None*, *seed: int = None*, *sample_outside_extent: bool = False*, *crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*) → geopandas.geodataframe.GeoDataFrame

Sampling interfaces from a raster

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster or arrays from which points are being sampled
- **extent** (*List[Union[int, float]]*) – List containing the extent of the raster (minx, maxx, miny, maxy), e.g. `extent=[0, 972, 0, 1069]`
- **point_x** (*Union[float, int, list, np.ndarray]*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`, default is `None`
- **point_y** (*Union[float, int, list, np.ndarray]*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`, default is `None`
- **random_samples** (*int*) – Number of random samples to be drawn, e.g. `random_samples=10`, default is `None`
- **formation** (*str*) – Name of the formation the raster belongs to, e.g. `formation='Layer1'`, default is `None`
- **seed** (*int*) – Integer to set a seed for the drawing of random values, e.g. `seed=1`, default is `None`
- **sample_outside_extent** (*bool*) – Allow sampling outside the extent of the rasterio object. Options include: `True` or `False`, default is `False`
- **crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Coordinate reference system to be passed to the GeoDataFrame upon creation, e.g. `crs='EPSG:4647`

Returns **gdf** – GeoDataFrame containing the sampled interfaces

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling interfaces from an array or rasterio object
>>> gdf = gg.raster.sample_interfaces(raster=raster, point_x=500, point_y=500)
>>> gdf
    X       Y       Z       geometry
0   500.00  500.00  561.65  POINT (500.000 500.000)
```

See also:

sample_from_array Sample values from NumPy array

sample_from_rasterio Sample values from rasterio object

sample_randomly Sample randomly from rasterio object or NumPy array

sample_orientations Sample orientations from raster

gemgis.raster.**sample_orientations**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *extent: List[Union[int, float]] = None*, *point_x: Union[float, int, list, numpy.ndarray] = None*, *point_y: Union[float, int, list, numpy.ndarray] = None*, *random_samples: int = None*, *formation: str = None*, *seed: int = None*, *sample_outside_extent: bool = False*, *crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*) → geopandas.geodataframe.GeoDataFrame

Sampling orientations from a raster

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – Raster or arrays from which points are being sampled
- **extent** (*List[Union[int, float]]*) – List containing the extent of the raster (minx, maxx, miny, maxy), e.g. `extent=[0, 972, 0, 1069]`
- **point_x** (*Union[float, int, list, np.ndarray]*) – Object containing the x coordinates of a point or points at which the array value is obtained, e.g. `point_x=100`, default is `None`
- **point_y** (*Union[float, int, list, np.ndarray]*) – Object containing the y coordinates of a point or points at which the array value is obtained, e.g. `point_y=100`, default is `None`
- **random_samples** (*int*) – Number of random samples to be drawn, e.g. `random_samples=10`, default is `None`
- **formation** (*str*) – Name of the formation the raster belongs to, e.g. `formation='Layer1'`, default is `None`
- **seed** (*int*) – Integer to set a seed for the drawing of random values, e.g. `seed=1`, default is `None`
- **sample_outside_extent** (*bool*) – Allow sampling outside the extent of the rasterio object. Options include: `True` or `False`, default is `False`

- **crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Coordinate reference system to be passed to the GeoDataFrame upon creation, e.g. `crs='EPSG:4647`

Returns **gdf** – GeoDataFrame containing the sampled interfaces

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling orientations from an array or rasterio object
>>> gdf = gg.raster.sample_orientations(raster=raster, point_x=500, point_y=500)
>>> gdf
    X       Y       Z       geometry                dip     azimuth polarity
0   500.00  500.00  561.65  POINT (500.000 500.000) 19.26   145.55  1
```

See also:

sample_from_array Sample values from NumPy array

sample_from_rasterio Sample values from rasterio object

sample_randomly Sample randomly from rasterio object or NumPy array

sample_interfaces Sample interfaces from raster

gemgis.raster.**sample_randomly**(*raster: Union[numpy.ndarray, rasterio.io.DatasetReader]*, *n: int = 1*, *extent: Optional[Sequence[float]] = None*, *seed: int = None*) → tuple

Sampling randomly from a raster (array or rasterio object) using sample_from_array or sample_from_rasterio and a randomly drawn point within the array/raster extent

Parameters

- **raster** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy Array or rasterio object containing the raster values
- **n** (*int*) – Number of samples to be drawn, e.g. `n=10`, default `1`
- **extent** (*Optional[Sequence[float]]*) – List containing the values for the extent of the array (minx,maxx,miny,maxy), default is `None`, e.g. `extent=[0, 972, 0, 1069]`
- **seed** (*int*) – Seed for the random variable for reproducibility, e.g. `seed=1`, default is `None`

Returns **sample** – Float of sampled raster value and list containing the x- and y-points of the point where sample was drawn

Return type tuple

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Sampling randomly from an array or rasterio object
>>> value = gg.raster.sample_randomly(raster=raster, n=1)
>>> value
(617.0579833984375, [529.5110732824818, 717.7358438674542])
```

See also:

sample_from_array Sample values from NumPy array

sample_from_rasterio Sample values from rasterio object

sample_orientations Sample orientations from raster

sample_interfaces Sample interfaces from raster

gemgis.raster.**save_as_tiff**(*raster: numpy.ndarray*, *path: str*, *extent: Union[List[Union[int, float]], Tuple[Union[int, float]]]*, *crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*, *nodata: Union[float, int] = None*, *transform=None*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Saving a np.array as tif file

Parameters

- **array** (*np.ndarray*) – Array containing the raster values
- **path** (*string*) – Path and name of the file, e.g. `path='mesh.msh'`
- **extent** (*Union[List[Union[int, float]], Tuple[Union[int, float]]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – CRS of the saved raster, e.g. `crs='EPSG:4647'`
- **nodata** (*Union[float, int]*) – Nodata value of the raster, e.g. `nodata=9999.0`, default `None`
- **transform** – Transform of the data, default is `None`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default is `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Defining raster extent and CRS
>>> extent = [0, 972, 0, 1069]
>>> crs = 'EPSG:4326'
```

```
>>> # Saving raster as tiff
>>> gg.raster.save_as_tiff(raster=raster.read(1), path='raster_saved.tif',␣
↪extent=extent, crs=crs)
Raster successfully saved
```

9.1.6 gemgis.utils module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

`gemgis.utils.`**`assign_properties`**(*lith_block: numpy.ndarray*, *property_dict: dict*) → numpy.ndarray

Replacing lith block IDs with physical properties

Parameters

- **lith_block** (*np.ndarray*) – GemPy lith block array containing the surface IDs
- **property_array** (*dict*) – Dict containing the property values mapped to a surface ID

Returns **property_block** – Array containing the properties

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and lith_block plus reshaping
>>> import gemgis as gg
>>> import numpy as np
>>> lith_block = np.load('lith_block.npy').reshape(50,50,50)
```

```
>>> # Defining properties
>>> density_values = [0.1, 2.5, 3.0]
```

```
>>> # Creating dict
>>> density_dict = {k: v for k,v in zip(np.unique(np.round(lith_block)), density_
↪values)}
>>> density_dict
{1.0: 0.1, 2.0: 2.5, 3.0: 3.0}
```

```
>>> # Assign properties
>>> property_block = gg.utils.assign_properties(lith_block=lith_block, property_
↪dict=property_dict)
```

gemgis.utils.**build_style_dict**(*classes: dict*) → dict

Building a style dict based on extracted style classes

Parameters **classes** (*dict*) – Dict containing the styles of objects

Returns **styles** – Dict containing styles for different objects

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> column, classes = gg.utils.parse_categorized_qml(qml_name='style.qml')
>>>column
'formation'
```

```
>>> # Inspecting classes
>>> classes
{'Sand1': {'border_width_map_unit_scale': '3x:0,0,0,0,0,0',
'color': '179,90,42,255',
'joinstyle': 'bevel',
'offset': '0,0',
'offset_map_unit_scale': '3x:0,0,0,0,0,0',
'offset_unit': 'MM',
'outline_color': '102,51,24,255',
'outline_style': 'solid',
'outline_width': '0.26',
'outline_width_unit': 'MM',
'style': 'solid'},....}
```

```
>>> # Creating Style Dict
>>> style_dict = gg.utils.build_style_dict(classes=classes)
>>> style_dict
{'Sand1': {'color': '#b35a2a',
'color_rgb': [179, 90, 42],
'opacity': 1.0,
'weight': 0.26,
'fillColor': '#b35a2a',
'fillOpacity': 1.0},...}
```

See also:

parse_categorized_qml Reading the contents of a QGIS Style file (qml)

load_surface_colors Loading surface colors as list

create_surface_color_dict Creating dict with colors for each formation

gemgis.utils.**calculate_lines**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, pandas.core.frame.DataFrame]*, *increment: Union[float, int]*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*) → geopandas.geodataframe.GeoDataFrame

Function to interpolate strike lines

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing existing strike lines
- **increment** (*Union[float, int]*) – Increment between the strike lines, e.g. `increment=50`
- **xcol** (*str*) – Name of X column, e.g. `x='X'`
- **ycol** (*str*) – Name of X column, e.g. `y='Y'`
- **zcol** (*str*) – Name of Z column, e.g. `z='Z'`

Returns **lines** – GeoDataFrame with interpolated strike lines

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='lines5_strike.shp')
>>> gdf
    id  Z   formation   geometry
0   7   0   Coal1   LINESTRING (1642.839 2582.579, 2829.348 2205.937)
1   6   150 Coal1   LINESTRING (1705.332 1759.201, 2875.795 1406.768)
2   5   200 Coal1   LINESTRING (1017.766 1722.234, 2979.938 1137.003)
3   4   250 Coal1   LINESTRING (99.956 1763.424, 765.837 1620.705,...
4   3   200 Coal1   LINESTRING (1078.147 1313.501, 2963.048 752.760)
```

```
>>> gdf_interpolated = gg.utils.calculate_lines(gdf=gdf, increment=50)
```

gemgis.utils.**calculate_number_of_isopoints**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, pandas.core.frame.DataFrame]*, *increment: Union[float, int]*, *zcol: str = 'Z'*) → int

Creating the number of isopoints to further interpolate strike lines

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing existing strike lines
- **increment** (*Union[float, int]*) – Increment between the strike lines, e.g. `increment=50`
- **zcol** (*string*) – Name of z column, e.g. `z='Z'`, default is `'Z'`

Returns **number** – Number of isopoints

Return type int

New in version 1.0.x.

Example

```
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='lines5_strike.shp')
>>> gdf
    id  Z   formation   geometry
0   7   0   Coal1   LINESTRING (1642.839 2582.579, 2829.348 2205.937)
1   6   150 Coal1   LINESTRING (1705.332 1759.201, 2875.795 1406.768)
2   5   200 Coal1   LINESTRING (1017.766 1722.234, 2979.938 1137.003)
3   4   250 Coal1   LINESTRING (99.956 1763.424, 765.837 1620.705,...
4   3   200 Coal1   LINESTRING (1078.147 1313.501, 2963.048 752.760)
```

```
>>> number = gg.utils.calculate_number_of_isopoints(gdf=gdf, increment=50)
>>> number
2
```

See also:

get_nearest_neighbor Getting the nearest neighbor to a point

gemgis.utils.**convert_crs_seismic_data**(*path_in: str*, *path_out: str*, *crs_in: Union[str, pyproj.crs.crs.CRS]*, *crs_out: Union[str, pyproj.crs.crs.CRS]*, *cdpx: int = 181*, *cdpy: int = 185*, *vert_domain: str = 'TWT'*, *coord_scalar: int = None*)

Convert CDP coordinates of seismic data to a new CRS.

Parameters

- **path_in** (*str*) – Path to the original seismic data, e.g. `path_in='seismic.sgy'`.
- **path_out** (*str*) – Path to the converted seismic data, e.g. `path_out='seismic_converted.sgy'`.

- **crs_in** (*str, pyproj.crs.crs.CRS*) – Coordinate reference system of the original seismic data.
- **crs_out** (*str, pyproj.crs.crs.CRS*) – Coordinate reference system of the converted seismic data.
- **cdpx** (*int*) – Byte position for the X coordinates, default is `cdpx=181`.
- **cdpy** (*int*) – Byte position for the Y coordinates, default is `cdpx=185`.
- **vert_domain** (*str*) – Vertical sampling domain. Options include `'TWT'` and `'DEPTH'`, default is `vert_domain='TWT'`.
- **coord_scalar** (*int*) – Coordinate scalar value to set if *NaN* columns are returned, default is *coord_scalar=None*.

New in version 1.1.1.

gemgis.utils.**convert_location_dict_to_gdf**(*location_dict: dict*) → geopandas.geodataframe.GeoDataFrame

Converting a location dict to a GeoDataFrame

Parameters **location_dict** (*dict*) – Dict containing the name of the location and the coordinates

Returns **gdf** – GeoDataFrame containing the location name and the coordinates of the location

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Creating a dict with coordinates
>>> coordinates_dict = gg.utils.get_locations(names = ['Aachen', 'Berlin', 'München
↪', 'Hamburg', 'Köln'], crs='EPSG:4647')
```

```
>>> # Converting dict to GeoDataFrame
>>> gdf = gg.utils.convert_location_dict_to_gdf(location_dict=coordinates_dict)
>>> gdf
    City    X           Y           geometry
0       Aachen  32294411.33 5629009.36  POINT (32294411.335 5629009.357)
1       Berlin  32797738.56 5827603.74  POINT (32797738.561 5827603.740)
2       München 32691595.36 5334747.27  POINT (32691595.356 5334747.274)
3       Hamburg 32566296.25 5933959.96  POINT (32566296.251 5933959.965)
4       Köln    32356668.82 5644952.10  POINT (32356668.818 5644952.100)
```

gemgis.utils.**convert_to_gempy_df**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: Union[rasterio.io.DatasetReader, numpy.ndarray] = None*, *extent: List[Union[int, float]] = None*) → pandas.core.frame.DataFrame

Converting a GeoDataFrame into a Pandas DataFrame ready to be read in for GemPy

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing spatial information, formation names and orientation values

- **dem** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy ndarray or rasterio object containing the height values
- **extent** (*List[Union[float,int]*) – List containing the extent of the np.ndarray, must be provided in the same CRS as the gdf, e.g. `extent=[0, 972, 0, 1069]`

Returns **df** – Interface or orientations DataFrame ready to be read in for GemPy

Return type pd.DataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0   None    Ton         POINT (19.150 293.313)
1   None    Ton         POINT (61.934 381.459)
2   None    Ton         POINT (109.358 480.946)
3   None    Ton         POINT (157.812 615.999)
4   None    Ton         POINT (191.318 719.094)
```

```
>>> # Loading Digital Elevation Model
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Defining extent
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Converting GeoDataFrame to DataFrame
>>> df = gg.utils.convert_to_gempy_df(gdf=gdf, dem=dem, extent=extent)
>>> df
    formation   X       Y       Z
0   Ton         19.15   293.31  364.99
1   Ton         61.93   381.46  400.34
2   Ton         109.36  480.95  459.55
3   Ton         157.81  616.00  525.69
4   Ton         191.32  719.09  597.63
```

gemgis.utils.**convert_to_petrel_points_with_attributes**(*mesh: pyvista.core.pointset.PolyData*, *path: str*, *crs: Optional[Union[str, pyproj.crs.crs.CRS]] = None*, *target_crs: Optional[Union[str, pyproj.crs.crs.CRS]] = None*)

Function to convert vertices of a PyVista Mesh to Petrel Points with Attributes

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista Mesh to be converted to points

- **path** (*str*) – Path to store the converted points, e.g. `path='project/'`
- **crs** (*str, pyproj.crs.crs.CRS, type(None)*) – Coordinate reference system for the GeoDataFrame, e.g. `crs='EPSG:4326'`, default is `None`
- **target_crs** (*str, pyproj.crs.crs.CRS, type(None)*) – Target coordinate reference system if coordinates of points should be reprojected, e.g. `crs='EPSG:4326'`, default is `None`

New in version 1.0.x.

gemgis.utils.**create_polygon_from_location**(*coordinates*) → shapely.geometry.polygon.Polygon

Creating Shapely Polygon from bounding box coordinates

Parameters **coordinates** (*geopy.location.Location*) – GeoPy location object

Returns **polygon** – Shapely Polygon marking the bounding box of the coordinate object

Return type shapely.geometry.polygon.Polygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries and get location object
>>> import gemgis as gg
>>> location = gg.utils.get_location_coordinate(name='Aachen')
>>> location
Location(Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland, (50.776351,␣
↪6.083862, 0.0))
```

```
>>> # Creating polygon from location bounds
>>> polygon = gg.utils.create_polygon_from_location(coordinates=location)
>>> polygon.wkt
'POLYGON ((50.8572449 5.9748624, 50.8572449 6.2180747, 50.6621373 6.2180747, 50.
↪6621373 5.9748624, 50.8572449 5.9748624))'
```

See also:

transform_location_coordinate Transforming location coordinate to another CRS

get_location_coordinate Get GeoPy Location Object

get_locations Get location information for a list of city names

gemgis.utils.**create_surface_color_dict**(*path: str*) → dict

Creating GemPy surface color dict from a QML file

Parameters **path** (*str*) – Path to the qml file, e.g. `qml_name='style.qml'`

Returns **surface_color_dict** – Dict containing the surface color values for GemPy

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> surface_colors_dict = gg.utils.create_surface_color_dict(path='style.qml')
>>> surface_colors_dict
{'Sand1': '#b35a2a', 'Sand2': '#b35a2a', 'Ton': '#525252'}
```

See also:

build_style_dict Building style dictionairy from loaded style file

parse_categorized_qml Reading the contents of a QGIS Style file (qml)

load_surface_colors Loading surface colors as list

gemgis.utils.**create_virtual_profile**(*names_surfaces: list*, *surfaces: list*, *borehole: pyvista.core.pointset.PolyData*, *first_point: bool = False*) → pandas.core.frame.DataFrame

Function to filter and sort the resulting well tops

Parameters

- **names_surfaces** (*list*) – List of the names of the calculated GemPy surfaces, e.g. `names_surfaces=['Layer1', 'Layer2']`
- **surfaces** (*list*) – List of calculated GemPy surfaces, e.g. `surfaces=['Layer1', 'Layer2']`
- **borehole** (*pv.core.pointset.PolyData*) – Coordinates of the bottom of the well
- **first_point** (*bool*) – Returns intersection of first point only. Options include: `True` or `False`, default set to `False`

Returns **df** – DataFrame containing the well tops

Return type pd.DataFrame

New in version 1.0.x.

gemgis.utils.**create_zmap_grid**(*surface: pyvista.core.pointset.PolyData*, *cell_width: int*, *comments: str = ''*, *name: str = 'ZMAP_Grid'*, *z_type: str = 'GRID'*, *nodes_per_line: int = 5*, *field_width: int = 15*, *nodata: Union[int, float] = - 9999.0*, *nodata2: Union[int, float, str] = ''*, *decimal_places: int = 5*, *start_column: int = 1*)

Function to write data to ZMAP Grid, This code is heavily inspired by https://github.com/abduhbm/zmapio

Parameters

- **surface** (*pv.core.pointset.PolyData*) – PyVista mesh
- **cell_width** (*int*) – Width of grid cell, e.g. `cell_width=50`
- **comments** (*str*) – Comments written to the ZMAP File, e.g. `comments='Project: Einstein'`, default is `''`
- **name** (*str*) – Name of the ZMAP File, e.g. `name='ZMAP_Grid'`, default is `'ZMAP_Grid'`
- **z_type** (*str*) – ZMAP Grid Type, e.g. `z_type='GRID'`, default is `'GRID'`
- **nodes_per_lines** (*int*) – Number of values per line, e.g. `nodes_per_line=5`, default is `5`

- **field_width** (*int*) – Width of each field, e.g. `field_width=15`, default is `15`
- **nodata** (*Union[int, float]*) – No data value, e.g. `nodata=-9999`, default is `-9999`
- **nodata2** (*Union[int, float, str]*) – No data value, e.g. `nodata2=-9999`, default is `''`
- **decimal_places** (*int*) – Number of Decimal Places, e.g. `decimal_places=5`, default is `5`
- **start_column** (*int*) – Number of the start column, e.g. `start_column=1`, default is `1`

Returns **lines** – String containing the ZMAP Grid Data

Return type str

New in version 1.0.x.

gemgis.utils.**extract_zmap_data**(*surface: pyvista.core.pointset.PolyData*, *cell_width: int*, *nodata: Union[float, int] = - 9999*)

Function to extract a meshgrid of values from a PyVista mesh

Parameters

- **surface** (*pv.core.pointset.PolyData*) – PyVista mesh
- **cell_width** (*int*) – Width of grid cell, e.g. `cell_width=50`
- **nodata** (*Union[float, int]*) – No data value, e.g. `nodata=-9999`, default is `-9999`

New in version 1.0.x.

gemgis.utils.**get_cdp_linestring_of_seismic_data**(*path_in: str*, *crs_in: Union[str, pyproj.crs.crs.CRS]*, *cdpx: int = 181*, *cdpy: int = 185*, *vert_domain: str = 'TWT'*)

Extracting the path of the seismic data as LineString.

Parameters

- **path_in** (*str*) – Path to the original seismic data, e.g. `path_in='seismic.sgy'`.
- **crs_in** (*str, pyproj.crs.crs.CRS*) – Coordinate reference system of the original seismic data.
- **cdpx** (*int*) – Byte position for the X coordinates, default is `cdpx=181`.
- **cdpy** (*int*) – Byte position for the Y coordinates, default is `cdpx=185`.
- **vert_domain** (*str*) – Vertical sampling domain. Options include `'TWT'` and `'DEPTH'`, default is `vert_domain='TWT'`.

Returns GeoDataFrame containing the surface path of the seismic data as LineString.

Return type gpd.GeoDataFrame

New in version 1.1.1.

gemgis.utils.**get_cdp_points_of_seismic_data**(*path_in: str*, *crs_in: Union[str, pyproj.crs.crs.CRS]*, *cdpx: int = 181*, *cdpy: int = 185*, *vert_domain: str = 'TWT'*, *filter: int = None*, *n_meter: Union[int, float] = None*)

Extracting the path of the seismic data as LineString.

Parameters

- **path_in** (*str*) – Path to the original seismic data, e.g. `path_in='seismic.sgy'`.

- **crs_in** (*str, pyproj.crs.crs.CRS*) – Coordinate reference system of the original seismic data.
- **cdpx** (*int*) – Byte position for the X coordinates, default is `cdpx=181`.
- **cdpy** (*int*) – Byte position for the Y coordinates, default is `cdpx=185`.
- **vert_domain** (*str*) – Vertical sampling domain. Options include `'TWT'` and `'DEPTH'`, default is `vert_domain='TWT'`.
- **filter** (*int*) – Filtering the points to only return every n-th point, e.g. `filter=100` to return only every 100-th point.
- **n_meter** (*int, float*) – Parameter to select a point along the line every n-th meter.

Returns GeoDataFrame containing the CDPs as Points.

Return type gpd.GeoDataFrame

New in version 1.1.1.

gemgis.utils.**get_location_coordinate**(*name: str*)

Obtaining coordinates of a given city

Parameters **name** (*str*) – Name of the location, e.g. `name='Aachen'`

Returns **coordinates** – GeoPy Location object

Return type geopy.location.Location

New in version 1.0.x.

Example

```
>>> # Loading Libraries and get location object
>>> import gemgis as gg
>>> location = gg.utils.get_location_coordinate(name='Aachen')
>>> location
Location(Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland, (50.776351,␣
↪6.083862, 0.0))
```

See also:

transform_location_coordinate Transforming location coordinate to another CRS

create_polygon_from_location Create Shapely Polygon from GeoPy Location Object bounds

get_locations Get location information for a list of city names

gemgis.utils.**get_locations**(*names: Union[list, str]*, *crs: Union[str, pyproj.crs.crs.CRS] = 'EPSG:4326'*) → dict

Obtaining coordinates for one city or a list of given cities. A CRS other than 'EPSG:4326' can be passed to transform the coordinates

Parameters

- **names** (*Union[list, str]*) – List of cities or single city name, e.g. `names=['Aachen', 'Cologne', 'Munich', 'Berlin']`
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – CRS that coordinates will be transformed to, e.g. `crs='EPSG:4647'`, default is the GeoPy crs `'EPSG:4326'`

Returns **location_dict** – Dict containing the addresses and coordinates of the selected cities

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and get location objects
>>> import gemgis as gg
>>> names = ['Aachen', 'Cologne', 'Munich', 'Berlin']
>>> location_dict = gg.utils.get_locations(names=names, crs='EPSG:4647')
>>> location_dict
{'Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland': (32294411.
↪33488576,  5629009.357074926),
'Köln, Nordrhein-Westfalen, Deutschland': (32356668.818424627, 5644952.099932303),
'München, Bayern, Deutschland': (32691595.356409974, 5334747.274305081),
'Berlin, 10117, Deutschland': (32797738.56053437, 5827603.740024588)}
```

See also:

transform_location_coordinate Transforming location coordinate to another CRS

get_location_coordinate Get GeoPy Location Object

create_polygon_from_location Create Shapely Polygon from GeoPy Location Object bounds

gemgis.utils.**get_nearest_neighbor**(*x: numpy.ndarray*, *y: numpy.ndarray*) → numpy.int64

Function to return the index of the nearest neighbor for a given point Y

Parameters

- **x** (*np.ndarray*) – Array with coordinates of a set of points, e.g. `x=np.array([(0,0), (10,10)])`
- **y** (*np.ndarray*) – Array with coordinates for point y, e.g. `y=np.array([2,2])`

Returns **index** – Index of the nearest neighbor of point set X to point Y

Return type np.int64

New in version 1.0.x.

Example

```
>>> import gemgis as gg
>>> import numpy as np
>>> x = np.array([(0,0), (10,10)])
>>> x
array([[ 0,  0],
       [10, 10]])
```

```
>>> y = np.array([2,2])
>>> y
array([2, 2])
```

```
>>> index = gg.utils.get_nearest_neighbor(x=x, y=y)
>>> index
0
```

See also:

calculate_number_of_isopoints Calculating the number of isopoints that are necessary to interpolate lines

gemgis.utils.**getfeatures**(*extent: Optional[List[Union[int, float]]]*, *crs_raster: Union[str, dict]*, *crs_bbox: Union[str, dict]*, *bbox: shapely.geometry.polygon.Polygon = None*) → list

Creating a list containing a dict with keys and values to clip a raster

Parameters

- **extent** (*Union[List[Union[int, float]]*) – List of bounds (minx,maxx, miny, maxy), e.g. `extent=[0, 972, 0, 1069]`
- **crs_raster** (*Union[str, dict]*) – String or dict containing the raster crs, e.g. `crs='EPSG:4647'`
- **crs_bbox** (*Union[str, dict]*) – String or dict containing the bbox crs, e.g. `crs='EPSG:4647'`
- **bbox** (*shapely.geometry.polygon.Polygon*) – Shapely polygon defining the bbox used to get the coordinates, , e.g. `polygon = Polygon([(0, 0), (0, 10), (10, 10), (10, 0)])`

Returns **data** – List containing a dict with keys and values to clip raster

Return type list

New in version 1.0.x.

gemgis.utils.**interpolate_strike_lines**(*gdf: geopandas.geodataframe.GeoDataFrame*, *increment: Union[float, int]*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*) → geopandas.geodataframe.GeoDataFrame

Interpolating strike lines to calculate orientations

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing existing strike lines
- **increment** (*Union[float, int]*) – Increment between the strike lines, e.g. `increment=50`
- **xcol** (*str*) – Name of X column, e.g. `x='X'`
- **ycol** (*str*) – Name of X column, e.g. `y='Y'`
- **zcol** (*str*) – Name of Z column, e.g. `z='Z'`

Returns **gdf_out** – GeoDataFrame containing the existing and interpolated strike lines

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

gemgis.utils.**load_surface_colors**(*path: str*, *gdf: geopandas.geodataframe.GeoDataFrame*) → List[str]

Loading surface colors from a QML file and storing the color values as list to be displayed with GeoPandas plots

Parameters

- **path** (*str*) – Path to the qml file, e.g. `qml_name='style.qml'`
- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame of which objects are supposed to be plotted, usually loaded from a polygon/line shape file

Returns **cols** – List of color values for each surface

Return type List[str]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
```

```
>>> # Loading surface colors
>>> colors = gg.utils.load_surface_colors(path='style.qml', gdf=gdf)
>>> colors
['#b35a2a', '#b35a2a', '#525252']
```

See also:

build_style_dict Building style dictionairy from loaded style file

parse_categorized_qml Reading the contents of a QGIS Style file (qml)

create_surface_color_dict Creating dict with colors for each formation

gemgis.utils.**open_mpk**(*path_in: str*)

Read ArcGIS .mpk file and return vector and raster data.

Parameters **path_in** (*str*) – Path to the .mpk file, e.g. `path='file.mpk'`

Returns

- **dict_vector_data** (*dict*) – Dictionary containing the extracted vector data.
- **dict_raster_data** (*dict*) – Dictionary containing the extracted raster data.

Example

gemgis.utils.**parse_categorized_qml**(*qml_name: str*) → tuple

Parsing a QGIS style file to retrieve surface color values

Parameters **qml_name** (*str*) – Path to the QML file, e.g. `qml_name='style.qml'`

Returns

- **column** (*str*) – Variable indicating after which formation the objects were colored (i.e. `'formation'`)
- **classes** (*dict*) – Dict containing the style attributes for all available objects

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> column, classes = gg.utils.parse_categorized_qml(qml_name='style.qml')
>>> column
'formation'
```

```
>>> # Inspecting classes
>>> classes
{'Sand1': {'border_width_map_unit_scale': '3x:0,0,0,0,0,0',
'color': '179,90,42,255',
'joinstyle': 'bevel',
'offset': '0,0',
'offset_map_unit_scale': '3x:0,0,0,0,0,0',
'offset_unit': 'MM',
'outline_color': '102,51,24,255',
'outline_style': 'solid',
'outline_width': '0.26',
'outline_width_unit': 'MM',
'style': 'solid'},....}
```

See also:

build_style_dict Building style dictionairy from loaded style file

load_surface_colors Loading surface colors as list

create_surface_color_dict Creating dict with colors for each formation

gemgis.utils.**ray_trace_multiple_surfaces**(*surfaces: list*, *borehole_top: Union[numpy.ndarray, list]*, *borehole_bottom: Union[numpy.ndarray, list]*, *first_point: bool = False*) → list

Function to return the depth of multiple surfaces in one well using PyVista ray tracing

Parameters

- **surfaces** (*list*) – List of calculated GemPy surfaces
- **borehole_top** – Coordinates of the top of the well
- **borehole_bottom** – Coordinates of the bottom of the well
- **first_point** (*bool*) – Returns intersection of first point only

Returns **intersections** – List of intersections

Return type list

New in version 1.0.x.

gemgis.utils.**ray_trace_one_surface**(*surface: Union[pyvista.core.pointset.PolyData, pyvista.core.pointset.UnstructuredGrid]*, *origin: Union[numpy.ndarray, list]*, *end_point: Union[numpy.ndarray, list]*, *first_point: bool = False*) → tuple

Function to return the depth of one surface in one well using PyVista ray tracing

Parameters

- **surface** (*Union[pv.core.pointset.PolyData, pv.core.pointset.UnstructuredGrid]*) – Calculated or clipped GemPy surface
- **origin** – Coordinates of the top of the well
- **end_point** – Coordinates of the bottom of the well
- **first_point** (*bool*) – Returns intersection of first point only

Returns **intersection_points, intersection_cells** – Location of the intersection points, Indices of the intersection cells

Return type tuple

New in version 1.0.x.

gemgis.utils.**read_csv_as_gdf**(*path: str*, *crs: Union[str, pyproj.crs.crs.CRS]*, *x: str = 'X'*, *y: str = 'Y'*, *z: str = None*, *delimiter: str = ','*) → geopandas.geodataframe.GeoDataFrame

Reading CSV files as GeoDataFrame

Parameters

- **path** (*str*) – Path of the CSV files, e.g. `path='file.csv'`
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – CRS of the spatial data, e.g. `crs='EPSG:4647'`
- **x** (*str*) – Name of the X column, e.g. `x='X'`, default is `'X'`
- **y** (*str*) – Name of the Y column, e.g. `y='Y'`, default is `'Y'`
- **z** (*str*) – Name of the Z column, e.g. `z='Z'`, default is `'Z'`
- **delimiter** (*str*) – Delimiter of CSV file, e.g. `delimiter=','`, default is ‘,’

Returns **gdf** – GeoDataFrame of the CSV data

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File as GeoDataFrame
>>> import gemgis as gg
>>> gdf = gg.utils.read_csv_as_gdf(path='file.csv')
>>> gdf
    id      formation   geometry
0   None    Ton         POINT (19.150 293.313)
1   None    Ton         POINT (61.934 381.459)
2   None    Ton         POINT (109.358 480.946)
3   None    Ton         POINT (157.812 615.999)
4   None    Ton         POINT (191.318 719.094)
```

gemgis.utils.**rotate_gempy_input_data**(*extent: Union[numpy.ndarray, shapely.geometry.polygon.Polygon, geopandas.geodataframe.GeoDataFrame]*, *interfaces: Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *orientations: Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*, *zmin: Union[float, int] = None*, *zmax: Union[float, int] = None*, *rotate_reverse_direction: bool = False*, *return_extent_gdf: bool = False*, *manual_rotation_angle: Union[float, int] = None*)

Function to rotate the GemPy Input Data horizontally or vertically

Parameters

- **extent** (*np.ndarray, shapely.geometry.Polygon, gpd.geodataframe.GeoDataFrame*) – Extent of the Model
- **interfaces** (*pd.DataFrame, gpd.geodataframe.GeoDataFrame*) – Interface points for the GemPy Model
- **orientations** (*pd.DataFrame, gpd.geodataframe.GeoDataFrame*) – Orientations for the GemPy Model
- **zmin** (*float, int*) – Lower Z limit of the GemPy Model, e.g. `zmin=-1000`, default is `None`
- **zmax** (*float, int*) – Upper Z limit of the GemPy Model, e.g. `zmax=1000`, default is `None`
- **rotate_reverse_direction** (*bool*) – Rotating the model the other direction. Options include: `True` or `False`, default set to `False`
- **return_extent_gdf** (*bool*) – Returning the extent GeoDataFrame. Options include: `True` or `False`, default set to `False`
- **manual_rotation_angle** (*float, int*) – Angle to manually rotate the data, e.g. `manual_rotation_angle=45`, default is `None`

Returns

- **extent** (*list*) – New GemPy Model extent, e.g. `extent=[0, 972, 0, 1069]`
- **interfaces_rotated** (*pd.DataFrame, gpd.geodataframe.GeoDataFrame*) – Rotated interfaces for the structural modeling in GemPy
- **orientations_rotated** (*pd.DataFrame, gpd.geodataframe.GeoDataFrame*) – Rotated orientations for the structural modeling in GemPy

New in version 1.1.

gemgis.utils.**save_zmap_grid**(*zmap_grid: list*, *path: str = 'ZMAP_Grid.dat'*)

Function to save ZMAP Grid information to file

Parameters

- **zmap_grid** (*list*) – List of strings containing the ZMAP Data
- **path** (*str*) – Path and filename to store the ZMAP Grid, e.g. `path='ZMAP_Grid.dat'`, default is `'ZMAP_Grid.dat'`

New in version 1.0.x.

gemgis.utils.**set_extent**(*minx: Union[int, float] = 0*, *maxx: Union[int, float] = 0*, *miny: Union[int, float] = 0*, *maxy: Union[int, float] = 0*, *minz: Union[int, float] = 0*, *maxz: Union[int, float] = 0*, *gdf: geopandas.geodataframe.GeoDataFrame = None*) → List[Union[int, float]]

Setting the extent for a model

Parameters

- **minx** (*Union[int, float]*) – Value defining the left border of the model, e.g. `minx=0`, default is `0`
- **maxx** (*Union[int, float]*) – Value defining the right border of the model, e.g. `max=972`, default is `0`
- **miny** (*Union[int, float]*) – Value defining the upper border of the model, e.g. `miny=0`, default is `0`
- **maxy** (*Union[int, float]*) – Value defining the lower border of the model, e.g. `maxy=1069`, default is `0`
- **minz** (*Union[int, float]*) – Value defining the top border of the model, e.g. `minz=0`, default is `0`
- **maxz** (*Union[int, float]*) – Value defining the bottom border of the model, e.g. `maxz=1000`, default is `0`
- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame from which bounds the extent will be set, default is `None`

Returns **extent** – List containing extent values

Return type List[Union[int, float]]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and setting the extent
>>> import gemgis as gg
>>> extent = gg.utils.set_extent(minx=0, maxx=972, miny=0, maxy=1069, minz=0,␣
↪maxz=1000)
>>> extent
[0, 972, 0, 1069, 0, 1000]
```

gemgis.utils.**set_resolution**(*x: int*, *y: int*, *z: int*) → List[int]

Setting the resolution for a model

Parameters

- **x** (*int*) – Value defining the resolution in X direction, e.g. `x=50`
- **y** (*int*) – Value defining the resolution in Y direction, e.g. `y=50`
- **z** (*int*) – Value defining the resolution in Z direction, e.g. `z=50`

Returns **resolution** – List containing resolution values

Return type List[int]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and setting the resolution
>>> import gemgis as gg
>>> res = gg.utils.set_resolution(x=50, y=50, z=50)
>>> res
[50, 50, 50]
```

gemgis.utils.**show_number_of_data_points**(*geo_model*)

Adding the number of Interfaces and Orientations to the GemPy Surface dataframe

Parameters **geo_model** (*gp.core.model.Project*) – GemPy geo_model object

Returns **geo_model.surfaces** – DataFrame-like object containing surface information and number of data points

Return type gempy.core.model.RestrictingWrapper

New in version 1.0.x.

Example

```
>>> # Loading Libraries and displaying surface DataFrame of a GemPy geo_model
>>> import gemgis as gg
>>> geo_model.surfaces
    surface     series          order_surfaces  color   id
0   Sand1       Strat_Series    1               #015482 1
1   Ton         Strat_Series    2               #9f0052 2
2   basement    Strat_Series    3               #ffbe00 3
```

```
>>> # Adding number of data points to DataFrame
>>> gg.utils.show_number_of_data_points(geo_model=geo_model)
    surface     series          order_surfaces  color   id  No. of Interfaces   No.␣
↪of Orientations
0   Sand1       Strat_Series    1               #015482 1   95                  0
1   Ton         Strat_Series    2               #9f0052 2   36                  8
2   basement    Strat_Series    3               #ffbe00 3   0                   0
```

gemgis.utils.**to_section_dict**(*gdf: geopandas.geodataframe.GeoDataFrame*, *section_column: str = 'section_name'*, *resolution: List[int] = None*) → dict

Converting custom sections stored in Shape files to GemPy section_dicts

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points or lines of custom sections
- **section_column** (*str*) – String containing the name of the column containing the section names, e.g. `section_column='section_name'`, default is `'section_name'`
- **List[int]** (*resolution* -) – List containing the x,y resolution of the custom section, e.g. `resolution=[80,80]`

Returns **section_dict** – Dict containing the section names, coordinates and resolution

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      geometry                    Section
0       None    POINT (695.467 3.226)       Section1
1       None    POINT (669.284 1060.822)        Section1
```

```
>>> # Creating Section dict
>>> section_dict = gg.utils.to_section_dict(gdf=gdf, section_column='Section')
>>> section_dict
{'Section1': ([695.4667461080886, 3.2262250771374283],
[669.2840030245482, 1060.822026058724], [100, 80])}
```

gemgis.utils.**transform_location_coordinate**(*coordinates*, *crs: Union[str, pyproj.crs.crs.CRS]*) → dict

Transforming coordinates of GeoPy Location

Parameters

- **coordinates** (*geopy.location.Location*) – GeoPy location object
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the target crs, e.g. `crs='EPSG:4647'`

Returns **result_dict** – Dict containing the location address and transformed coordinates

Return type dict

New in version 1.0.x.

Changed in version 1.1.7.

Updated to use the latest pyproj transformer

Example

```
>>> # Loading Libraries and get location object
>>> import gemgis as gg
>>> location = gg.utils.get_location_coordinate(name='Aachen')
>>> location
Location(Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland, (50.776351,␣
↪6.083862, 0.0))
```

```
>>> # Transforming location coordinates
>>> result_dict = gg.utils.transform_location_coordinate(coordinates=location, crs=
↪'EPSG:4647')
>>> result_dict
{'Aachen, Städteregion Aachen, Nordrhein-Westfalen, Deutschland': (32294411.
↪33488576, 5629009.357074926)}
```

See also:

get_location_coordinate Get GeoPy Location Object

create_polygon_from_location Create Shapely Polygon from GeoPy Location Object bounds

get_locations Get location information for a list of city names

9.1.7 gemgis.vector module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

gemgis.vector.**calculate_angle**(*linestring: shapely.geometry.linestring.LineString*) → float

Calculating the angle of a LineString to the vertical

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString consisting of two vertices, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **angle** – Angle of a LineString to the vertical

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Calculating the strike angle of the LineString
>>> angle = gg.vector.calculate_angle(linestring=linestring)
>>> angle
135.0
```

See also:

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

Note: The LineString must only consist of two points (start and end point)

gemgis.vector.**calculate_azimuth**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) → List[Union[int, float]]

Calculating the azimuth for an orientation Geodataframe represented by LineStrings

Parameters **gdf** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame or list containing the LineStrings of orientations

Returns **azimuth_list** – List containing the azimuth values of the orientation LineString

Return type List[Union[float, int]]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring1 = LineString([(0, 0), (20, -20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(0, 0), (20, -10)])
>>> linestring2.wkt
'LINESTRING (0 0, 20 -10)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
>>> gdf
    geometry
0       LINESTRING (0.0 0.0, 20.0 -20.0)
1       LINESTRING (0.0 0.0, 20.0 -10.0)
```

```
>>> # Calculating the azimuths of the LineStrings
>>> azimuths = gg.vector.calculate_azimuth(gdf=gdf)
>>> azimuths
[135.0, 116.56505117707799]
```

See also:

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.**calculate_coordinates_for_linestring_on_cross_sections**(*linestring: shapely.geometry.linestring.LineString*, *interfaces: shapely.geometry.linestring.LineString*)

Calculating the coordinates of vertices for a LineString on a straight cross section provided as Shapely LineString

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (20, 20)])`
- **interfaces** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the interfaces points digitized on a cross section, e.g. `interfaces = LineString([(2, -2), (5, -5)])`

Returns **points** – List of Shapely Points with real world coordinates of digitized points on cross section

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Calculating coordinates for LineString on cross section
>>> points = gg.vector.calculate_coordinates_for_linestring_on_cross_
↪sections(linestring=linestring, interfaces=interfaces)
>>> points
[<shapely.geometry.point.Point at 0x231e8dc4d60>,
<shapely.geometry.point.Point at 0x231e5d9b070>]
```

```
>>> # Inspecting the first element of the list
>>> points[0].wkt
'POINT (1.414213562373095 -1.414213562373095)'
```

```
>>> # Inspecting the second element of the list
>>> points[1].wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

gemgis.vector.**calculate_coordinates_for_linestrings_on_cross_sections**(*linestring: shapely.geometry.linestring.LineString*, *linestring_interfaces_list: List[shapely.geometry.linestring.LineString*] → List[shapely.geometry.point.Point]

Calculating the coordinates of vertices for LineStrings on a straight cross section provided as Shapely LineString

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`
- **linestring_interfaces_list** (*List[shapely.geometry.linestring.LineString]*) – List containing Shapely LineStrings representing interfaces on cross sections

Returns **points** – List containing Shapely Points with real world coordinates of the digitized interfaces on the cross section

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating list of LineStrings
>>> linestring_interfaces_list = [interfaces, interfaces]
```

```
>>> # Calculating coordinates for LineStrings on cross section
>>> points = gg.vector.calculate_coordinates_for_linestrings_on_cross_
↪sections(linestring=linestring, linestring_interfaces_list=linestring_interfaces_
↪list)
>>> points
[<shapely.geometry.point.Point at 0x231e8019730>,
 <shapely.geometry.point.Point at 0x231e801e400>,
 <shapely.geometry.point.Point at 0x231e80192e0>,
 <shapely.geometry.point.Point at 0x231e80191f0>]
```

```
>>> # Inspecting the first element of the list
>>> points[0].wkt
'POINT (1.414213562373095 -1.414213562373095)'
```

```
>>> # Inspecting the second element of the list
>>> points[1].wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

```
>>> # Inspecting the third element of the list
>>> points[2].wkt
'POINT (1.414213562373095 -1.414213562373095)'
```

```
>>> # Inspecting the fourth element of the list
>>> points[3].wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for one LineString on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

gemgis.vector.**calculate_coordinates_for_point_on_cross_section**(*linestring: shapely.geometry.linestring.LineString*, *point: Union[shapely.geometry.point.Point, Tuple[float, float]]*)

Calculating the coordinates for one point digitized on a cross section provided as Shapely LineString

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (20, 20)])`
- **point** (*Union[shapely.geometry.point.Point, Tuple[float, float]]*) – Shapely object or tuple of X and Y coordinates digitized on a cross section e.g. `point =`

Point(5, 0)

Returns **point** – Shapely Point with real world X and Y coordinates extracted from cross section LineString on Map

Return type shapely.geometry.point.Point

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating Point
>>> point = Point(5, 0)
>>> point.wkt
'POINT (5 0)'
```

```
>>> # Calculating real world coordinates for point on cross section
>>> point_xy = gg.vector.calculate_coordinates_for_point_on_cross_
↪section(linestring=linestring, point=point)
>>> point_xy.wkt
'POINT (3.535533905932737 -3.535533905932737)'
```

See also:

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for a LineString on a cross section

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

gemgis.vector.**calculate_dipping_angle_linestring**(*linestring: shapely.geometry.linestring.LineString*)

Calculating the dipping angle of a LineString digitized on a cross section

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString digitized on a cross section, e.g. `linestring = LineString([(0, 0), (20, 20)])`

Returns **dip** – Dipping angle of the LineString

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating dipping angle from LineString
>>> angle = gg.vector.calculate_dipping_angle_linestring(linestring=linestring)
>>> angle
45.0
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.**calculate_dipping_angles_linestrings**(*linestring_list: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*)

Calculating the dipping angles of LineStrings digitized on a cross section

Parameters **linestring_list** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame containing LineStrings or list of LineStrings

Returns **dipping_angles** – List containing the dipping angles of LineStrings

Return type List[float]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating list of LineStrings
>>> linestring_list = [linestring, linestring]
```

```
>>> # Calculating dipping angles for LineStrings
>>> angles = gg.vector.calculate_dipping_angles_linestrings(linestring_
↪list=linestring_list)
>>> angles
[45.0, 45.0]
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.**calculate_distance_linestrings**(*ls1: shapely.geometry.linestring.LineString*, *ls2: shapely.geometry.linestring.LineString*) → float

Calculating the minimal distance between two LineStrings

Parameters

- **ls1** (*shapely.geometry.linestring.LineString*) – LineString 1, e.g. `ls1 = LineString([(0, 0), (10, 10), (20, 20)])`
- **ls2** (*shapely.geometry.linestring.LineString*) – LineString 2, e.g. `ls2 = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **distance** – Minimum distance between two Shapely LineStrings

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineStrings
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring1 = LineString([(0, 0), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(0, 10), (20, 30)])
>>> linestring2.wkt
'LINESTRING (0 10, 20 30)'
```

```
>>> # Calculating distance between LineStrings
>>> distance = gg.vector.calculate_distance_linestrings(ls1=linestring1,␣
↪ls2=linestring2)
```

(continues on next page)

(continued from previous page)

```
>>> distance
7.0710678118654755
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.**calculate_midpoint_linestring**(*linestring: shapely.geometry.linestring.LineString*) → shapely.geometry.point.Point

Calculating the midpoint of a LineString with two vertices

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – LineString consisting of two vertices from which the midpoint will be extracted, e.g. `linestring = LineString([(0, 0), (20, 20)])`

Returns **point** – Shapely Point representing the midpoint of the LineString

Return type shapely.geometry.point.Point

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Calculating the midpoint of a LineString
>>> midpoint = gg.vector.calculate_midpoint_linestring(linestring=linestring)
>>> midpoint.wkt
'POINT (10 -10)'
```

See also:

calculate_midpoints_linestrings Calculating the midpoints of LineStrings

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.**calculate_midpoints_linestrings**(*linestring_gdf: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) → List[shapely.geometry.point.Point]

Calculating the midpoints of LineStrings with two vertices each

Parameters **linestring_gdf** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame containing LineStrings or list of LineStrings of which the midpoints will be calculated

Returns **midpoint_list** – List of Shapely Points representing the midpoints of the provided LineStrings

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating list of LineStrings
>>> linestring_list = [linestring, linestring]
```

```
>>> # Calculating midpoints from LineStrings
>>> midpoints = gg.vector.calculate_midpoints_linestrings(linestring_gdf=linestring_
↪list)
>>> midpoints
[<shapely.geometry.point.Point at 0x231ea982880>,
 <shapely.geometry.point.Point at 0x231ea982700>]
```

```
>>> # Inspecting the first element of the list
>>> midpoints[0].wkt
'POINT (10 -10)'
```

```
>>> # Inspecting the second element of the list
>>> midpoints[1].wkt
'POINT (10 -10)'
```

See also:

calculate_midpoint_linestring Calculating the midpoint of one LineString

gemgis.vector.**calculate_orientation_for_three_point_problem**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Calculating the orientation for a three point problem

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the three points and respective altitudes

Returns **orientation** – GeoDataFrame containing the calculated orientation value

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
>>> import geopandas as gpd
>>> points = gpd.read_file(filename='points.shp')
>>> points
    id  formation   Z   geometry
0   None    Coal    200 POINT (1842.732 602.462)
1   None    Coal    400 POINT (1696.262 1775.038)
2   None    Coal    600 POINT (104.302 1770.385)
```

```
>>> # Calculating Orientation
>>> orientation = gg.vector.calculate_orientation_for_three_point_
↪problem(gdf=points)
>>> orientation
    Z       formation   azimuth dip     polarity    X       Y       geometry
0   400.0   Coal        140.84  11.29   1           1214.43 1382.63 POINT (1214.432␣
↪1382.628)
```

gemgis.vector.**calculate_orientation_from_bent_cross_section**(*profile_linestring: shapely.geometry.linestring.LineString*, *orientation_linestring: shapely.geometry.linestring.LineString*) → list

Calculating the orientation of a LineString on a bent cross section provided as Shapely LineString

Parameters

- **profile_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map e.g. `profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])`
- **orientation_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString representing an orientation measurement on the cross section, e.g. `orientation_linestring = LineString([(2, -2), (5, -5)])`

Returns **orientation** – List containing a Shapely Point with X and Y coordinates, the Z value, dip, azimuth and polarity values

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])
>>> profile_linestring.wkt
'LINESTRING (0 0, 5 10, 20 20)'
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Calculating the orientation from a bent cross section
>>> orientations = gg.vector.calculate_orientation_from_bent_cross_section(profile_
↪linestring=profile_linestring, orientation_linestring=orientation_linestring)
>>> orientations
[<shapely.geometry.point.Point at 0x231e7f00820>, -3.5, 45.0, 26.565051177078004, 1]
```

```
>>> # Inspecting the Point object of the list
>>> orientations[0].wkt
'POINT (1.565247584249853 3.130495168499706)'
```

See also:

calculate_orientation_from_cross_section Calculating the orientation of a LineString on a cross section

calculate_orientations_from_cross_section Calculating orientations for LineStrings on a cross section

extract_orientations_from_cross_sections Calculating the orientations for LineStrings on cross sections

gemgis.vector.**calculate_orientation_from_cross_section**(*profile_linestring: shapely.geometry.linestring.LineString*, *orientation_linestring: shapely.geometry.linestring.LineString*) → list

Calculating the orientation for one LineString on one cross sections

Parameters

- **profile_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `profile_linestring = LineString([(0, 0), (20, 20)])`
- **orientation_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString representing an orientation measurement on the cross section e.g. `orientation_linestring = LineString([(2, -2), (5, -5)])`

Returns **orientation** – List containing a Shapely Point with X and Y coordinates, the Z value, dip, azimuth and polarity values

Return type list

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> profile_linestring = LineString([(0, 0), (20, 20)])
>>> profile_linestring.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Calculating orientation orientation from cross section
>>> orientations = gg.vector.calculate_orientation_from_cross_section(profile_
↪linestring=profile_linestring, orientation_linestring=orientation_linestring)
>>> orientations
[<shapely.geometry.point.Point at 0x231e79a5370>, -3.5, 45.0, 45.0, 1]
```

```
>>> # Inspecting the Point object of the list
>>> orientations[0].wkt
'POINT (2.474873734152916 2.474873734152916)'
```

See also:

calculate_orientation_from_bent_cross_section Calculating the orientation of a LineString on a bent cross section

calculate_orientations_from_cross_section Calculating orientations for LineStrings on a cross section

extract_orientations_from_cross_sections Calculating the orientations for LineStrings on cross sections

gemgis.vector.**calculate_orientations_from_cross_section**(*profile_linestring: shapely.geometry.linestring.LineString*, *orientation_linestrings: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*, *extract_coordinates: bool = True*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations from a cross sections using multiple LineStrings

Parameters

- **profile_linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])`
- **orientations_linestrings** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.linestring.LineString]]*) – GeoDataFrame or list containing multiple orientation LineStrings

- **extract_coordinates** (*bool*) – Variable to extract the X and Y coordinates from point objects. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing the Shapely Points with X, Y coordinates, the Z value, dips, azimuths and polarities

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> profile_linestring = LineString([(0, 0), (5, 10), (20, 20)])
>>> profile_linestring.wkt
'LINESTRING (0 0, 5 10, 20 20)'
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating List of LineStrings
>>> orientations_list = [orientation_linestring, orientation_linestring]
```

```
>>> # Calculating orientations from cross sections
>>> orientations = gg.vector.calculate_orientations_from_cross_section(profile_
↪linestring=profile_linestring, orientation_linestrings=orientations_list)
>>> orientations
    X       Y       Z       dip     azimuth polarity    geometry
0   1.57    3.13    -3.50   45.00   26.57   1.00        POINT (1.56525 3.13050)
1   1.57    3.13    -3.50   45.00   26.57   1.00        POINT (1.56525 3.13050)
```

See also:

calculate_orientation_from_cross_section Calculating the orientation of a LineString on a cross section

calculate_orientation_from_bent_cross_section Calculating orientations of a LineStrings on a bent cross section

extract_orientations_from_cross_sections Calculating the orientations for LineStrings on cross sections

gemgis.vector.**calculate_orientations_from_strike_lines**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations based on LineStrings representing strike lines

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing LineStrings representing strike lines

Returns **gdf_orient** – GeoDataFrame containing the location of orientation measurements and their associated orientation values

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Changed in version 1.1.7.

Fixing indexing issue.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring1 = LineString([(0, 0), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Create second LineString
>>> linestring2 = LineString([(0, 10), (20, 30)])
>>> linestring2.wkt
'LINESTRING (0 10, 20 30)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
>>> gdf['Z'] = [100,200]
>>> gdf['id'] = [1,2]
>>> gdf
    geometry                              Z   id
0       LINESTRING (0.0 0.0, 20.0 20.0)     100 1
1       LINESTRING (0.0 10.0, 20.0 30.0)    200 2
```

```
>>> # Calculating orientations strike lines
>>> orientations = gg.vector.calculate_orientations_from_strike_lines(gdf=gdf)
>>> orientations
    dip     azimuth     Z           geometry            polarity        X           ␣
↪Y
0       85.96   135.00  150.00  POINT (10.0 15.0)       1.00        10.00       15.
↪00
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between two LineStrings

gemgis.vector.**calculate_strike_direction_bent_linestring**(*linestring: shapely.geometry.linestring.LineString*) → List[float]

Calculating the strike direction of a LineString with multiple elements

Parameters **linestring** (*linestring: shapely.geometry.linestring.LineString*) – Shapely LineString containing more than two vertices, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **angles_splitted_linestrings** – List containing the strike angles of each line segment of the original

Return type List[float]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Calculating the strike angles for LineString elements
>>> angles = gg.vector.calculate_strike_direction_bent_
↪linestring(linestring=linestring)
>>> angles
[45.0, 45.0]
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_straight_linestring Calculating the strike direction of a straight LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

gemgis.vector.**calculate_strike_direction_straight_linestring**(*linestring: shapely.geometry.linestring.LineString*) → float

Function to calculate the strike direction of a straight Shapely LineString. The strike will always be calculated from start to end point

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString representing the surface trace of a straight geological profile, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **angle** – Strike angle calculated from start to end point for a straight Shapely LineString

Return type float

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 20)'
```

```
>>> # Calculating the strike angle of the LineString
>>> angle = gg.vector.calculate_strike_direction_straight_
↪linestring(linestring=linestring)
>>> angle
45.0
```

See also:

calculate_angle Calculating the angle of a LineString

calculate_strike_direction_bent_linestring Calculating the strike direction of a bent LineString

calculate_dipping_angle_linestring Calculate the dipping angle of a LineString

calculate_dipping_angles_linestrings Calculate the dipping angles of LineStrings

---

Note: The LineString must only consist of two points (start and end point)

---

gemgis.vector.**clip_by_bbox**(*gdf: geopandas.geodataframe.GeoDataFrame*, *bbox: List[Union[int, float]]*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*) → geopandas.geodataframe.GeoDataFrame

Clipping vector data contained in a GeoDataFrame to a provided bounding box/extent

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing vector data that will be clipped to a provided bounding box/extent
- **bbox** (*List[Union[float, int]]*) – Bounding box of minx, maxx, miny, maxy values to clip the GeoDataFrame, , e.g. `bbox=[0, 972, 0, 1069]`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing vector data clipped by a bounding box

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id  geometry
0       None    POINT (281.526 902.087)
1       None    POINT (925.867 618.577)
2       None    POINT (718.131 342.799)
3       None    POINT (331.011 255.684)
4       None    POINT (300.083 600.535)
```

```
>>> # Returning the length of the original gdf
>>> len(gdf)
50
```

```
>>> # Defining bounding box
>>> bbox = [0,972, 0, 1069]
```

```
>>> # Clipping data by bounding box
>>> gdf_clipped = gg.vector.clip_by_bbox(gdf=gdf, bbox=bbox)
>>> gdf_clipped
    geometry             X       Y
0       POINT (281.526 902.087) 281.53  902.09
1       POINT (925.867 618.577) 925.87  618.58
2       POINT (718.131 342.799) 718.13  342.80
3       POINT (331.011 255.684) 331.01  255.68
4       POINT (300.083 600.535) 300.08  600.54
```

```
>>> # Returning the length of the clipped gdf
>>> len(gdf_clipped)
25
```

See also:

clip_by_polygon Clipping vector data with a Shapely Polygon

gemgis.vector.**clip_by_polygon**(*gdf: geopandas.geodataframe.GeoDataFrame*, *polygon: shapely.geometry.polygon.Polygon*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*) → geopandas.geodataframe.GeoDataFrame

Clipping vector data contained in a GeoDataFrame to a provided bounding box/extent

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing vector data that will be clipped to a provided bounding box/extent
- **polygon** (*polygon: shapely.geometry.polygon*) – Shapely Polygon defining the extent of the data, e.g. `polygon = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing vector data clipped by a bounding box

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id  geometry
0       None    POINT (281.526 902.087)
1       None    POINT (925.867 618.577)
2       None    POINT (718.131 342.799)
3       None    POINT (331.011 255.684)
4       None    POINT (300.083 600.535)
```

```
>>> # Returning the length of the original gdf
>>> len(gdf)
50
```

```
>>> # Creating Shapely Polygon
>>> from shapely.geometry import Polygon
>>> polygon = Polygon([(0,0),(972, 0), (972,1069), (0, 1069)])
>>> polygon.wkt
'POLYGON ((0 0, 972 0, 972 1069, 0 1069, 0 0))'
```

```
>>> # Clipping data by the polygon
>>> gdf_clipped = gg.vector.clip_by_polygon(gdf=gdf, polygon=polygon)
>>> gdf_clipped
    geometry            X       Y
0       POINT (281.526 902.087) 281.53  902.09
1       POINT (925.867 618.577) 925.87  618.58
2       POINT (718.131 342.799) 718.13  342.80
3       POINT (331.011 255.684) 331.01  255.68
4       POINT (300.083 600.535) 300.08  600.54
```

```
>>> # Returning the length of the clipped gdf
>>> len(gdf_clipped)
25
```

See also:

clip_by_bbox Clipping vector data with a bbox

gemgis.vector.**create_bbox**(*extent: List[Union[int, float]]*) → shapely.geometry.polygon.Polygon

Creating a rectangular polygon from the provided bounding box values, with counter-clockwise order by default.

Parameters **extent** (*List[Union[int, float]]*) – List of minx, maxx, miny, maxy values, e.g. `extent=[0, 972, 0, 1069]`

Returns **bbox** – Rectangular polygon based on extent

Return type shapely.geometry.polygon.Polygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
```

```
>>> # Defining extent
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Creating bounding box
>>> bbox = gg.vector.create_bbox(extent=extent)
>>> bbox.wkt
'POLYGON ((972 0, 972 1069, 0 1069, 0 0, 972 0))'
```

gemgis.vector.**create_buffer**(*geom_object: shapely.geometry.base.BaseGeometry*, *distance: Union[float, int]*) → shapely.geometry.polygon.Polygon

Creating a buffer around a Shapely LineString or a Point

Parameters

- **geom_object** (*shapely.geometry.base.BaseGeometry*) – Shapely LineString or Point, e.g. `geom_object=Point(0, 0)`
- **distance** (*float, int*) – Distance of the buffer around the geometry object, e.g. `distance=10`

Returns **polygon** – Polygon representing the buffered area around a geometry object

Return type shapely.geometry.polygon.Polygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> point = Point(0,0)
>>> point.wkt
'POINT (0 0)'
```

```
>>> # Creating Buffer around Point
>>> point_buffered = gg.vector.create_buffer(geom_object=point, distance=10)
>>> point_buffered.wkt
'POLYGON ((100 0, 99.5184726672197 -9.801714032956051, 98.07852804032305 -19.
↪50903220161281, 95.69403357322089
-29.02846772544621, 92.38795325112869 -38.26834323650894, 88.19212643483553...))'
```

See also:

create_unified_buffer Creating a unified buffer around Shapely LineStrings or Points

gemgis.vector.**create_hexagon**(*center: shapely.geometry.point.Point*, *radius: Union[int, float]*)

Function to create one hexagon

Parameters

- **center** (*shapely.geometry.Point*) – Shapely Point representing the center of the hexagon
- **radius** (*int, float*) – Radius of the hexagon

Returns **geometry.Polygon(hex_coords)** – Shapley Polygon in the shape of a hexagon

Return type shapely.geometry.Polygon

New in version 1.0.x.

Changed in version 1.1.3: Optimized creation of hexagon

See also:

create_hexagon_grid Creating a hexagon grid

gemgis.vector.**create_hexagon_grid**(*gdf: geopandas.geodataframe.GeoDataFrame*, *radius: Union[int, float]*, *crop_gdf: bool = True*)

Function to create a grid of hexagons based on a GeoDataFrame containing Polygons and a radius provided for the single hexagons

Parameters

- **gdf** (*gpd.GeoDataFrame*) – GeoDataFrame containing the polygons for which a hexagon grid is created
- **radius** (*int, float*) – Radius of the hexagon

- **crop_gdf** (*bool*) – Boolean to define if the resulting GeoDataFrame should be cropped to the extend of the provided GeoDataFrame Options include: `True` or `False`, default set to `True`

Returns **hex_gdf** – GeoDataFrame containing the hexagon grid

Return type gpd.GeoDataFrame

New in version 1.0.x.

Changed in version 1.1.3: Optimized creation of hexagon

See also:

create_hexagon Creating one hexagon based on a given center and radius

gemgis.vector.**create_linestring_from_points**(*gdf: geopandas.geodataframe.GeoDataFrame*, *formation: str*, *altitude: Union[int, float]*) → shapely.geometry.linestring.LineString

Creating a LineString object from a GeoDataFrame containing surface points at a given altitude and for a given formation

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points of intersections between topographic contours and layer boundaries
- **formation** (*str*) – Name of the formation, e.g. `formation='Layer1'`
- **altitude** (*Union[int, float]*) – Value of the altitude of the points, e.g. `altitude=100`

Returns **linestring** – LineString containing a LineString object

Return type shapely.geometry.linestring.LineString

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating points
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> import geopandas as gpd
>>> point1 = Point(0,0)
>>> point2 = Point (10,10)
```

```
>>> # Creating GeoDataFrame from points and adding additional information
>>> gdf = gpd.GeoDataFrame(geometry=[point1, point2])
>>> gdf['formation'] = 'Layer1'
>>> gdf['Z'] = 100
>>> gdf
    geometry            formation   Z
0   POINT (0.0 0.0)     Layer1      100
1   POINT (10.0 10.0)   Layer1      100
```

```
>>> # Creating LineString from Points
>>> linestring = gg.vector.create_linestring_from_points(gdf=gdf, formation='Layer1
↪', altitude=100)
>>> linestring.wkt
'LINESTRING (0 0, 10 10)'
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_gdf Create GeoDataFrame with LineStrings from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.**create_linestring_from_xyz_points**(*points: Union[numpy.ndarray, geopandas.geodataframe.GeoDataFrame]*, *nodata: Union[int, float] = 9999.0*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*, *drop_nan: bool = True*) → shapely.geometry.linestring.LineString

Create LineString from an array or GeoDataFrame containing X, Y, and Z coordinates of points.

Parameters

- **points** (*Union[np.ndarray, gpd.geodataframe.GeoDataFrame]*) – NumPy Array or GeoDataFrame containing XYZ points.
- **nodata** (*Union[int, float]*) – Nodata value to filter out points outside a designated area, e.g. `nodata=9999.0`, default is `9999.0`.
- **xcol** (*str*) – Name of the X column in the dataset, e.g. `xcol='X'`, default is `'X'`.
- **ycol** (*str*) – Name of the Y column in the dataset, e.g. `ycol='Y'`, default is `'Y'`.
- **zcol** (*str*) – Name of the Z column in the dataset, e.g. `zcol='Z'`, default is `'Z'`.
- **drop_nan** (*bool*) – Boolean argument to drop points that contain a `nan` value as Z value. Options include `True` and `False`, default is `True`.

Returns **line** – LineString Z constructed from provided point values

Return type shapely.geometry.linestring.LineString

New in version 1.0.x.

Changed in version 1.1: Adding argument *drop_nan* and code to drop coordinates that contain `nan` values as Z coordinates.

Example

```
>>> # Loading Libraries and creating points
>>> import gemgis as gg
>>> import numpy as np
>>> points = np.array([[3.23, 5.69, 2.03],[3.24, 5.68, 2.02],[3.25, 5.67, 1.97],[3.
↪26, 5.66, 1.95]])
```

```
>>> # Creating LineStrings from points
>>> linestring = gg.vector.create_linestring_from_xyz_points(points=points)
>>> linestring.wkt
'LINESTRING Z (3.23 5.69 2.03, 3.24 5.68 2.02, 3.25 5.67 1.97, 3.26 5.66 1.95)'
```

gemgis.vector.**create_linestring_gdf**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Creating LineStrings from Points

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points of intersections between topographic contours and layer boundaries

Returns **gdf_linestring** – GeoDataFrame containing LineStrings

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Points
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> import geopandas as gpd
>>> point1 = Point(0,0)
>>> point2 = Point (10,10)
```

```
>>> # Creating GeoDataFrame from points and adding additional information
>>> gdf = gpd.GeoDataFrame(geometry=[point1, point2])
>>> gdf['formation'] = 'Layer1'
>>> gdf['Z'] = 100
>>> gdf['id'] = 1
>>> gdf
    geometry            formation   Z   id
0   POINT (0.0 0.0)     Layer1      100 1
1   POINT (10.0 10.0)   Layer1      100 1
```

```
>>> # Creating LineString GeoDataFrame
>>> linestring_gdf = gg.vector.create_linestring_gdf(gdf=gdf)
>>> linestring_gdf
    index formation     Z       id      geometry
0   0     Layer1        100     1       LINESTRING (0.00000 0.00000, 10.00000 10.
↪00000)
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

extract_orientations_from_map Extracting orientations from a map

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.**create_linestrings_from_contours**(*contours: pyvista.core.pointset.PolyData*, *return_gdf: bool = True*, *crs: Union[str, pyproj.crs.crs.CRS] = None*) → Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame]

Creating LineStrings from PyVista Contour Lines and save them as list or GeoDataFrame

Parameters

- **contours** (*pv.core.pointset.PolyData*) – PyVista PolyData dataset containing contour lines extracted from a mesh
- **return_gdf** (*bool*) – Variable to create GeoDataFrame of the created list of Shapely Objects. Options include: `True` or `False`, default set to `True`
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`

Returns **linestrings** – List of LineStrings or GeoDataFrame containing the contours that were converted

Return type Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> contours = pv.read('file.vtk')
>>> contours
Header
PolyData    Information
N Cells     36337
N Points    36178
X Bounds    3.233e+07, 3.250e+07
Y Bounds    5.704e+06, 5.798e+06
Z Bounds    -2.400e+03, 3.500e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -2.400e+03  3.500e+02
```

```
>>> # Extracting LineStrings from contours
>>> gdf = gg.vector.create_linestrings_from_contours(contours=contours)
>>> gdf
```

(continues on next page)

(continued from previous page)

```
    geometry                                                Z
0   LINESTRING Z (32409587.930 5780538.824 -2350.0...   -2350.00
1   LINESTRING Z (32407304.336 5777048.086 -2050.0...   -2050.00
2   LINESTRING Z (32408748.977 5778005.047 -2200.0...   -2200.00
3   LINESTRING Z (32403693.547 5786613.994 -2400.0...   -2400.00
4   LINESTRING Z (32404738.664 5782672.480 -2350.0...   -2350.00
```

gemgis.vector.**create_linestrings_from_xyz_points**(*gdf: geopandas.geodataframe.GeoDataFrame*, *groupby: str*, *nodata: Union[int, float] = 9999.0*, *xcol: str = 'X'*, *ycol: str = 'Y'*, *zcol: str = 'Z'*, *dem: Union[numpy.ndarray, rasterio.io.DatasetReader] = None*, *extent: List[Union[int, float]] = None*, *return_gdf: bool = True*, *drop_nan: bool = True*) → Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame]

Creating LineStrings from a GeoDataFrame containing X, Y, and Z coordinates of vertices of multiple LineStrings

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing extracted X, Y, and Z coordinates of LineStrings
- **groupby** (*str*) – Name of a unique identifier the LineStrings can be separated from each other, e.g. `groupby='Object_ID'`
- **nodata** (*Union[int, float]*) – Nodata value to filter out points outside a designated area, e.g. `nodata=9999.0`, default is `9999.0`
- **xcol** (*str*) – Name of the X column in the dataset, e.g. `xcol='X'`, default is `'X'`
- **ycol** (*str*) – Name of the Y column in the dataset, e.g. `ycol='Y'`, default is `'Y'`
- **zcol** (*str*) – Name of the Z column in the dataset, e.g. `zcol='Z'`, default is `'Z'`
- **dem** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy ndarray or rasterio object containing the height values, default value is `None` in case geometries contain Z values
- **extent** (*List[Union[float, int]]*) – Values for minx, maxx, miny and maxy values to define the boundaries of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **return_gdf** (*bool*) – Variable to either return the data as GeoDataFrame or as list of LineStrings. Options include: `True` or `False`, default set to `True`
- **drop_nan** (*bool*) – Boolean argument to drop points that contain a `nan` value as Z value. Options include `True` and `False`, default is `True`

Returns **linestrings** – List of LineStrings or GeoDataFrame containing the LineStrings with Z component

Return type Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Changed in version 1.1: Removed manual dropping of additional columns. Now automatically drops unnecessary coloumns. Adding argument *drop_nan* and code to drop coordinates that contain `nan` values as Z coordinates.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
```

```
>>> # Creating LineStrings with Z component from gdf
>>> gdf_linestring = gg.vector.create_linestrings_from_xyz_points(gdf=gdf, groupby=
↪'ABS')
>>> gdf_linestring
```

gemgis.vector.**create_polygons_from_faces**(*mesh: pyvista.core.pointset.PolyData*, *crs: Union[str, pyproj.crs.crs.CRS]*, *return_gdf: bool = True*) → Union[List[shapely.geometry.polygon.Polygon], geopandas.geodataframe.GeoDataFrame]

Extracting faces from PyVista PolyData as Shapely Polygons

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista PolyData dataset
- **crs** – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`

Returns **polygons** – Triangular Shapely Polygons representing the faces of the mesh

Return type Union[List[shapely.geometry.polygon.Polygon], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Importing Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> mesh = pv.read(filename='mesh.vtk')
>>> mesh
Header
PolyData    Information
N Cells     29273
N Points    40343
X Bounds    2.804e+05, 5.161e+05
Y Bounds    5.640e+06, 5.833e+06
Z Bounds    -8.067e+03, 1.457e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -8.067e+03  1.457e+02
```

```
>>> # Create polygons from mesh faces
>>> polygons = gg.vector.create_polygons_from_faces(mesh=mesh)
>>> polygons
```

(continues on next page)

(continued from previous page)

```
    geometry
0   POLYGON Z ((297077.414 5677487.262 -838.496, 2...
1   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
2   POLYGON Z ((297437.539 5676992.094 -816.608, 2...
3   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
4   POLYGON Z ((295827.680 5680951.574 -825.328, 2...
```

gemgis.vector.**create_unified_buffer**(*geom_object: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]], distance: Union[numpy.ndarray, List[Union[int, float]], float, int]*) → shapely.geometry.multipolygon.MultiPolygon

Creating a unified buffer around Shapely LineStrings or Points

Parameters

- **geom_object** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]]*) – GeoDataFrame or List of Shapely objects
- **distance** (*Union[np.ndarray, List[Union[float, int]], Union[float, int]]*) – Distance of the buffer around the geometry object, e.g. `distance=10`

Returns **polygon** – Polygon representing the buffered area around a geometry object

Return type shapely.geometry.multipolygon.MultiPolygon

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> point1 = Point(0,0)
>>> point1.wkt
'POINT (0 0)'
```

```
>>> # Creating Point
>>> point2 = Point(20,20)
>>> point2.wkt
'POINT (20 20)'
```

```
>>> # Creating list of points
>>> point_list = [point1, point2]
```

```
>>> # Creating unified buffer
>>> unified_buffer = gg.vector.create_unified_buffer(geom_object=point_list,␣
↪distance=10)
>>> unified_buffer
'MULTIPOLYGON (((10 0, 9.95184726672197 -0.980171403295605, 9.807852804032306 -1.
↪950903220161281, 9.56940335732209
-2.902846772544621, 9.23879532511287 -3.826834323650894,...)))'
```

See also:

create_buffer Creating a buffer around a Shapely LineString or Point

gemgis.vector.**create_voronoi_polygons**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Function to create Voronoi Polygons from Point GeoDataFrame using the SciPy Spatial Voronoi class (https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.Voronoi.html#scipy.spatial.Voronoi)

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the Shapely Points

Returns **gdf_polygons** – GeoDataFrame containing the valid Voronoi Polygons

Return type gpd.geodataframe.GeoDataFrame

New in version 1.1.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file('file.shp')
>>> gdf_polygons = gg.vector.create_voronoi_polygons(gdf=gdf)
```

gemgis.vector.**explode_geometry_collection**(*collection: shapely.geometry.collection.GeometryCollection*) → List[shapely.geometry.base.BaseGeometry]

Exploding a Shapely Geometry Collection to a List of Base Geometries

Parameters **collection** (*shapely.geometry.collection.GeometryCollection*) – Shapely Geometry Collection consisting of different Base Geometries

Returns **collection_exploded** – List of Base Geometries from the original Geometry Collection

Return type List[shapely.geometry.base.BaseGeometry]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Geometry Collection
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> a = LineString([(0, 0), (1, 1), (1,2), (2,2)])
>>> b = LineString([(0, 0), (1, 1), (2,1), (2,2)])
>>> collection = a.intersection(b)
>>> collection.wkt
'GEOMETRYCOLLECTION (POINT (2 2), LINESTRING (0 0, 1 1))'
```

```
>>> # Exploding Geometry collection into single Base Geometries
>>> collection_exploded = gg.vector.explode_geometry_
↪collection(collection=collection)
>>> collection_exploded
[<shapely.geometry.point.Point at 0x1faf63ccac0>,
<shapely.geometry.linestring.LineString at 0x1faf63ccb80>]
```

```
>>> # Inspecting the first element of the list
>>> collection_exploded[0].wkt
'POINT (2 2)'
```

```
>>> # Inspecting the second element of the list
>>> collection_exploded[1].wkt
'LINESTRING (0 0, 1 1)'
```

See also:

explode_geometry_collections Exploding a GeoDataFrame containing different Base Geometries

gemgis.vector.**explode_geometry_collections**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *remove_points: bool = True*) → geopandas.geodataframe.GeoDataFrame

Exploding Shapely Geometry Collections stored in GeoDataFrames to different Shapely Base Geometries

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type GeometryCollection
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **remove_points** (*bool*) – Variable to remove points from exploded GeoDataFrame. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing different geometry types

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Geometries
>>> import gemgis as gg
>>> from shapely.geometry import LineString, Polygon
>>> import geopandas as gpd
>>> a = LineString([(0, 0), (1, 1), (1,2), (2,2)])
>>> b = LineString([(0, 0), (1, 1), (2,1), (2,2)])
>>> collection = a.intersection(b)
>>> polygon = Polygon([(0, 0), (10, 0), (10, 10), (0, 10)])
```

```
>>> # Creating GeoDataFrame from Base Geometries
>>> gdf = gpd.GeoDataFrame(geometry=[a, b, collection, polygon])
>>> gdf
```

(continues on next page)

(continued from previous page)

```
    geometry
0       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
1       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
2       GEOMETRYCOLLECTION (POINT (2.00000 2.00000), L...
3       POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1..
```

```
>>> # Explode Geometry Collection into single Base Geometries
>>> gdf_exploded = gg.vector.explode_geometry_collections(gdf=gdf)
>>> gdf_exploded
    geometry
0       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
1       LINESTRING (0.00000 0.00000, 1.00000 1.00000, ...
2       LINESTRING (0.00000 0.00000, 1.00000 1.00000)
3       POLYGON ((0.00000 0.00000, 10.00000 0.00000, 1...
```

See also:

explode_geometry_collection Exploding a Shapely Geometry Collection Object into a list of Base Geometries

gemgis.vector.**explode_linestring**(*linestring: shapely.geometry.linestring.LineString*) → List[shapely.geometry.point.Point]

Exploding a LineString to its vertices, also works for LineStrings with Z components

Parameters **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString from which vertices are extracted, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **points_list** – List of extracted Shapely Points

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Exploding LineString into single points
>>> linestring_exploded = gg.vector.explode_linestring(linestring=linestring)
>>> linestring_exploded
[<shapely.geometry.point.Point at 0x20118cb27f0>,
<shapely.geometry.point.Point at 0x20118cb28b0>,
<shapely.geometry.point.Point at 0x20118cb26d0>]
```

```
>>> # Inspecting the first element of the list
>>> linestring_exploded[0].wkt
'POINT (0 0)'
```

```
>>> # Inspecting the second element of the list
>>> linestring_exploded[1].wkt
'POINT (10 10)'
```

```
>>> # Inspecting the third element of the list
>>> linestring_exploded[2].wkt
'POINT (20 20)'
```

See also:

explode_linestring_to_elements Exploding a LineString with more than two vertices into single LineStrings

gemgis.vector.**explode_linestring_to_elements**(*linestring: shapely.geometry.linestring.LineString*) → List[shapely.geometry.linestring.LineString]

Separating a LineString into its single elements and returning a list of LineStrings representing these elements, also works for LineStrings with Z components

Parameters **linestring** (*linestring: shapely.geometry.linestring.LineString*) – Shapely LineString containing more than two vertices, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`

Returns **splitted_linestrings** – List containing the separate elements of the original LineString stored as LineStrings

Return type List[shapely.geometry.linestring.LineString]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Exploding LineString into single elements
>>> linestring_exploded = gg.vector.explode_linestring_to_
↪elements(linestring=linestring)
>>> linestring_exploded
[<shapely.geometry.linestring.LineString at 0x201448a2100>,
<shapely.geometry.linestring.LineString at 0x20144b5e610>]
```

```
>>> # Inspecting the first element of the list
>>> linestring_exploded[0].wkt
'LINESTRING (0 0, 10 10)'
```

```
>>> # Inspecting the second element of the list
>>> linestring_exploded[1].wkt
'LINESTRING (10 10, 20 20)'
```

See also:

explode_linestring Exploding a LineString into its single vertices

gemgis.vector.**explode_multilinestring**(*multilinestring: shapely.geometry.multilinestring.MultiLineString*) → List[shapely.geometry.linestring.LineString]

Exploding a MultiLineString into a list of LineStrings

Parameters **multilinestring** (*shapely.geometry.multilinestring.MultiLineString*) – Shapely MultiLineString consisting of multiple LineStrings, e.g. `multilinestring = MultiLineString([((0, 0), (1, 1)), ((-1, 0), (1, 0))])`

Returns **splitted_multilinestring** – List of Shapely LineStrings

Return type List[shapely.geometry.linestring.LineString]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating MultiLineString
>>> import gemgis as gg
>>> from shapely.geometry import MultiLineString
>>> coords = [((0, 0), (1, 1)), ((-1, 0), (1, 0))]
>>> lines = MultiLineString(coords)
>>> lines.wkt
'MULTILINESTRING ((0 0, 1 1), (-1 0, 1 0))'
```

```
>>> lines_splitted = gg.vector.explode_multilinestrings(multilinestring=lines)
>>> lines_splitted
[<shapely.geometry.linestring.LineString at 0x2014a5f0ee0>,
<shapely.geometry.linestring.LineString at 0x20149dda430>]
```

```
>>> # Inspecting the first element of the list
>>> lines_splitted[0].wkt
'LINESTRING (0 0, 1 1)'
```

```
>>> # Inspecting the second element of the list
>>> lines_splitted[1].wkt
'LINESTRING (-1 0, 1 0)'
```

See also:

explode_multilinestrings Exploding a GeoDataFrame containing MultiLineStrings into a GeoDataFrame containing LineStrings only

gemgis.vector.**explode_multilinestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*) → geopandas.geodataframe.GeoDataFrame

Exploding Shapely MultiLineStrings stored in a GeoDataFrame to Shapely LineStrings

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type MultiLineString
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing LineStrings

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    geometry
0   MULTILINESTRING ((0.0 0.0, 1.0 1.0))
1   MULTILINESTRING ((0.0 0.0, 1.0 1.0))
```

```
>>> # Exploding MultiLineStrings into single LineStrings
>>> gdf_linestrings = gg.vector.explode_multilinestrings(gdf=gdf, reset_index=True)
>>> gdf_linestrings
    geometry
0       LINESTRING (0.0 0.0, 1.0 1.0)
1       LINESTRING (-1.0 0.0, 1.0 0.0)
2       LINESTRING (0.0 0.0, 1.0 1.0)
3       LINESTRING (-1.0 0.0, 1.0 0.0)
```

See also:

explode_multilinestring Exploding a MultiLineString into a list of single LineStrings

gemgis.vector.**explode_polygon**(*polygon: shapely.geometry.polygon.Polygon*) → List[shapely.geometry.point.Point]

Exploding Shapely Polygon to list of Points

Parameters **polygon** (*shapely.geometry.polygon.Polygon*) – Shapely Polygon from which vertices are extracted, e.g. `polygon = Polygon([(0, 0), (1, 1), (1, 0)])`

Returns **point_list** – List containing the vertices of a polygon as Shapely Points

Return type List[shapely.geometry.point.Point]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon = Polygon([(0, 0), (1, 1), (1, 0)])
>>> polygon.wkt
'POLYGON ((0 0, 1 1, 1 0, 0 0))'
```

```
>>> # Exploding Polygon into single Points
>>> polygon_exploded = gg.vector.explode_polygon(polygon=polygon)
>>> polygon_exploded
[<shapely.geometry.point.Point at 0x201459734f0>,
<shapely.geometry.point.Point at 0x20145973670>,
<shapely.geometry.point.Point at 0x20145973640>,
<shapely.geometry.point.Point at 0x201459732e0>]
```

```
>>> # Inspecting the first element of the list
>>> polygon_exploded[0].wkt
'POINT (0 0)'
```

```
>>> # Inspecting the second element of the list
>>> polygon_exploded[1].wkt
'POINT (1 1)'
```

See also:

explode_polygons Exploding a GeoDataFrame containing Polygons into a GeoDataFrame containing LineStrings

gemgis.vector.**explode_polygons**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Converting a GeoDataFrame containing elements of geom_type Polygons to a GeoDataFrame with LineStrings

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type Polygon

Returns **gdf_linestrings** – GeoDataFrame containing elements of type MultiLineString and LineString

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    geometry
```

(continues on next page)

(continued from previous page)

```
0       POLYGON ((0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0))
1       POLYGON ((0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0))
```

```
>>> # Exploding Polygons into LineStrings
>>> gdf_exploded = gg.vector.explode_polygons(gdf=gdf)
>>> gdf_exploded
    geometry
0       LINESTRING (0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0)
1       LINESTRING (0.0 0.0, 1.0 1.0, 1.0 0.0, 0.0 0.0)
```

See also:

explode_polygon Exploding a Polygon into single Points

gemgis.vector.**extract_interfaces_coordinates_from_cross_section**(*linestring: shapely.geometry.linestring.LineString*, *interfaces_gdf: geopandas.geodataframe.GeoDataFrame*, *extract_coordinates: bool = True*) → geopandas.geodataframe.GeoDataFrame

Extracting coordinates of interfaces digitized on a cross section

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – Shapely LineString containing the trace of a cross section on a map, e.g. `linestring = LineString([(0, 0), (20, 20)])`
- **interfaces_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the LineStrings of interfaces digitized on a cross section

Returns **gdf** – GeoDataFrame containing the extracted coordinates, depth/elevation data and additional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> import geopandas as gpd
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating GeoDataFrame from LineString
>>> gdf = gpd.GeoDataFrame(geometry=[interfaces, interfaces])
>>> gdf
    geometry
0   LINESTRING (2.0 -2.0, 5.0 -5.0)
1   LINESTRING (2.0 -2.0, 5.0 -5.0)
```

```
>>> # Extracting interfaces coordinates from cross sections
>>> gdf_points = gg.vector.extract_interfaces_coordinates_from_cross_
↪section(linestring=linestring, interfaces_gdf=gdf)
>>> gdf_points
    geometry                    X       Y       Z
0   POINT (1.41421 -1.41421)    1.41    -1.41   -2.00
1   POINT (3.53553 -3.53553)    3.54    -3.54   -5.00
2   POINT (1.41421 -1.41421)    1.41    -1.41   -2.00
3   POINT (3.53553 -3.53553)    3.54    -3.54   -5.00
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for one LineString on cross sections

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_xyz_from_cross_sections Extracting the X, Y, and Z coordinates of interfaces from cross sections

gemgis.vector.**extract_orientations_from_cross_sections**(*profile_gdf: geopandas.geodataframe.GeoDataFrame*, *orientations_gdf: geopandas.geodataframe.GeoDataFrame*, *profile_name_column: str = 'name'*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations digitized from cross sections

Parameters

- **profile_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the different profile traces as LineStrings
- **orientations_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the orientation LineStrings for different profiles and formations
- **profile_name_column** (*str*) – Name of the profile column, e.g. `profile_name_column='name'`, default is `'name'`

Returns **gdf** – GeoDataFrame containing the orientation and location data for orientations digitized on cross sections

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> import geopandas as gpd
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating GeoDataFrame from LineString and adding profile names
>>> profile_gdf = gpd.GeoDataFrame(geometry=[linestring, linestring])
>>> profile_gdf['name'] = ['Profile2', 'Profile1']
>>> profile_gdf
    geometry                              name
0   LINESTRING (0.0 0.0, 20.0 -20.0)    Profile2
1   LINESTRING (0.0 0.0, 20.0 -20.0)    Profile1
```

```
>>> # Creating second LineString
>>> orientation_linestring = LineString([(2, -2), (5, -5)])
>>> orientation_linestring.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating GeoDataFrame from LineString and adding profile names
>>> orientations_gdf = gpd.GeoDataFrame(geometry=[orientation_linestring,␣
↪orientation_linestring])
>>> orientations_gdf
    geometry                          name
0   LINESTRING (2.0 -2.0, 5.0 -5.0) Profile2
1   LINESTRING (2.0 -2.0, 5.0 -5.0) Profile1
```

```
>>> # Extract orientations from cross sections
>>> orientations = gg.vector.extract_orientations_from_cross_sections(profile_
↪gdf=profile_gdf, orientations_gdf=orientations_gdf)
>>> orientations
    X       Y       Z       dip     azimuth     polarity    geometry              ␣
↪    name
0   2.47    -2.47   -3.50   45.00   135.00      1.00        POINT (2.47487 -2.
↪47487)    Profile2
1   2.47    -2.47   -3.50   45.00   135.00      1.00        POINT (2.47487 -2.
↪47487)    Profile1
```

gemgis.vector.**extract_orientations_from_map**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dz: str = 'dZ'*) → geopandas.geodataframe.GeoDataFrame

Calculating orientations from LineStrings

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the orientation LineStrings
- **dz** (*str*) – Name of the height difference column, e.g. `dz='dZ'`

Returns **gdf** – GeoDataFrame containing the orientation values

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring1 = LineString([(0, 0), (20, -20)])
>>> linestring1.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(0, 0), (20, -10)])
>>> linestring2.wkt
'LINESTRING (0 0, 20 -10)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
>>> gdf['dZ'] = [100, 200]
>>> gdf
    geometry                            dz
0   LINESTRING (0.0 0.0, 20.0 -20.0)    100
1   LINESTRING (0.0 0.0, 20.0 -10.0)    200
```

```
>>> # Extracting orientations from map
>>> orientations = gg.vector.extract_orientations_from_map(gdf=gdf)
>>> orientations
    geometry            azimuth dip     X       Y       polarity
0   POINT (10.0 -10.0)  135.00  74.21   10.00   -10.00  1
1   POINT (10.0 -5.0)   116.57  83.62   10.00   -5.00   1
```

See also:

calculate_azimuth Calculating the azimuth for orientations on a map

create_linestring_from_points Create LineString from points

create_linestring_gdf Create GeoDataFrame with LineStrings from points

calculate_distance_linestrings Calculating the distance between LineStrings

calculate_orientations_from_strike_lines Calculating the orientations from strike lines

gemgis.vector.**extract_xy**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *overwrite_xy: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings, Polygons, Geometry Collections) and returning a GeoDataFrame with X and Y coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data such as Shapely Points, LineStrings, MultiLineStrings or Polygons or data loaded from disc with GeoPandas (i.e. Shape File)
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **overwrite_xy** (*bool*) – Variable to overwrite existing X and Y values. Options include: `True` or `False`, default set to `False`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to re-project coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame with appended x, y columns and Point geometry features

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Changed in version 1.1: If a GeoDataFrame contains LineStrings and MultiLineStrings, the index of the exploded GeoDataFrame will now be reset. Not resetting the index will cause index errors later on.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
```

(continues on next page)

(continued from previous page)

```
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Extracting X and Y Coordinates from Shapely Base Geometries
>>> gdf_xy = gg.vector.extract_xy(gdf=gdf, reset_index=False)
>>> gdf_xy
    formation   geometry                X       Y
0       Ton             POINT (19.150 293.313)  19.15   293.31
1       Ton             POINT (61.934 381.459)  61.93   381.46
2       Ton             POINT (109.358 480.946) 109.36  480.95
3       Ton             POINT (157.812 615.999) 157.81  616.00
4       Ton             POINT (191.318 719.094) 191.32  719.09
```

See also:

extract_xy_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points

extract_xy_linestring Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings and saving the X and Y coordinates as lists for each LineString

extract_xy_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings

---

Note: GeoDataFrames that contain multiple types of geometries are currently not supported. Please use `gdf = gdf.explode().reset_index(drop=True)` to create a GeoDataFrame with only one type of geometries

---

gemgis.vector.**extract_xy_from_polygon_intersections**(*gdf: geopandas.geodataframe.GeoDataFrame*, *extract_coordinates: bool = False*, *drop_index: bool = True*) → geopandas.geodataframe.GeoDataFrame

Calculating the intersections between Polygons; the table must be sorted by stratigraphic age

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Polygons of a geological map ordered by their stratigraphic age
- **extract_coordinates** (*bool*) – Variable to extract X and Y coordinates from resulting Shapely Objects. Options include: `True` or `False`, default set to `False`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`

Returns **intersections** – GeoDataFrame containing the intersections of the polygons of a geological map

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> import geopandas as gpd
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Creating GeoDataFrame from polygons and adding formation names
>>> gdf = gpd.GeoDataFrame(geometry=[polygon1, polygon2])
>>> gdf['formation'] = ['Formation1', 'Formation2']
>>> gdf
    geometry                                        formation
0   POLYGON (((0 0, 10 0, 10 10, 0 10, 0 0))    Formation1
1   POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))  Formation2
```

```
>>> # Extracting X an Y coordinates from polygon intersections
>>> intersection = gg.vector.extract_xy_from_polygon_intersections(gdf=gdf)
>>> intersection
    formation   geometry
0   Formation1  LINESTRING (10.0 0.0, 10.0 10.0)
```

See also:

intersection_polygon_polygon Intersecting a polygon with a polygon

intersections_polygon_polygons Intersecting a polygons with multiple polygons

intersections_polygons_polygons Intersecting multiple polygons with multiple polygons

gemgis.vector.**extract_xy_linestring**(*gdf: geopandas.geodataframe.GeoDataFrame*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting the coordinates of Shapely LineStrings within a GeoDataFrame and storing the X and Y coordinates in lists per LineString

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type LineString
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*Optional[Sequence[float]]*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`

Returns **gdf** – GeoDataFrame containing the additional X and Y columns with lists of X and Y coordinates

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Sand1       LINESTRING (0.256 264.862, 10.593 276.734, 17....
1       None    Ton         LINESTRING (0.188 495.787, 8.841 504.142, 41.0...
2       None    Ton         LINESTRING (970.677 833.053, 959.372 800.023, ...
```

```
>>> # Extracting X and Y Coordinates from LineString Objects
>>> gdf_xy = gg.vector.extract_xy_linestring(gdf=gdf)
>>> gdf_xy
    id      formation   geometry                                                X␣
↪                                                   Y
0       None    Sand1       LINESTRING (0.256 264.862, 10.593 276.734, 17....   [0.
↪256327195431048, 10.59346813871597, 17.1349...       [264.86214748436396, 276.
↪73370778641777, 289.0...
1       None    Ton         LINESTRING (0.188 495.787, 8.841 504.142, 41.0...   [0.
↪1881868620686138, 8.840672956663411, 41.092...       [495.787213546976, 504.
↪1418419288791, 546.4230...
2       None    Ton         LINESTRING (970.677 833.053, 959.372 800.023, ...  ␣
↪[970.6766251230017, 959.3724321757514, 941.291...       [833.052616499831, 800.
↪0232029873156, 754.8012...
```

See also:

extract_xy_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings

extract_xy_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points

extract_xy Extracting X and Y coordinates from Vector Data

gemgis.vector.**extract_xy_linestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_id: bool = True*, *drop_index: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *overwrite_xy: bool = False*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (LineStrings) and returning a GeoDataFrame with X and Y coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type LineString

- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **overwrite_xy** (*bool*) – Variable to overwrite existing X and Y values. Options include: `True` or `False`, default set to `False`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to re-project coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*Optional[Sequence[float]]*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`

Returns **gdf** – GeoDataFrame with appended X and Y coordinates as additional columns and optional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Sand1       LINESTRING (0.256 264.862, 10.593 276.734, 17....
1       None    Ton         LINESTRING (0.188 495.787, 8.841 504.142, 41.0...
2       None    Ton         LINESTRING (970.677 833.053, 959.372 800.023, ...
```

```
>>> # Extracting X and Y Coordinates from LineString Objects
>>> gdf_xy = gg.vector.extract_xy_linestrings(gdf=gdf, reset_index=False)
>>> gdf_xy
    formation   geometry                X       Y
0       Sand1           POINT (0.256 264.862)   0.26    264.86
1       Sand1           POINT (10.593 276.734)  10.59   276.73
2       Sand1           POINT (17.135 289.090)  17.13   289.09
3       Sand1           POINT (19.150 293.313)  19.15   293.31
4       Sand1           POINT (27.795 310.572)  27.80   310.57
```

See also:

extract_xy_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points

extract_xy_linestring Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings and saving the X and Y coordinates as lists for each LineString

extract_xy Extracting X and Y coordinates from Vector Data

Note: The function was adapted to also extract Z coordinates from LineStrings

---

gemgis.vector.**extract_xy_points**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_id: bool = True*, *drop_index: bool = True*, *overwrite_xy: bool = False*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points) and returning a GeoDataFrame with X and Y coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing elements of geom_type Point
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **overwrite_xy** (*bool*) – Variable to overwrite existing X and Y values. Options include: `True` or `False`, default set to `False`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`

Returns **gdf** – GeoDataFrame with appended X and Y coordinates as new columns and optional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation       geometry
0   None    Ton             POINT (19.150 293.313)
1   None    Ton             POINT (61.934 381.459)
```

(continues on next page)

(continued from previous page)

```
2   None    Ton             POINT (109.358 480.946)
3   None    Ton             POINT (157.812 615.999)
4   None    Ton             POINT (191.318 719.094)
```

```
>>> # Extracting X and Y Coordinates from Point Objects
>>> gdf_xy = gg.vector.extract_xy_points(gdf=gdf, reset_index=False)
>>> gdf_xy
    formation       geometry                X       Y
0   Ton             POINT (19.150 293.313)  19.15   293.31
1   Ton             POINT (61.934 381.459)  61.93   381.46
2   Ton             POINT (109.358 480.946) 109.36  480.95
3   Ton             POINT (157.812 615.999) 157.81  616.00
4   Ton             POINT (191.318 719.094) 191.32  719.09
```

See also:

extract_xy_linestring Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings and saving the X and Y coordinates as lists for each LineString

extract_xy_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings

extract_xy Extracting X and Y coordinates from Vector Data

gemgis.vector.**extract_xyz**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: Union[numpy.ndarray, rasterio.io.DatasetReader] = None*, *minz: float = None*, *maxz: float = None*, *extent: List[Union[int, float]] = None*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and Z values from a NumPy nd.array or a Rasterio object and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing Shapely Points, LineStrings, MultiLineStrings or Polygons
- **dem** (*Union[np.ndarray, rasterio.io.DatasetReader]*) – NumPy ndarray or Rasterio object containing the height values, default value is None in case geometries contain Z values
- **minz** (*float*) – Value defining the minimum elevation of the data that needs to be returned, e.g. `minz=50`, default `None`
- **maxz** (*float*) – Value defining the maximum elevation of the data that needs to be returned, e.g. `maxz=500`, default `None`
- **extent** (*List[Union[float,int]]*) – List containing the extent of the np.ndarray, must be provided in the same CRS as the gdf, e.g. `extent=[0, 972, 0, 1069]`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`

- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z coordinates as additional columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Loading raster file
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Extracting X, Y, and Z Coordinates from Shapely Base Geometries and DEM
>>> gdf_xyz = gg.vector.extract_xyz(gdf=gdf, dem=dem, reset_index=reset_index)
>>> gdf_xyz
    formation   geometry                X       Y       Z
0   Ton         POINT (19.150 293.313)  19.15   293.31  364.99
1       Ton             POINT (61.934 381.459)  61.93   381.46  400.34
2       Ton             POINT (109.358 480.946) 109.36  480.95  459.55
3       Ton             POINT (157.812 615.999) 157.81  616.00  525.69
4       Ton             POINT (191.318 719.094) 191.32  719.09  597.63
```

See also:

extract_xyz_array Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model as array

extract_xyz_rasterio Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation as rasterio object

gemgis.vector.**extract_xyz_array**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: numpy.ndarray*, *extent: List[float]*, *minz: float = None*, *maxz: float = None*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and Z values from a NumPy nd.array and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing Shapely Points, LineStrings, MultiLineStrings or Polygons
- **dem** (*np.ndarray*) – NumPy ndarray containing the height values
- **extent** (*list*) – List containing the extent of the np.ndarray, must be provided in the same CRS as the gdf, e.g. `extent=[0, 972, 0, 1069]`
- **minz** (*float*) – Value defining the minimum elevation the data needs to be returned, e.g. `minz=50`, default `None`
- **maxz** (*float*) – Value defining the maximum elevation the data needs to be returned, e.g. `maxz=500`, default `None`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`

- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to re-project coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z coordinates

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Loading raster file
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Defining the extent of the array
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Extracting X, Y, and Z Coordinates from Shapely Base Geometries and array
>>> gdf_xyz = gg.vector.extract_xyz_array(gdf=gdf, dem=dem.read(1), extent=extent,␣
↪reset_index=reset_index)
>>> gdf_xyz
    formation   geometry                X       Y       Z
0   Ton         POINT (19.150 293.313)  19.15   293.31  364.99
```

(continues on next page)

(continued from previous page)

```
1       Ton             POINT (61.934 381.459)  61.93   381.46  400.34
2       Ton             POINT (109.358 480.946) 109.36  480.95  459.55
3       Ton             POINT (157.812 615.999) 157.81  616.00  525.69
4       Ton             POINT (191.318 719.094) 191.32  719.09  597.63
```

See also:

extract_xyz_rasterio Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model as rasterio object

extract_xyz Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model

gemgis.vector.**extract_xyz_from_cross_sections**(*profile_gdf: geopandas.geodataframe.GeoDataFrame*, *interfaces_gdf: geopandas.geodataframe.GeoDataFrame*, *profile_name_column: str = 'name'*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from cross sections and digitized interfaces

Parameters

- **profile_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the traces (LineStrings) of cross sections on a map and a profile name
- **interfaces_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the LineStrings of digitized interfaces, associated formation and the profile name
- **profile_name_column** (*str*) – Name of the profile column, default is `profile_name_column='name'`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z information of all extracted digitized interfaces on cross sections

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> import geopandas as gpd
>>> linestring = LineString([(0, 0), (20, -20)])
>>> linestring.wkt
'LINESTRING (0 0, 20 -20)'
```

```
>>> # Creating GeoDataFrame from LineString and ad Profile names
>>> profile_gdf = gpd.GeoDataFrame(geometry=[linestring, linestring])
>>> profile_gdf['name'] = ['Profile1', 'Profile2']
>>> profile_gdf
    geometry    name
0       LINESTRING (0.0 0.0, 20.0 -20.0)        Profile1
1       LINESTRING (0.0 0.0, 20.0 -20.0)        Profile2
```

```
>>> # Creating second LineString
>>> interfaces = LineString([(2, -2), (5, -5)])
>>> interfaces.wkt
'LINESTRING (2 -2, 5 -5)'
```

```
>>> # Creating GeoDataFrame from LineString and ad Profile names
>>> gdf = gpd.GeoDataFrame(geometry=[interfaces, interfaces])
>>> gdf['name'] = ['Profile1', 'Profile2']
>>> gdf
    geometry    name
0       LINESTRING (2.0 -2.0, 5.0 -5.0) Profile1
1       LINESTRING (2.0 -2.0, 5.0 -5.0) Profile2
```

```
>>> # Extracting X, Y, and Z coordinates from cross sections
>>> gdf_points = gg.vector.extract_xyz_from_cross_sections(profile_gdf=profile_gdf,␣
↪interfaces_gdf=gdf)
>>> gdf_points
    name        geometry                          X       Y       Z
0       Profile1        POINT (1.41421 -1.41421)        1.41    -1.41   -2.00
1       Profile1        POINT (3.53553 -3.53553)        3.54    -3.54   -5.00
2       Profile2        POINT (1.41421 -1.41421)        1.41    -1.41   -2.00
3       Profile2        POINT (3.53553 -3.53553)        3.54    -3.54   -5.00
```

See also:

calculate_coordinates_for_point_on_cross_section Calculating the coordinates for a Point on a cross section

calculate_coordinates_for_linestring_on_cross_sections Calculating the coordinates for one LineString on cross sections

calculate_coordinates_for_linestrings_on_cross_sections Calculating the coordinates for LineStrings on cross sections

extract_interfaces_coordinates_from_cross_section Extracting the coordinates of interfaces from cross sections

gemgis.vector.**extract_xyz_linestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_index: bool = True*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely LineStrings with Z components

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Shapely LineStrings with X, Y, and Z components
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing Shapely Points with appended X, Y, and Z columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Shapely LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> import geopandas as gpd
>>> linestring = LineString(([1,2,3], [4,5,6]))
>>> linestring.wkt
'LINESTRING Z (1 2 3, 4 5 6)'
```

```
>>> # Creating GeoDataFrame from LineString
>>> gdf = gpd.GeoDataFrame(geometry=[linestring, linestring])
>>> gdf
    geometry
0   LINESTRING Z (1.00000 2.00000 3.00000, 4.00000...
1   LINESTRING Z (1.00000 2.00000 3.00000, 4.00000...
```

```
>>> # Extracting X, Y, and Z Coordinates from Point Objects
>>> gdf = gg.vector.extract_xyz_linestrings(gdf=gdf)
>>> gdf
    geometry                points          X       Y       Z
0   POINT (1.00000 2.00000) (1.0, 2.0, 3.0) 1.00    2.00    3.00
1   POINT (4.00000 5.00000) (4.0, 5.0, 6.0) 4.00    5.00    6.00
2   POINT (1.00000 2.00000) (1.0, 2.0, 3.0) 1.00    2.00    3.00
3   POINT (4.00000 5.00000) (4.0, 5.0, 6.0) 4.00    5.00    6.00
```

See also:

extract_xyz_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points with Z components

extract_xyz_polygons Extracting X and Y coordinates from a GeoDataFrame containing Shapely Polygons with Z component

gemgis.vector.**extract_xyz_points**(*gdf: geopandas.geodataframe.GeoDataFrame*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely Points with Z components

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Shapely Points with X, Y, and Z components

Returns **gdf** – GeoDataFrame containing Shapely Points with appended X, Y, and Z columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Shapely Point
>>> import gemgis as gg
>>> from shapely.geometry import Point
>>> import geopandas as gpd
>>> point = Point(1,2,4)
>>> point.wkt
'POINT Z (0 0 0)'
```

```
>>> # Creating GeoDataFrame from Point
>>> gdf = gpd.GeoDataFrame(geometry=[point, point])
>>> gdf
    geometry
0   POINT Z (0.00000 0.00000 0.00000)
1   POINT Z (0.00000 0.00000 0.00000)
```

```
>>> # Extracting X, Y, and Z Coordinates from Point Objects
>>> gdf = gg.vector.extract_xyz_points(gdf=gdf)
>>> gdf
    geometry                              X       Y       Z
0   POINT Z (1.00000 2.00000 3.00000)   1.00    2.00    3.00
1   POINT Z (1.00000 2.00000 3.00000)   1.00    2.00    3.00
```

See also:

extract_xyz_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings with Z components

extract_xyz_polygons Extracting X and Y coordinates from a GeoDataFrame containing Shapely Polygons with Z component

gemgis.vector.**extract_xyz_polygons**(*gdf: geopandas.geodataframe.GeoDataFrame*, *reset_index: bool = True*, *drop_index: bool = True*) → geopandas.geodataframe.GeoDataFrame

Extracting X, Y, and Z coordinates from a GeoDataFrame containing Shapely Polygons with Z components

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing Shapely Polygons with X, Y, and Z components
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`

Returns **gdf** – GeoDataFrame containing Shapely Points with appended X, Y, and Z columns

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Shapely Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> import geopandas as gpd
>>> polygon = Polygon([[0, 0, 1], [1, 0, 1], [1, 1, 1], [0, 1, 1], [0, 0, 1]])
>>> polygon.wkt
'POLYGON Z ((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1))'
```

```
>>> # Creating GeoDataFrame from LineString
>>> gdf = gpd.GeoDataFrame(geometry=[polygon, polygon])
>>> gdf
    geometry
0       POLYGON Z ((0.00000 0.00000 1.00000, 1.00000 0...
1       POLYGON Z ((0.00000 0.00000 1.00000, 1.00000 0...
```

```
>>> # Extracting X, Y, and Z Coordinates from Point Objects
>>> gdf = gg.vector.extract_xyz_polygons(gdf=gdf)
>>> gdf
    geometry                points          X       Y       Z
0   POINT (0.00000 0.00000) [0.0, 0.0, 1.0] 0.00    0.00    1.00
1   POINT (1.00000 0.00000) [1.0, 0.0, 1.0] 1.00    0.00    1.00
2   POINT (1.00000 1.00000) [1.0, 1.0, 1.0] 1.00    1.00    1.00
3   POINT (0.00000 1.00000) [0.0, 1.0, 1.0] 0.00    1.00    1.00
```

See also:

extract_xyz_points Extracting X and Y coordinates from a GeoDataFrame containing Shapely Points with Z component

extract_xyz_linestrings Extracting X and Y coordinates from a GeoDataFrame containing Shapely LineStrings with Z components

gemgis.vector.**extract_xyz_rasterio**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: rasterio.io.DatasetReader*, *minz: float = None*, *maxz: float = None*, *reset_index: bool = True*, *drop_index: bool = True*, *drop_id: bool = True*, *drop_points: bool = True*, *drop_level0: bool = True*, *drop_level1: bool = True*, *target_crs: Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS] = None*, *bbox: Optional[Sequence[float]] = None*, *remove_total_bounds: bool = False*, *threshold_bounds: Union[float, int] = 0.1*) → geopandas.geodataframe.GeoDataFrame

Extracting X and Y coordinates from a GeoDataFrame (Points, LineStrings, MultiLineStrings Polygons) and z values from a rasterio object and returning a GeoDataFrame with X, Y, and Z coordinates as additional columns

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame created from vector data containing Shapely Points, LineStrings, MultiLineStrings or Polygons
- **dem** (*rasterio.io.DatasetReader*) – Rasterio object containing the height values
- **minz** (*float*) – Value defining the minimum elevation the data needs to be returned, e.g. `minz=50`, default `None`

- **maxz** (*float*) – Value defining the maximum elevation the data needs to be returned, e.g. `maxz=500`, default `None`
- **reset_index** (*bool*) – Variable to reset the index of the resulting GeoDataFrame, default `True`
- **drop_level0** (*bool*) – Variable to drop the level_0 column. Options include: `True` or `False`, default set to `True`
- **drop_level1** (*bool*) – Variable to drop the level_1 column. Options include: `True` or `False`, default set to `True`
- **drop_index** (*bool*) – Variable to drop the index column. Options include: `True` or `False`, default set to `True`
- **drop_id** (*bool*) – Variable to drop the id column. Options include: `True` or `False`, default set to `True`
- **drop_points** (*bool*) – Variable to drop the points column. Options include: `True` or `False`, default set to `True`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS, rasterio.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `target_crs='EPSG:4647'`
- **bbox** (*list*) – Values (minx, maxx, miny, maxy) to limit the extent of the data, e.g. `bbox=[0, 972, 0, 1069]`
- **remove_total_bounds** (*bool*) – Variable to remove the vertices representing the total bounds of a GeoDataFrame consisting of Polygons Options include: `True` or `False`, default set to `False`
- **threshold_bounds** (*Union[float, int]*) – Variable to set the distance to the total bound from where vertices are being removed, e.g. `threshold_bounds=10`, default set to `0.1`

Returns **gdf** – GeoDataFrame containing the X, Y, and Z coordinates

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0       None    Ton             POINT (19.150 293.313)
1       None    Ton             POINT (61.934 381.459)
2       None    Ton             POINT (109.358 480.946)
3       None    Ton             POINT (157.812 615.999)
4       None    Ton             POINT (191.318 719.094)
```

```
>>> # Loading raster file
>>> dem = rasterio.open(fp='dem.tif')
>>> dem
<open DatasetReader name='dem.tif' mode='r'>
```

```
>>> # Extracting X, Y, and Z Coordinates from Shapely Base Geometries and raster
>>> gdf_xyz = gg.vector.extract_xyz_rasterio(gdf=gdf, dem=dem, reset_index=reset_
↪index)
>>> gdf_xyz
    formation   geometry                X       Y       Z
0   Ton         POINT (19.150 293.313)  19.15   293.31  364.99
1       Ton             POINT (61.934 381.459)  61.93   381.46  400.34
2       Ton             POINT (109.358 480.946) 109.36  480.95  459.55
3       Ton             POINT (157.812 615.999) 157.81  616.00  525.69
4       Ton             POINT (191.318 719.094) 191.32  719.09  597.63
```

See also:

extract_xyz_array Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model as array

extract_xyz Extracting X, Y, and Z coordinates from a GeoDataFrame and Digital Elevation Model

gemgis.vector.**interpolate_raster**(*gdf: geopandas.geodataframe.GeoDataFrame*, *value: str = 'Z'*, *method: str = 'nearest'*, *n: int = None*, *res: int = 1*, *extent: List[Union[int, float]] = None*, *seed: int = None*, ***kwargs*) → numpy.ndarray

Interpolating a raster/digital elevation model from point or line Shape file

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing vector data of geom_type Point or Line containing the Z values of an area
- **value** (*str*) – Value to be interpolated, e.g. `value='Z'`, default is `'Z'`
- **method** (*string*) – Method used to interpolate the raster. Options include: `'nearest'`, `'linear'`, `'cubic'`, `'rbf'`
- **res** (*int*) – Resolution of the raster in X and Y direction, e.g. `res=50`
- **seed** (*int*) – Seed for the drawing of random numbers, e.g. `seed=1`
- **n** (*int*) – Number of samples used for the interpolation, e.g. `n=100`
- **extent** (*List[Union[float, int]]*) – Values for minx, maxx, miny and maxy values to define the boundaries of the raster, e.g. `extent=[0, 972, 0, 1069]`
- ****kwargs** (*optional keyword arguments*) – For kwargs for rbf and griddata see: https://docs.scipy.org/doc/scipy/reference/interpolate.html

Returns **array** – Array representing the interpolated raster/digital elevation model

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id  Z       geometry
0       None    400     LINESTRING (0.741 475.441, 35.629 429.247, 77....
1       None    300     LINESTRING (645.965 0.525, 685.141 61.866, 724...
2       None    400     LINESTRING (490.292 0.525, 505.756 40.732, 519...
3       None    600     LINESTRING (911.433 1068.585, 908.856 1026.831...
4       None    700     LINESTRING (228.432 1068.585, 239.772 1017.037...
```

```
>>> # Interpolating vector data
>>> raster = gg.vector.interpolate_raster(gdf=gdf, method='rbf')
>>> raster[:2]
array([[393.56371914, 393.50838517, 393.45386851, ..., 396.15856133,
    398.11421775, 400.06334288],
   [393.41982945, 393.36494645, 393.31088433, ..., 396.20694282,
    398.16690286, 400.12027997]])
```

gemgis.vector.**intersection_polygon_polygon**(*polygon1: shapely.geometry.polygon.Polygon*, *polygon2: shapely.geometry.polygon.Polygon*) → Union[shapely.geometry.linestring.LineString, shapely.geometry.polygon.Polygon]

Calculating the intersection between to Shapely Polygons

Parameters

- **polygon1** (*shapely.geometry.polygon.Polygon*) – First polygon used for intersecting, e.g. `polygon1=Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`
- **polygon2** (*shapely.geometry.polygon.Polygon*) – Second polygon used for intersecting, e.g. `polygon2=Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`

Returns **intersection** – Intersected geometry as Shapely Object

Return type Union[shapely.geometry.linestring.LineString, shapely.geometry.polygon.Polygon]

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
```

(continues on next page)

(continued from previous page)

```
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Calculating the intersection between two polygons
>>> intersection = gg.vector.intersection_polygon_polygon(polygon1=polygon1,␣
↪polygon2=polygon2)
>>> intersection.wkt
'LINESTRING (10 0, 10 10)'
```

See also:

intersections_polygon_polygons Intersecting a polygon with mutiple polygons

intersections_polygons_polygons Intersecting multiple polygons with multiple polygons

extract_xy_from_polygon_intersections Extracting intersections between multiple polygons

gemgis.vector.**intersections_polygon_polygons**(*polygon1: shapely.geometry.polygon.Polygon*, *polygons2: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) → List[shapely.geometry.base.BaseGeometry]

Calculating the intersections between one polygon and a list of polygons

Parameters

- **polygon1** (*shapely.geometry.polygon.Polygon*) – First polygon used for intersecting, e.g. polygon1=Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
- **polygons2** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) – List of polygons as list or GeoDataFrame to get intersected

Returns **intersections** – List of intersected geometries

Return type List[shapely.geometry.base.BaseGeometry]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Creating list of polygons
>>> polygons2 = [polygon2, polygon2]
```

```
>>> # Calculating the intersections between a polygon with polygons
>>> intersection = gg.vector.intersections_polygon_polygons(polygon1=polygon1,␣
↪polygons2=polygons2)
>>> intersection
[<shapely.geometry.linestring.LineString at 0x231eaf22100>,
<shapely.geometry.linestring.LineString at 0x231eab22970>]
```

```
>>> # Inspecting the first element of the list
>>> intersection[0].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Creating the second element of the list
>>> intersection[1].wkt
'LINESTRING (10 0, 10 10)'
```

See also:

intersection_polygon_polygon Intersecting a polygon with a polygon

intersections_polygons_polygons Intersecting multiple polygons with multiple polygons

extract_xy_from_polygon_intersections Extracting intersections between multiple polygons

gemgis.vector.**intersections_polygons_polygons**(*polygons1: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*, *polygons2: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) → List[shapely.geometry.base.BaseGeometry]

Calculating the intersections between a list of Polygons

Parameters

- **polygons1** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) – List of Polygons or GeoDataFrame containing Polygons to be intersected
- **polygons2** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.polygon.Polygon]]*) – List of Polygons or GeoDataFrame containing Polygons to be intersected

Returns **intersections** – List of intersected geometries

Return type List[shapely.geometry.base.BaseGeometry]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating list of polygons
>>> polygons1 = [polygon1, polygon1]
```

```
>>> # Creating second polygon
>>> polygon2 = Polygon([[10, 0], [20, 0], [20, 10], [10, 10], [10, 0]])
>>> polygon2.wkt
'POLYGON ((10 0, 20 0, 20 10, 10 10, 10 0))'
```

```
>>> # Creating list of polygons
>>> polygons2 = [polygon2, polygon2]
```

```
>>> # Calculating intersections between polygons and polygons
>>> intersection = gg.vector.intersections_polygons_polygons(polygons1=polygons1,␣
↪polygons2=polygons2)
>>> intersection
[<shapely.geometry.linestring.LineString at 0x231eaf4dd90>,
<shapely.geometry.linestring.LineString at 0x231ec6e8df0>,
 <shapely.geometry.linestring.LineString at 0x231eaf4dc70>,
 <shapely.geometry.linestring.LineString at 0x231eaf4dd00>]
```

```
>>> # Inspecting the first element of the list
>>> intersection[0].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Inspecting the second element of the list
>>> intersection[1].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Inspecting the third element of the list
>>> intersection[2].wkt
'LINESTRING (10 0, 10 10)'
```

```
>>> # Inspecting the fourth element of the list
>>> intersection[3].wkt
'LINESTRING (10 0, 10 10)'
```

See also:

intersection_polygon_polygon Intersecting a polygon with a polygon

intersections_polygon_polygons Intersecting a polygons with multiple polygons

extract_xy_from_polygon_intersections Extracting intersections between multiple polygons

gemgis.vector.**load_gpx**(*path: str*, *layer: Union[int, str] = 'tracks'*) → Collection

Loading a GPX file as collection

Parameters

- **path** (*str*) – Path to the GPX file, e.g. `path='file.gpx'`
- **layer** (*Union[int, str]*) – The integer index or name of a layer in a multi-layer dataset, e.g. `layer='tracks'`, default is `tracks`

Returns **gpx** – Collection containing the GPX data

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> gpx = gg.vector.load_gpx(path='file.gpx', layer='tracks')
>>> gpx
<open Collection 'file.gpx:tracks', mode 'r' at 0x24f1c90ffa0>
```

See also:

load_gpx_as_dict Loading a GPX file as dict

load_gpx_as_geometry Loading a GPX file as Shapely BaseGeometry

gemgis.vector.**load_gpx_as_dict**(*path: str*, *layer: Union[int, str] = 'tracks'*) → Collection

Loading a GPX file as dict

Parameters

- **path** (*str*) – Path to the GPX file, e.g. `path='file.gpx'`
- **layer** (*Union[int, str]*) – The integer index or name of a layer in a multi-layer dataset, e.g. `layer='tracks'`, default is `tracks`

Returns **gpx_dict** – Dict containing the GPX data

Return type dict

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> gpx = gg.vector.load_gpx_as_dict(path='file.gpx', layer='tracks')
>>> gpx
{'type': 'Feature',
 'id': '0',
 'properties': OrderedDict([('name',
                'First half marathon distance of the year'),
               ('cmt', None),
```

(continues on next page)

(continued from previous page)

```
              ('desc', None),
              ('src', None),
              ('link1_href', None),
              ('link1_text', None),
              ('link1_type', None),
              ('link2_href', None),
              ('link2_text', None),
              ('link2_type', None),
              ('number', None),
              ('type', '9')]),
 'geometry': {'type': 'MultiLineString',
  'coordinates': [[(8.496285, 52.705566),
    (8.49627, 52.705593),
    (8.496234, 52.705629),...]]}}
```

See also:

load_gpx_as Loading a GPX file as Collection

load_gpx_as_geometry Loading a GPX file as Shapely BaseGeometry

gemgis.vector.**load_gpx_as_geometry**(*path: str*, *layer: Union[int, str] = 'tracks'*) → shapely.geometry.base.BaseGeometry

Loading a GPX file as Shapely Geometry

Parameters

- **path** (*str*) – Path to the GPX file, e.g. `path='file.gpx'`
- **layer** (*Union[int, str]*) – The integer index or name of a layer in a multi-layer dataset, e.g. `layer='tracks'`, default is `tracks`

Returns **shape** – Shapely BaseGeometry containing the geometry data of the GPX file

Return type shapely.geometry.base.BaseGeometry

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> gpx = gg.vector.load_gpx_as_geometry(path='file.gpx', layer='tracks')
>>> gpx.wkt
'MULTILINESTRING ((8.496285 52.705566, 8.496270000000001 52.705593, 8.
↪496233999999999 52.705629, 8.496205
52.705664, 8.496181 52.705705, 8.496171 52.705754,...)
```

See also:

load_gpx Loading a GPX file as Collection

load_gpx_as_dict Loading a GPX file as dict

gemgis.vector.**remove_interfaces_within_fault_buffers**(*fault_gdf: geopandas.geodataframe.GeoDataFrame*, *interfaces_gdf: geopandas.geodataframe.GeoDataFrame*, *distance: Union[int, float] = None*, *remove_empty_geometries: bool = True*, *extract_coordinates: bool = True*) → Tuple[geopandas.geodataframe.GeoDataFrame, geopandas.geodataframe.GeoDataFrame]

Function to create a buffer around a GeoDataFrame containing fault data and removing interface points that are located within this buffer

Parameters

- **fault_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the fault data
- **interfaces_gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the interface point data
- **distance** (*float, int*) – Distance of the buffer around the geometry object, e.g. `distance=10`
- **remove_empty_geometries** (*bool*) – Variable to remove empty geometries, Options include: `True` or `False` default `True`
- **extract_coordinates** (*bool*) – Variable to extract X and Y coordinates from resulting Shapely Objects, Options include: `True` or `False` default `True`

Returns

- **gdf_out** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the vertices located outside the fault buffer
- **gdf_in** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the vertices located inside the fault buffer

New in version 1.0.x.

Example

```
>>> # Loading Libraries
>>> import gemgis as gg
>>> import geopandas as gpd
>>> from shapely.geometry import Point, LineString
```

```
>>> # Creating first Point
>>> point1 = Point(0, 0)
>>> point1.wkt
'POINT (0 0)'
```

```
>>> # Creating second Point
>>> point2 = Point(5, 0)
>>> point2.wkt
'POINT (5 0)'
```

```
>>> # Creating GeoDataFrame from Points
>>> fault_gdf = gpd.GeoDataFrame(geometry=[point1, point2])
```

```
>>> # Creating first LineString
>>> linestring1 = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(10, 0), (20, 10), (30, 20)])
>>> linestring2.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating GeoDataFrame from LineStrings
>>> buffer_objects_gdf = gpd.GeoDataFrame(geometry=[linestring1, linestring2])
```

```
>>> # Removing interfaces within fault buffers
>>> result_out, result_in = gg.vector.remove_interfaces_within_fault_buffers(fault_
↪gdf=fault_gdf, interfaces_gdf=buffer_objects_gdf, distance=10)
```

```
>>> # Inspecting the Base Geometries that remain outside
>>> result_out
    geometry                     X       Y
0       POINT (7.07107 7.07107)         7.07    7.07
1       POINT (10.00000 10.00000)       10.00   10.00
2       POINT (20.00000 20.00000)       20.00   20.00
3       POINT (10.00000 0.00000)        10.00   0.00
4       POINT (20.00000 10.00000)       20.00   10.00
5       POINT (30.00000 20.00000)       30.00   20.00
```

```
>>> # Inspecting the Base Geometries that remain inside
>>> result_in
    geometry            X       Y
0       POINT (0.00000 0.00000) 0.00    0.00
1       POINT (7.07107 7.07107) 7.07    7.07
```

See also:

remove_object_within_buffer Removing one object from one buffered object

remove_objects_within_buffer Removing several objects from one buffered object

gemgis.vector.**remove_object_within_buffer**(*buffer_object: shapely.geometry.base.BaseGeometry*, *buffered_object: shapely.geometry.base.BaseGeometry*, *distance: Union[int, float] = None*, *buffer: bool = True*) → Tuple[shapely.geometry.base.BaseGeometry, shapely.geometry.base.BaseGeometry]

Removing object from a buffered object by providing a distance

Parameters

- **buffer_object** (*shapely.geometry.base.BaseGeometry*) – Shapely object for which a buffer will be created, e.g. `buffer_object=Point(0, 0)`

- **buffered_object** (*shapely.geometry.base.BaseGeometry*) – Shapely object that will be removed from the buffer, e.g. `buffered_object=LineString([(0, 0), (10, 10), (20, 20)])`
- **distance** (*Union[float, int]*) – Distance of the buffer around the geometry object, e.g. `distance=10`, default is `None`
- **buffer** (*bool*) – Variable to create a buffer. Options include: `True` or `False`, default set to `True`

Returns

- **result_out** (*shapely.geometry.base.BaseGeometry*) – Shapely object that remains after the buffering (outside the buffer)
- **result_in** (*shapely.geometry.base.BaseGeometry*) – Shapely object that was buffered (inside the buffer)

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> point = Point(0, 0)
>>> point.wkt
'POINT (0 0)'
```

```
>>> # Creating LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Removing object within buffer
>>> result_out, result_in = gg.vector.remove_object_within_buffer(buffer_
↪object=point, buffered_object=linestring, distance=10)
```

```
>>> # Inspecting the Base Geometry that remains outside
>>> result_out.wkt
'LINESTRING (7.071067811865473 7.071067811865473, 10 10, 20 20)'
```

```
>>> # Inspecting the Base Geometry that remains inside
>>> result_in.wkt
'LINESTRING (0 0, 7.071067811865473 7.071067811865473)'
```

See also:

remove_objects_within_buffer Removing several objects from one buffered object

remove_interfaces_within_fault_buffers Removing interfaces of layer boundaries within fault line buffers

gemgis.vector.**remove_objects_within_buffer**(*buffer_object: shapely.geometry.base.BaseGeometry*, *buffered_objects_gdf: Union[geopandas.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]]*, *distance: Union[int, float] = None*, *return_gdfs: bool = False*, *remove_empty_geometries: bool = False*, *extract_coordinates: bool = False*, *buffer: bool = True*) → Tuple[Union[List[shapely.geometry.base.BaseGeometry], geopandas.geodataframe.GeoDataFrame], Union[List[shapely.geometry.base.BaseGeometry], geopandas.geodataframe.GeoDataFrame]]

Removing objects from a buffered object by providing a distance

Parameters

- **buffer_object** (*shapely.geometry.base.BaseGeometry*) – Shapely object for which a buffer will be created, e.g. `buffer_object=Point(0, 0)`
- **buffered_object_gdf** (*Union[gpd.geodataframe.GeoDataFrame, List[shapely.geometry.base.BaseGeometry]]*) – GeoDataFrame or List of Base Geometries containing Shapely objects that will be buffered by the buffer object
- **distance** (*float, int*) – Distance of the buffer around the geometry object, e.g. `distance=10`
- **return_gdfs** (*bool*) – Variable to create GeoDataFrames of the created list of Shapely Objects. Options include: `True` or `False`, default set to `False`
- **remove_empty_geometries** (*bool*) – Variable to remove empty geometries. Options include: `True` or `False`, default set to `False`
- **extract_coordinates** (*bool*) – Variable to extract X and Y coordinates from resulting Shapely Objects. Options include: `True` or `False`, default set to `False`
- **buffer** (*bool*) – Variable to create a buffer. Options include: `True` or `False`, default set to `True`

Returns

- **result_out** (*list, gpd.geodataframe.GeoDataFrame*) – List or GeoDataFrame of Shapely objects that remain after the buffering (outside the buffer)
- **result_in** (*list, gpd.geodataframe.GeoDataFrame*) – List or GeoDataFrame of Shapely objects that was buffered (inside the buffer)

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Point
>>> import gemgis as gg
>>> from shapely.geometry import Point, LineString
>>> point = Point(0, 0)
>>> point.wkt
'POINT (0 0)'
```

```
>>> # Creating first LineString
>>> linestring1 = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring1.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating second LineString
>>> linestring2 = LineString([(10, 0), (20, 10), (30, 20)])
>>> linestring2.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Create list of buffer objects
>>> buffer_objects = [linestring1, linestring2]
```

```
>>> # Removing objects within buffer
>>> result_out, result_in = gg.vector.remove_objects_within_buffer(buffer_
↪object=point, buffered_object_gdf=buffer_objects, distance=10)
```

```
>>> # Inspecting the Base Geometries that remain outside
>>> result_out
[<shapely.geometry.linestring.LineString at 0x2515421e4f0>,
<shapely.geometry.linestring.LineString at 0x2515421e3d0>]
```

```
>>> # Inspecting the Base Geometries that remain inside
>>> result_in
[<shapely.geometry.linestring.LineString at 0x2515421e310>,
<shapely.geometry.linestring.LineString at 0x2515421e6a0>]
```

See also:

remove_object_within_buffer Removing one object from one buffered object

remove_interfaces_within_fault_buffers Removing interfaces of layer boundaries within fault line buffers

gemgis.vector.**set_dtype**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dip: str = 'dip'*, *azimuth: str = 'azimuth'*, *formation: str = 'formation'*, *polarity: str = 'polarity'*, *x: str = 'X'*, *y: str = 'Y'*, *z: str = 'Z'*) → geopandas.geodataframe.GeoDataFrame

Checking and setting the dtypes of the input data GeoDataFrame

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the input vector data with uncorrected dtypes
- **dip** (*str*) – Name of the column containing the dip data, e.g `dip='dip'`
- **azimuth** (*str*) – Name of the column containing the azimuth data, e.g `azimuth='azimuth'`
- **formation** (*str*) – Name of the column containing the formation data, e.g `formation='formation'`
- **polarity** (*str*) – Name of the column containing the polarity data, e.g `polarity='polarity'`
- **x** (*str*) – Name of the column containing the x coordinates, e.g `x='X'`

- **y** (*str*) – Name of the column containing the y coordinates, e.g `y='Y'`
- **z** (*str*) – Name of the column containing the z coordinates, e.g `z='Z'`

Returns **gdf** – GeoDataFrame containing the input vector data with corrected dtypes

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
```

```
>>> # Setting the data types
>>> gdf_dtypes = gg.vector.set_dtype(gdf=gdf)
```

gemgis.vector.**sort_by_stratigraphy**(*gdf: geopandas.geodataframe.GeoDataFrame*, *stratigraphy: List[str]*, *formation_column: str = 'formation'*) → geopandas.geodataframe.GeoDataFrame

Sorting a GeoDataFrame by a provided list of Stratigraphic Units

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the unsorted input polygons
- **stratigraphy** (*List[str]*) – List containing the stratigraphic units sorted by age, e.g. `stratigraphy=['Layer1' , 'Layer2']`
- **formation_column** (*str*) – Name of the formation column, default is formation, e.g. `formation_colum='formation'`

Returns **gdf_sorted** – GeoDataFrame containing the sorted input polygons

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> import geopandas as gpd
>>> polygon1 = Polygon([(0, 0), (1, 1), (1, 0)])
>>> polygon1.wkt
'POLYGON ((0 0, 1 1, 1 0, 0 0))'
```

```
>>> # Creating second polygon
>>> polygon2 = Polygon([(0, 0), (2, 2), (2, 0)])
>>> polygon2.wkt
'POLYGON ((0 0, 2 2, 2 0, 0 0))'
```

```
>>> # Creating GeoDataFrame from polygons
>>> gdf = gpd.GeoDataFrame(geometry=[polygon1, polygon2])
>>> gdf['formation'] = ['Layer2', 'Layer1']
>>> gdf
    geometry                                                formation
0   POLYGON ((0.00000 0.00000, 1.00000 1.00000, 1....   Layer2
1   POLYGON ((10.00000 0.00000, 20.00000 0.00000, ...   Layer1
```

```
>>> # Creating stratigraphy list
>>> stratigraphy = ['Layer1' , 'Layer2']
```

```
>>> # Sorting GeoDataFrame by stratigraphy
>>> gdf_sorted = gg.vector.sort_by_stratigraphy(gdf=gdf, stratigraphy=stratigraphy)
>>> gdf_sorted
    geometry                                                formation
0   POLYGON ((10.00000 0.00000, 20.00000 0.00000, ...   Layer1
1   POLYGON ((0.00000 0.00000, 1.00000 1.00000, 1....   Layer2
```

gemgis.vector.**subtract_geom_objects**(*geom_object1: shapely.geometry.base.BaseGeometry*, *geom_object2: shapely.geometry.base.BaseGeometry*) → shapely.geometry.base.BaseGeometry

Subtracting Shapely geometry objects from each other and returning the left over object

Parameters

- **geom_object1** (*shapely.geometry.base.BaseGeometry*) – Shapely object from which other object will be subtracted, e.g. `geom_object1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])`
- **geom_object2** (*shapely.geometry.base.BaseGeometry*) – Shapely object which will be subtracted from other object e.g. `geom_object2 = Polygon([[5, 0], [15, 0], [15, 10], [5, 10], [5, 0]])`

Returns **result** – Shapely object from which the second object was subtracted

Return type shapely.geometry.base.BaseGeometry

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating Polygon
>>> import gemgis as gg
>>> from shapely.geometry import Polygon
>>> polygon1 = Polygon([[0, 0], [10, 0], [10, 10], [0, 10], [0, 0]])
>>> polygon1.wkt
'POLYGON ((0 0, 10 0, 10 10, 0 10, 0 0))'
```

```
>>> # Creating second Polygon
>>> polygon2 = Polygon([[5, 0], [15, 0], [15, 10], [5, 10], [5, 0]])
>>> polygon2.wkt
'POLYGON ((5 0, 15 0, 15 10, 5 10, 5 0))'
```

```
>>> # Subtracting geometries from each other
>>> difference = gg.vector.subtract_geom_objects(geom_object1=polygon1, geom_
↪object2=polygon2)
>>> difference.wkt
'POLYGON ((5 0, 0 0, 0 10, 5 10, 5 0))'
```

gemgis.vector.**unify_linestrings**(*linestrings: Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame]*, *crs: Union[str, pyproj.crs.crs.CRS] = None*, *return_gdf: bool = True*) → Union[List[shapely.geometry.linestring.LineString], geopandas.geodataframe.GeoDataFrame]

Unifying adjacent LineStrings to form LineStrings with multiple vertices

Parameters

- **linestrings** (*Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]*) – LineStrings consisting of two vertices representing extracted contour lines
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`
- **return_gdf** (*bool*) – Variable to either return the data as GeoDataFrame or as list of LineStrings. Options include: `True` or `False`, default set to `True`

Returns **linestrings_merged** – Merged Shapely LineStrings

Return type Union[List[shapely.geometry.linestring.LineString], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> linestrings = gpd.read_file(filename='file.shp')
>>> linestrings
    geometry                                                Z
0   LINESTRING Z (32409587.930 5780538.824 -2350.0...   -2350.00
1   LINESTRING Z (32407304.336 5777048.086 -2050.0...   -2050.00
2   LINESTRING Z (32408748.977 5778005.047 -2200.0...   -2200.00
3   LINESTRING Z (32403693.547 5786613.994 -2400.0...   -2400.00
4   LINESTRING Z (32404738.664 5782672.480 -2350.0...   -2350.00
```

```
>>> # Merging linestrings
>>> polygons_linestrings = gg.vector.unify_linestrings(linestrings=linestrings)
>>> polygons_linestrings
    geometry
0   LINESTRING Z (32331825.641 5708789.973 -200.00...
1   LINESTRING Z (32334315.359 5723032.766 -250.00...
2   LINESTRING Z (32332516.312 5722028.768 -250.00...
3   LINESTRING Z (32332712.750 5721717.561 -250.00...
4   LINESTRING Z (32332516.312 5722028.768 -250.00...
```

gemgis.vector.**unify_polygons**(*polygons: Union[List[shapely.geometry.polygon.Polygon], geopandas.geodataframe.GeoDataFrame], crs: Union[str, pyproj.crs.crs.CRS] = None, return_gdf: bool = True*) → Union[List[shapely.geometry.polygon.Polygon], geopandas.geodataframe.GeoDataFrame]

Unifying adjacent triangular polygons to form larger objects

Parameters

- **polygons** (*Union[List[shapely.geometry.polygon.Polygon], gpd.geodataframe.GeoDataFrame]*) – Triangular Shapely Polygons representing the faces of the mesh
- **crs** (*Union[str, pyproj.crs.crs.CRS]*) – Name of the CRS provided to reproject coordinates of the GeoDataFrame, e.g. `crs='EPSG:4647'`
- **return_gdf** (*bool*) – Variable to either return the data as GeoDataFrame or as list of LineStrings. Options include: `True` or `False`, default set to `True`

Returns **polygons_merged** – Merged Shapely Polygons

Return type Union[List[shapely.geometry.polygon.Polygon], gpd.geodataframe.GeoDataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> polygons = gpd.read_file(filename='file.shp')
>>> polygons
    geometry
0   POLYGON Z ((297077.414 5677487.262 -838.496, 2...
1   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
2   POLYGON Z ((297437.539 5676992.094 -816.608, 2...
3   POLYGON Z ((298031.070 5678779.547 -648.688, 2...
4   POLYGON Z ((295827.680 5680951.574 -825.328, 2...
```

```
>>> # Merging polygons
>>> polygons_merged = gg.vector.unify_polygons(polygons=polygons)
>>> polygons_merged
    geometry
0   POLYGON Z ((396733.222 5714544.109 -186.252, 3...
1   POLYGON Z ((390252.635 5712409.037 -543.142, 3...
2   POLYGON Z ((391444.965 5710989.453 -516.000, 3...
3   POLYGON Z ((388410.007 5710903.900 -85.654, 38...
4   POLYGON Z ((384393.963 5714293.104 -614.106, 3...
```

9.1.8 gemgis.visualization module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

gemgis.visualization.**add_row_to_boreholes**(*df_groups: List[pandas.core.frame.DataFrame]*) → List[pandas.core.frame.DataFrame]

Adding additional row to each borehole for further processing for 3D visualization

Parameters **df_groups** (*List[pd.DataFrame]*) – List of Pandas DataFrames containing the borehole data

Returns **df_groups** – List of Pandas DataFrames with additional row

Return type List[pd.DataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Adding row to DataFrames
>>> grouped = df.groupby(['Index'])
>>> df_groups = [grouped.get_group(x) for x in grouped.groups]
>>> list_df = gg.visualization.add_row_to_boreholes(df_groups)
>>> list_df[0]
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   NaN         GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00                  NaN
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

See also:

create_lines_from_points Creating lines from points

create_borehole_tube Creating borehole tube

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.**calculate_vector**(*dip: Union[float, int]*, *azimuth: Union[float, int]*) → numpy.ndarray

Calculating the plunge vector of a borehole section

Parameters

- **dip** (*Union[float, int]*) – Dipping value of a borehole segment, e.g. `dip=90`
- **azimuth** (*Union[float, int]*) – Dipping direction of a borehole segment, e.g. `azimuth=20`

Returns **vector** – Plunging/dipping vector of a borehole segment

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and define dip and azimuth
>>> import gemgis as gg
>>> dip = 90
>>> azimuth = 20
```

```
>>> # Calculating plunging vector
>>> vector = gg.visualization.calculate_vector(dip=dip, azimuth=azimuth)
>>> vector
array([[ 0.364824  ],
[-0.18285081],
[ 0.91294525]])
```

gemgis.visualization.**clip_seismic_data**(*seismic_data*, *cdp_start: Optional[int] = None*, *cdp_end: Optional[int] = None*) → pandas.core.frame.DataFrame

Clipping seismic data loaded with segysak to CDP defined start and end CDP values

Parameters

- **seismic_data** (*xarray.core.dataset.Dataset*) – seismic data loaded with the segysak package
- **cdp_start** (*Union[int, type(None)]*) – Value for the start CDP number, e.g. `cdp_start=100`, default is `'None'`
- **cdp_end** (*Union[int, type(None)]*) – Value for the end CDP number, e.g. `cdp_start=200`, default is `'None'`

Returns **df_seismic_data_selection** – DataFrame containing the clipped seismic data

Return type pd.DataFrame

New in version 1.0.x.

gemgis.visualization.**convert_to_rgb**(*array: numpy.ndarray*) → numpy.ndarray

Converting array values to RGB values

Parameters **array** (*np.ndarray*) – Array containing the different bands of a raster

Returns **array_stacked** – Array with converted array values to RGB values

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and showing predefined array
>>> import gemgis as gg
>>> import numpy as np
>>> array
array([[0.3647059 , 0.3647059 , 0.49411765],
[0.40784314, 0.40784314, 0.52156866],
[0.8901961 , 0.8901961 , 0.91764706],
...,
[0.59607846, 0.69803923, 0.8       ],
[0.627451  , 0.7372549 , 0.7882353 ],
[0.80784315, 0.78431374, 0.70980394]], dtype=float32)
```

```
>>> # Inspecting shape of array
>>> array.shape
(2000, 2800, 3)
```

```
>>> # Converting to RGB array
>>> array_stacked = gg.visualization.convert_to_rgb(array=array)
>>> array_stacked
array([[[ 93,  93, 126],
[104, 104, 133],
[227, 227, 234],
...,
[152, 178, 204],
[160, 188, 201],
[206, 200, 181]],
[[247, 246, 248],
[241, 240, 246],
[243, 241, 241],
...,
[150, 177, 205],
[175, 187, 177],
[232, 228, 219]]], dtype=uint8)
```

```
>>> # Inspecting shape of array
>>> array_stacked.shape
(2000, 2800, 3)
```

See also:

read_raster Reading Digital Elevation Model as xarray

drape_array_over_dem Draping an array of the Digital Elevation Model

gemgis.visualization.**create_borehole_labels**(*df: Union[pandas.core.frame.DataFrame, geopandas.geodataframe.GeoDataFrame]*) → pyvista.core.pointset.PolyData

Create labels for borehole plots.

Parameters **df** (*Union[pd.DataFrame, gpd.geodataframe.GeoDataFrame]*) – (Geo-)DataFrame containing the borehole data.

Returns **borehole_locations** – Borehole locations with labels.

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Changed in version 1.1.1: Fixed a ValueError that was introduced with pandas>2.0.0.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating borehole labels
>>> labels = gg.visualization.create_borehole_labels(df=df)
>>> labels
Header
PolyData    Information
N Cells     2
N Points    2
X Bounds    3.239e+07, 3.240e+07
Y Bounds    5.753e+06, 5.763e+06
Z Bounds    6.000e+01, 1.070e+02
N Arrays    1
Data Arrays
Name    Field   Type    N Comp  Min Max
Labels  Points          1       nan nan
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing.

create_lines_from_points Creating lines from points.

create_borehole_tube Creating borehole tube.

create_borehole_tubes Creating tubes from lines.

create_boreholes_3d Creating PyVista objects for plotting.

gemgis.visualization.**create_borehole_tube**(*df: pandas.core.frame.DataFrame*, *line: pyvista.core.pointset.PolyData*, *radius: Union[float, int]*) → pyvista.core.pointset.PolyData

Creating a tube from a line for the 3D visualization of boreholes

Parameters

- **df** (*pd.DataFrame*) – DataFrame containing the borehole data
- **line** (*pv.core.pointset.PolyData*) – PyVista line object
- **radius** (*Union[float,int]*) – Radius of the tube, e.g. `'radius=10'`

Returns **tube** – PolyData Object representing the borehole tube

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Adding row to DataFrames
>>> grouped = df.groupby(['Index'])
>>> df_groups = [grouped.get_group(x) for x in grouped.groups]
>>> list_df = gg.visualization.add_row_to_boreholes(df_groups)
>>> list_df[0]
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   NaN         GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00                  NaN
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating Lines from points
>>> line = gg.visualization.create_lines_from_points(df=list_df[0])
>>> line
```

(continues on next page)

(continued from previous page)

```
PolyData    Information
N Cells     39
N Points    20
X Bounds    3.239e+07, 3.239e+07
Y Bounds    5.763e+06, 5.763e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    0
```

```
>>> # Creating Tubes from lines
>>> tube = gg.visualization.create_borehole_tube(df=list_df[0], line=line,␣
↪radius=100)
>>> tube
Header
PolyData    Information
N Cells     418
N Points    1520
X Bounds    3.239e+07, 3.239e+07
Y Bounds    5.762e+06, 5.764e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    2
Data Arrays
Name        Field   Type    N Comp  Min         Max
scalars     Points  int32   1       0.000e+00   1.900e+01
TubeNormals Points  float32 3       -1.000e+00  1.000e+00
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_lines_from_points Creating lines from points

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.**create_borehole_tubes**(*df: pandas.core.frame.DataFrame*, *min_length: Union[float, int]*, *radius: Union[int, float] = 10*) → Tuple[List[pyvista.core.pointset.PolyData], List[pandas.core.frame.DataFrame]]

Creating PyVista Tubes for plotting boreholes in 3D

Parameters

- **df** (*pd.DataFrame*) – DataFrame containing the extracted borehole data
- **min_length** (*Union[float, int]*) – Length defining the minimum depth of boreholes to be plotted, e.g. `min_length=1000`
- **radius** (*Union[int, float]*) – Radius of the boreholes plotted with PyVista, e.g. `radius=100` default is `10` m

Returns

- **tubes** (*List[pv.core.pointset.PolyData]*) – List of PyVista PolyData Objects
- **df_groups** (*List[pd.DataFrame]*) – List of DataFrames containing the borehole data

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating borehole tubes
>>> tubes, df_groups = gg.visualization.create_borehole_tubes(df=df, min_
↪length=1000, radius=100)
>>> tubes[0]
Header
PolyData    Information
N Cells     418
N Points    1520
X Bounds    3.239e+07, 3.239e+07
Y Bounds    5.762e+06, 5.764e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    2
Data Arrays
Name        Field   Type    N Comp  Min         Max
scalars     Points  int32   1       0.000e+00   1.900e+01
TubeNormals Points  float32 3       -1.000e+00  1.000e+00
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_lines_from_points Creating lines from points

create_borehole_tube Creating borehole tube

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.**create_boreholes_3d**(*df: pandas.core.frame.DataFrame*, *min_length: Union[float, int]*, *color_dict: dict*, *radius: Union[float, int] = 10*) → Tuple[List[pyvista.core.pointset.PolyData], pyvista.core.pointset.PolyData, List[pandas.core.frame.DataFrame]]

Plotting boreholes in 3D

Parameters

- **df** (*pd.DataFrame*) – DataFrame containing the extracted borehole data

- **min_length** (*Union[float, int]*) – Value defining the minimum depth of boreholes to be plotted, e.g. `min_length=1000`
- **color_dict** (*dict*) – Dict containing the surface colors of the model
- **radius** (*Union[float, int]*) – Values of the radius of the boreholes plotted with PyVista, e.g. `radius=100`, default is `10`

Returns

- **tubes** (*List[pv.core.pointset.PolyData]*) – List of PyVista tubes
- **labels** (*pv.core.pointset.PolyData*) – PyVista PolyData with Borehole Labels
- **df_groups** (*List[pd.DataFrame]*) – List containing DataFrames

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating tubes
>>> tubes, labels, df_groups = gg.visualization.create_boreholes_3d(df=df, min_
↪length=10, color_dict=color_dict, radius=1000)
>>> tubes
Information
MultiBlock  Values
N Blocks    2
X Bounds    32385176.360, 32404939.830
Y Bounds    5751889.550, 5764283.150
Z Bounds    -5849.000, 107.000
Blocks
Index   Name        Type
0       Block-00    PolyData
1       Block-01    PolyData
```

```
>>> # Inspecting labels
>>> labels
        Header
PolyData    Information
N Cells     2
N Points    2
X Bounds    3.239e+07, 3.240e+07
Y Bounds    5.753e+06, 5.763e+06
Z Bounds    6.000e+01, 1.070e+02
N Arrays    1
```

(continues on next page)

(continued from previous page)

```
Data Arrays
Name    Field   Type    N Comp  Min Max
Labels  Points          1       nan nan
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_lines_from_points Creating lines from points

create_borehole_tube Creating borehole tube

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

gemgis.visualization.**create_delaunay_mesh_from_gdf**(*gdf: geopandas.geodataframe.GeoDataFrame*, *z: str = 'Z'*) → pyvista.core.pointset.PolyData

Creating a delaunay triangulated mesh from surface contour lines

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing LineStrings representing surface contours

Returns **mesh** – Mesh representing the triangulated mesh

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    OBJECTID    Z       EINHEIT         Shape_Leng  geometry
0   1.00        -2450   gg_kru_b_l_Z50m 3924.67     LINESTRING (32403313.109␣
↪5785053.637, 32402917...
1   2.00        -2400   gg_kru_b_l_Z50m 26332.90    LINESTRING (32410198.859␣
↪5781110.785, 32409807...
2   3.00        -2350   gg_kru_b_l_Z50m 31104.28    LINESTRING (32409587.930␣
↪5780538.824, 32408824...
3   4.00        -2300   gg_kru_b_l_Z50m 35631.73    LINESTRING (32408977.008␣
↪5779966.863, 32408808...
4   5.00        -2250   gg_kru_b_l_Z50m 41702.52    LINESTRING (32407319.922␣
↪5779788.672, 32407246...
```

```
>>> # Creating PolyData from isolines
>>> mesh = gg.visualization.create_delaunay_mesh_from_gdf(gdf=gdf)
>>> mesh
Header
PolyData    Information
N Cells     45651
```

(continues on next page)

(continued from previous page)

```
N Points    23009
X Bounds    3.233e+07, 3.250e+07
Y Bounds    5.702e+06, 5.798e+06
Z Bounds    -2.450e+03, 4.000e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -2.450e+03  4.000e+02
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

gemgis.visualization.**create_dem_3d**(*dem: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *extent: List[Union[int, float]] = None*, *res: int = 1*) → pyvista.core.pointset.StructuredGrid

Plotting the dem in 3D with PyVista

Parameters

- **dem** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Rasterio object or NumPy array containing the height values
- **extent** (*List[Union[int, float]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **res** (*int*) – Resolution of the meshgrid, e.g. `resolution=1`, default is 1

Returns **grid** – Grid storing the elevation data

Return type pyvista.core.pointset.StructuredGrid

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import rasterio
>>> raster = rasterio.open(fp='raster.tif')
```

```
>>> # Defining raster extent
>>> extent = [0, 972, 0, 1069]
```

```
>>> # Creating mesh from raster data
>>> grid = gg.visualization.create_dem_3d(dem=raster.read(1), extent=extent)
>>> grid
Header
             StructuredGrid  Information
N            Cells           1037028
```

(continues on next page)

(continued from previous page)

```
N           Points          1039068
X           Bounds          0.000e+00, 9.710e+02
Y           Bounds          0.000e+00, 1.068e+03
Z           Bounds          2.650e+02, 7.300e+02
Dimensions                  1069, 972, 1
N Arrays                    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Elevation   Points  float64 1       2.656e+02   7.305e+02
```

See also:

create_lines_3d_polydata Creating a mesh from lines

create_points_3d Creating a mesh from points

gemgis.visualization.**create_depth_map**(*mesh: pyvista.core.pointset.PolyData*, *name: str = 'Depth [m]'*) → pyvista.core.pointset.PolyData

Extracting the depth values of the vertices and add them as scalars to the mesh

Parameters

- **mesh** (*pv.core.pointset.PolyData*) – PyVista PolyData dataset
- **name** (*str*) – Name of the data array, e.g. `name='Depth [m]'`, default is `'Depth [m]'`

Returns **mesh** – PyVista PolyData dataset with depth values as data array

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> mesh = pv.read(filename='mesh.vtk')
>>> mesh
PolyData    Information
N Cells     4174
N Points    2303
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    0
```

```
>>> # Creating depth map from surface
>>> mesh = gg.visualization.create_depth_map(mesh=mesh)
>>> mesh
Header
PolyData    Information
N Cells     4174
N Points    2303
```

(continues on next page)

(continued from previous page)

```
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    1
Data Arrays
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       3.050e+02   7.250e+02
```

See also:

create_depth_maps_from_gempy Creating depth maps from GemPy Model Surfaces

create_thickness_maps Creating thickness map from PolyData datasets

create_temperature_map Creating temperature map from PolyData datasets

gemgis.visualization.**create_depth_maps_from_gempy**(*geo_model*, *surfaces: Union[str, List[str]]*) → Dict[str, List[Union[pyvista.core.pointset.PolyData, numpy.ndarray, List[str]]]]

Creating depth map of model surfaces, adapted from https://github.com/cgre-aachen/gempy/blob/20550fffdd1ccb3c6a9a402bc162e7eed3dd7352/gempy/plot/vista.py#L440-L477

Parameters

- **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model
- **surfaces** (*Union[str, List[str]]*) – Name of the surface or list with surface names of which the depth maps are created, e.g. `surfaces=['Layer1', 'Layer2']`

Returns **surfaces_poly** – Dict containing the mesh data, depth data and color data for selected surfaces

Return type Dict[str, List[Union[pv.core.pointset.PolyData, np.ndarray, List[str]]]]

New in version 1.0.x.

Changed in version 1.1.8: Ensure compatibility with GemPy>=3

Example

```
>>> # Loading Libraries and creating depth map
>>> import gemgis as gg
>>> dict_sand1 = gg.visualization.create_depth_maps(geo_model=geo_model, surfaces=
↪'Sand1')
>>> dict_sand1
{'Sand1':   [PolyData (0x2dd0f46c820)
N Cells:    4174
N Points:   2303
X Bounds:   9.720e+00, 9.623e+02
Y Bounds:   1.881e+02, 9.491e+02
Z Bounds:   3.050e+02, 7.250e+02
N Arrays:   1,
'#015482']}
```

See also:

create_depth_map Creating depth map from PolyData dataset

create_thickness_maps Creating thickness map from PolyData datasets

create_temperature_map Creating temperature map from PolyData datasets

gemgis.visualization.**create_deviated_borehole_df**(*df_survey: pandas.core.frame.DataFrame*, *position: Union[numpy.ndarray, shapely.geometry.point.Point]*, *depth: str = 'depth'*, *dip: str = 'dip'*, *azimuth: str = 'azimuth'*) → pandas.core.frame.DataFrame

Creating Pandas DataFrame containing parameters to create 3D boreholes

Parameters

- **df_survey** (*pd.DataFrame*) – Pandas DataFrame containing the survey data of one borehole
- **position** (*np.ndarray*) – NumPy array containing the X, Y, and Z coordinates/the position of the borehole, e.g. `position = np.array([12012.68053 , 30557.53476 , 2325.532416])`
- **depth** (*str*) – Name of the column that contains the depth values, e.g. `depth='depth'`, default is `'depth'`
- **dip** (*str*) – Name of the column that contains the dip values, e.g. `dip='dip'`, default is `'dip'`
- **azimuth** (*str*) – Name of the column that contains the azimuth values, e.g. `azimuth='azimuth'` default is `'azimuth'`

Returns **df_survey** – Pandas DataFrame containing parameters to create 3D boreholes

Return type pd.DataFrame

New in version 1.0.x.

Changed in version 1.1.7.

Replace pandas append with concat.

Example

```
>>> # Loading Libraries and file
>>> import gemgis as gg
>>> import pandas as pd
>>> df_survey = pd.read_csv('survey.csv')
    holeid      depth   dip     azimuth
0   SonicS_006  0       90.00   20
1   SonicS_006  10      89.50   20
2   SonicS_006  20      89.00   20
3   SonicS_006  30      88.50   20
4   SonicS_006  40      88.00   20
```

```
>>> # Defining the position of the borehole at the surface
>>> position = np.array([12012.68053 , 30557.53476 ,  2325.532416])
```

```
>>> # Creating the survey DataFrame with additional parameters
>>> df_survey = gg.visualization.create_deviated_well_df(df_survey=df_survey,
↪position=position)
```

gemgis.visualization.**create_deviated_boreholes_3d**(*df_collar: pandas.core.frame.DataFrame*, *df_survey: pandas.core.frame.DataFrame*, *min_length: Union[float, int]*, *radius: Union[float, int] = 10*, *collar_depth: str = 'Depth'*, *survey_depth: str = 'Depth'*, *index: str = 'Index'*, *dip: str = 'dip'*, *azimuth: str = 'azimuth'*) → Tuple[List[pyvista.core.pointset.PolyData], pyvista.core.pointset.PolyData, List[pandas.core.frame.DataFrame]]

Plotting boreholes in 3D

Parameters

- **df_collar** (*pd.DataFrame*) – DataFrame containing the extracted borehole data
- **df_survey** (*pd.DataFrame*) – DataFrame containing the extracted borehole survey data
- **min_length** (*Union[float, int]*) – Value defining the minimum depth of boreholes to be plotted, e.g. `min_length=1000`
- **color_dict** (*dict*) – Dict containing the surface colors of the model
- **radius** (*Union[float, int]*) – Values of the radius of the boreholes plotted with PyVista, e.g. `radius=100`, default is `10`
- **collar_depth** (*str*) – Name of the column that contains the depth values, e.g. `collar_depth='depth'`, default is `'Depth'`
- **survey_depth** (*str*) – Name of the column that contains the depth values, e.g. `survey_depth='depth'`, default is `'Depth'`
- **index** (*str*) – Name of the column that contains the index values, e.g. `index='index'`, default is `'index'`
- **dip** (*str*) – Name of the column that contains the dip values, e.g. `dip='dip'`, default is `'dip'`
- **azimuth** (*str*) – Name of the column that contains the azimuth values, e.g. `azimuth='azimuth'` default is `'azimuth'`

Returns

- **tubes** (*List[pv.core.pointset.PolyData]*) – List of PyVista tubes
- **labels** (*pv.core.pointset.PolyData*) – PyVista PolyData with Borehole Labels
- **df_groups** (*List[pd.DataFrame]*) – List containing DataFrames

New in version 1.0.x.

Example

gemgis.visualization.**create_lines_3d_linestrings**(*gdf: geopandas.geodataframe.GeoDataFrame*, *dem: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *extent: List[Union[int, float]] = None*) → geopandas.geodataframe.GeoDataFrame

Creating lines with z-component (LineString Z)

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the LineStrings to be converted to linestrings with z-component
- **dem** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Rasterio object or NumPy array containing the height values
- **extent** (*List[Union[int, float]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`

Returns **gdf_3d** – GeoDataFrame containing the LineStrings with Z component (LineString Z)

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> import rasterio
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
id      formation   geometry
0   None    Unterjura   LINESTRING (32522415.430 5777985.396, 32521520...
1   None    Unterjura   LINESTRING (32479802.616 5782183.163, 32480593...
2   None    Mitteljura  LINESTRING (32522376.263 5779907.729, 32520580...
3   None    Mitteljura  LINESTRING (32463272.196 5788327.350, 32464107...
```

```
>>> # Loading Digital Elevation Model
>>> dem = rasterio.open('raster.tif')
```

```
>>> # Create LineStrings with Z-component
>>> gdf_3d = gg.visualization.create_lines_3d_linestrings(gdf=gdf, dem=dem)
>>> gdf_3d
    id      formation   geometry
0   None    Unterjura   LINESTRING Z (32522415.430 5777985.396 213.000...
1   None    Unterjura   LINESTRING Z (32479802.616 5782183.163 84.000,...
2   None    Mitteljura  LINESTRING Z (32522376.263 5779907.729 116.000...
3   None    Mitteljura  LINESTRING Z (32463272.196 5788327.350 102.000...
```

See also:

create_lines_3d_polydata Creating lines with z-component for the plotting with PyVista

create_dem_3d Creating a mesh from a Digital Elevation Model

create_points_3d Creating a mesh from points

gemgis.visualization.**create_lines_3d_polydata**(*gdf: geopandas.geodataframe.GeoDataFrame*) → pyvista.core.pointset.PolyData

Creating lines with z-component for the plotting with PyVista

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the contour information

Returns **poly** – PyVista Polydata Set containing the lines and vertices

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      Z   geometry
0   None    400 LINESTRING (0.741 475.441, 35.629 429.247, 77....
1   None    300 LINESTRING (645.965 0.525, 685.141 61.866, 724...
2   None    400 LINESTRING (490.292 0.525, 505.756 40.732, 519...
3   None    600 LINESTRING (911.433 1068.585, 908.856 1026.831...
4   None    700 LINESTRING (228.432 1068.585, 239.772 1017.037...
```

```
>>> # Create mesh from LineStrings
>>> polydata = gg.visualization.create_lines_3d_polydata(gdf=gdf)
>>> polydata
    PolyData    Information
N   Cells       7
N   Points      121
X   Bounds      7.409e-01, 9.717e+02
Y   Bounds      5.250e-01, 1.069e+03
Z   Bounds      3.000e+02, 7.000e+02
N   Arrays      0
```

See also:

create_dem_3d Creating a mesh from a Digital Elevation Model

create_points_3d Creating a mesh from points

gemgis.visualization.**create_lines_from_points**(*df: pandas.core.frame.DataFrame*) → pyvista.core.pointset.PolyData

Creating a line set from a Pandas DataFrame

Parameters **df** (*pd.DataFrame*) – Pandas DataFrame containing the data for one borehole

Returns **poly** – Creating borehole traces from points

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
>>> df
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Adding row to DataFrames
>>> grouped = df.groupby(['Index'])
>>> df_groups = [grouped.get_group(x) for x in grouped.groups]
>>> list_df = gg.visualization.add_row_to_boreholes(df_groups)
>>> list_df[0]
    Unnamed: 0  Index   Name                              X           Y           Z  ␣
↪    Altitude    Depth   formation       geometry
0   NaN         GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00                  NaN
0   2091        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  27.
↪00   107.00      5956.00 OberCampanium   POINT (32386176.36 5763283.15)
1   2092        GD1017  ForschungsbohrungMünsterland1   32386176.36 5763283.15  -
↪193.00 107.00      5956.00 UnterCampanium  POINT (32386176.36 5763283.15)
```

```
>>> # Creating Lines from points
>>> line = gg.visualization.create_lines_from_points(df=list_df[0])
>>> line
PolyData    Information
N Cells     39
N Points    20
X Bounds    3.239e+07, 3.239e+07
Y Bounds    5.763e+06, 5.763e+06
Z Bounds    -5.849e+03, 1.070e+02
N Arrays    0
```

See also:

add_row_to_boreholes Adding a row to each borehole for later processing

create_borehole_tube Creating borehole tube

create_borehole_tubes Creating tubes from lines

create_borehole_labels Creating labels for boreholes

create_boreholes_3d Creating PyVista objects for plotting

gemgis.visualization.**create_mesh_from_cross_section**(*linestring: shapely.geometry.linestring.LineString*, *zmax: Union[float, int]*, *zmin: Union[float, int]*) → pyvista.core.pointset.PolyData

Creating a PyVista Mesh from one cross section

Parameters

- **linestring** (*shapely.geometry.linestring.LineString*) – LineString representing the trace of the cross section on a geological map, e.g. `linestring = LineString([(0, 0), (10, 10), (20, 20)])`
- **zmax** (*Union[float, int]*) – Upper vertical extent of the cross section, e.g. `zmax=1000`
- **zmin** (*Union[float, int]*) – Lower vertical extent of the cross section, e.g. `zmin=0`

Returns **surface** – Mesh defining the cross section in space

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating LineString
>>> import gemgis as gg
>>> from shapely.geometry import LineString
>>> linestring = LineString([(0, 0), (10, 10), (20, 20)])
>>> linestring.wkt
'LINESTRING (0 0, 10 10, 20 20)'
```

```
>>> # Creating PolyData from LineStrings
>>> polydata = gg.visualization.create_mesh_from_cross_
↪section(linestring=linestring, zmax=1000, zmin=0)
>>> polydata
Header
PolyData    Information
N Cells     4
N Points    6
X Bounds    0.000e+00, 2.000e+01
Y Bounds    0.000e+00, 2.000e+01
Z Bounds    0.000e+00, 1.000e+03
N Arrays    1
Data Arrays
Name                Field   Type    N Comp  Min         Max
Texture Coordinates Points  float64 2       0.000e+00   1.000e+00
```

See also:

create_meshes_from_cross_sections Creating meshes from cross sections

gemgis.visualization.**create_meshes_from_cross_sections**(*gdf: geopandas.geodataframe.GeoDataFrame*) → List[pyvista.core.pointset.PolyData]

Creating PyVista Meshes from multiple cross section

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the traces of the profiles as LineStrings

Returns **meshes_list** – List containing the meshes of all profiles

Return type List[pyvista.core.pointset.PolyData]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      zmax    zmin    name        geometry
0   None    500     -6000   Muenster    LINESTRING (32386148.890 5763304.720,␣
↪32393549...
1   None    500     -2000   Rheine      LINESTRING (32402275.136 5761828.501,␣
↪32431165...
```

```
>>> # Creating list of PolyData datasets from GeoDataFrame
>>> meshes_list = gg.visualization.create_meshes_from_cross_sections(gdf=gdf)
>>> meshes_list
[PolyData (0x2526e543ee0)
N Cells:    20
N Points:   22
X Bounds:   3.239e+07, 3.242e+07
Y Bounds:   5.717e+06, 5.763e+06
Z Bounds:   -6.000e+03, 5.000e+02
N Arrays:   1,
PolyData (0x2526a4687c0)
N Cells:    2
N Points:   4
X Bounds:   3.240e+07, 3.243e+07
Y Bounds:   5.762e+06, 5.814e+06
Z Bounds:   -2.000e+03, 5.000e+02
N Arrays:   1]
```

See also:

create_mesh_from_cross_section Creating a mesh from a cross section

gemgis.visualization.**create_meshes_hypocenters**(*gdf: geopandas.geodataframe.GeoDataFrame*, *magnitude: str = 'Magnitude'*, *magnitude_factor: int = 200*, *year: str = 'Year'*) → pyvista.core.composite.MultiBlock

Plotting earthquake hypocenters with PyVista

Parameters

- **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the earthquake hypocenter data
- **magnitude** (*str*) – Name for the column containing the magnitude value, e.g. `magnitude='Magnitude'`, default is `'Magnitude'`
- **magnitude_factor** (*int*) – Scaling factor for the magnitude values defining the size of the spheres, e.g. `magnitude_factor=200`, default is `200`

- **year** (*str*) – Name for the column containing the year of each earthquake event, e.g. `year='Year'`, default to `'Year'`

Returns **spheres** – PyVista MultiBlock object containing the hypocenters stored as spheres

Return type pv.core.composite.MultiBlock

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    Y           X           Z           RASTERVALU  Tiefe [km]  Magnitude  ␣
↪Epizentrum      Year    geometry
0   5645741.63  32322660.15 -8249.25    150.75      8.40        1.50       ␣
↪STETTERNICH     2002    POINT (32322660.151 5645741.630)
1   5645947.18  32323159.51 89.63       89.63       0.00        0.80       ␣
↪SOPHIENHOEHE    2014    POINT (32323159.505 5645947.183)
```

```
>>> # Creating Spheres for hypocenters
>>> spheres = gg.visualization.create_meshes_hypocenters(gdf=gdf)
>>> spheres
Information
MultiBlock  Values
N Blocks    497
X Bounds    32287780.000, 32328260.000
Y Bounds    5620074.000, 5648385.000
Z Bounds    -24317.020, 309.130
Blocks
Index   Name        Type
0       Block-00    PolyData
1       Block-01    PolyData
2       Block-02    PolyData
```

gemgis.visualization.**create_points_3d**(*gdf: geopandas.geodataframe.GeoDataFrame*) → pyvista.core.pointset.PolyData

Plotting points in 3D with PyVista

Parameters **points** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the points including X, Y, and Z columns

Returns **points_mesh** – PyVista PolyData Pointset

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   geometry
0   None    Ton         POINT (19.150 293.313)
1   None    Ton         POINT (61.934 381.459)
2   None    Ton         POINT (109.358 480.946)
3   None    Ton         POINT (157.812 615.999)
4   None    Ton         POINT (191.318 719.094)
```

```
>>> # Creating PolyData from points
>>> polydata = gg.visualization.create_points_3d(gdf=gdf)
>>> polydata
    PolyData    Information
N Cells     41
N Points        41
X Bounds        8.841e+00, 9.661e+02
Y Bounds        1.650e+02, 1.045e+03
Z Bounds        2.769e+02, 7.220e+02
N Arrays        0
```

See also:

create_lines_3d_polydata Creating a mesh from lines

create_dem_3d Creating a mesh from a Digital Elevation model

gemgis.visualization.**create_polydata_from_dxf**(*gdf: geopandas.geodataframe.GeoDataFrame*) → pyvista.core.pointset.PolyData

Converting loaded DXF object to PyVista PolyData

Parameters **gdf** (*gpd.geodataframe.GeoDataFrame*) – GeoDataFrame containing the faces/polygons of the loaded DXF object

Returns **polydata** – PyVista PolyData containing the mesh values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.dxf')
>>> gdf
    geometry
0   POLYGON Z ((1.00869 0.92852 1.00000, 0.97744 0...
1   POLYGON Z ((1.00869 0.92852 1.00000, 1.01735 0...
```

(continues on next page)

(continued from previous page)

```
2   POLYGON Z ((0.97744 0.92853 1.00000, 0.94619 0...
3   POLYGON Z ((0.97744 0.92853 1.00000, 0.98610 0...
4   POLYGON Z ((0.94619 0.92853 1.00000, 0.91494 0...
```

```
>>> # Creating PolyData from dxf file
>>> polydata = gg.visualization.create_polydata_from_dxf(gdf=gdf)
>>> polydata
PolyData    Information
N Cells     98304
N Points    393216
X Bounds    -1.576e+00, 2.530e+00
Y Bounds    -9.751e+00, 1.000e+00
Z Bounds    -9.167e-01, 1.000e+00
N Arrays        0
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.**create_polydata_from_msh**(*data: Dict[str, numpy.ndarray]*) → pyvista.core.pointset.PolyData

Converting loaded Leapfrog mesh to PyVista PolyData

Parameters **data** (*Dict[str, np.ndarray]*) – Dict containing the data loaded from a Leapfrog mesh with read_msh() of the raster module

Returns **polydata** – PyVista PolyData containing the mesh values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> data = gg.raster.read_msh('mesh.msh')
>>> data
{'Tri': array([[    0,     1,     2],
[    0,     3,     1],
[    4,     3,     0],
...,
[53677, 53672, 53680],
[53679, 53677, 53680],
[53673, 53672, 53677]]),
'Location': array([[ 1.44625109e+06,  5.24854344e+06, -1.12743862e+02],
```

(continues on next page)

(continued from previous page)

```
[ 1.44624766e+06,  5.24854640e+06, -1.15102216e+02],
[ 1.44624808e+06,  5.24854657e+06, -1.15080548e+02],
...,
[ 1.44831008e+06,  5.24896679e+06, -1.24755449e+02],
[ 1.44830385e+06,  5.24896985e+06, -1.33694397e+02],
[ 1.44829874e+06,  5.24897215e+06, -1.42506587e+02]])}
```

```
>>> # Creating PolyData from msh file
>>> polydata = gg.visualization.create_polydata_from_msh(data=data)
>>> polydata
PolyData    Information
N Cells     107358
N Points    53681
X Bounds    1.444e+06, 1.449e+06
Y Bounds    5.246e+06, 5.249e+06
Z Bounds    -2.464e+02, 7.396e+02
N Arrays    0
```

See also:

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.**create_polydata_from_ts**(*data: Tuple[list, list]*, *concat: bool = False*) → pyvista.core.pointset.PolyData

Converting loaded GoCAD mesh to PyVista PolyData

Parameters

- **data** (*Tuple[list, list]*) – Tuple containing the data loaded from a GoCAD mesh with read_ts() of the raster module
- **concat** (*bool*) – Boolean defining whether the DataFrames should be concatenated or not

Returns **polydata** – PyVista PolyData containing the mesh values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> vertices, faces = gg.raster.read_ts('mesh.ts')
```

```
>>> # Inspecting vertices
>>> vertices
    id  X           Y           Z
0   0   297077.41   5677487.26  -838.50
1   1   297437.54   5676992.09  -816.61
```

```
>>> # Inspecting
>>> faces
array([[    0,     1,     2],
[    3,     2,     4],
[    1,     5,     6],...,
[40335, 40338, 40336],
[40339, 40340, 40341],
[40341, 40342, 40339]])
```

```
>>> # Creating PolyData from ts file
>>> polydata = gg.visualization.create_polydata_from_ts((vertices, faces))
>>> polydata
PolyData    Information
N Cells     29273
N Points    40343
X Bounds    2.804e+05, 5.161e+05
Y Bounds    5.640e+06, 5.833e+06
Z Bounds    -8.067e+03, 1.457e+02
N Arrays    0
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.**create_structured_grid_from_asc**(*data: dict*) → pyvista.core.pointset.StructuredGrid

Converting loaded ASC object to PyVista StructuredGrid

Parameters **data** (*dict*) – Dict containing the extracted ASC data using read_asc(...) of the raster module

Returns **grid** – PyVista StructuredGrid created from ASC data

Return type pv.core.pointset.StructuredGrid

New in version 1.0.x.

Example

```
>>> # Loading Libraries and data
>>> import gemgis as gg
>>> data = gg.raster.read_asc('raster.asc')
```

```
>>> # Creating StructuredGrid from data
>>> grid = gg.visualization.create_structured_grid_from_asc(data=data)
>>> grid
Header  Data Arrays
StructuredGrid  Information
N Cells         2880012
N Points        2883540
X Bounds        -4.225e+04, 2.788e+05
Y Bounds        3.060e+05, 8.668e+05
Z Bounds        -1.000e+05, 2.880e+02
Dimensions      2244, 1285, 1
N Arrays        1
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -1.132e+04  2.887e+02
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_zmap Creating StructuredGrid vom Petrel ZMAP Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.**create_structured_grid_from_zmap**(*data: dict*) → pyvista.core.pointset.StructuredGrid

Converting loaded ZMAP object to PyVista StructuredGrid

Parameters **data** (*dict*) – Dict containing the extracted ZAMP data using read_zmap(...) of the raster module

Returns **grid** – PyVista StructuredGrid created from zmap data

Return type pv.core.pointset.StructuredGrid

New in version 1.0.x.

Example

```
>>> # Loading Libraries and data
>>> import gemgis as gg
>>> data = gg.raster.read_zmap('raster.dat')
```

```
>>> # Creating StructuredGrid from data
>>> grid = gg.visualization.create_structured_grid_from_zmap(data=data)
>>> grid
```

(continues on next page)

(continued from previous page)

```
Header  Data Arrays
StructuredGrid  Information
N Cells         2880012
N Points        2883540
X Bounds        -4.225e+04, 2.788e+05
Y Bounds        3.060e+05, 8.668e+05
Z Bounds        -1.000e+05, 2.880e+02
Dimensions      2244, 1285, 1
N Arrays        1
Name        Field   Type    N Comp  Min         Max
Depth [m]   Points  float64 1       -1.132e+04  2.887e+02
```

See also:

create_polydata_from_msh Creating PolyData dataset from Leapfrog mesh file

create_polydata_from_ts Creating PolyData dataset from GoCAD Tsurface file

create_polydata_from_dxf Creating PolyData dataset from DXF object

create_structured_grid_from_asc Creating StructuredGrid vom ESRI ASC Grid

create_delaunay_mesh_from_gdf Create Mesh from GeoDataFrame containing contour lines

gemgis.visualization.**create_temperature_map**(*dem: rasterio.io.DatasetReader*, *mesh: pyvista.core.pointset.PolyData*, *name: str = 'Thickness [m]'*, *apply_threshold: bool = True*, *tsurface: Union[float, int] = 10*, *gradient: Union[float, int] = 0.03*) → pyvista.core.pointset.PolyData

Creating a temperature map for a surface at depth taking the topography into account

Parameters

- **dem** (*rasterio.io.DatasetReader*) – Digital Elevation Model of the area
- **mesh** (*pv.core.pointset.PolyData*) – PolyData dataset for which the temperature at depth will be calculated
- **name** (*str*) – Name of the array to be added to the mesh, e.g. `name='Thickness [m]'`, default is `'Thickness [m]'`
- **apply_threshold** (*bool*) – Variable to apply a threshold to the mesh to remove vertices that were located above the topography. Options include: `True` or `False`, default set to `True`
- **tsurface** (*Union[float, int]*) – Surface temperature in degrees Celsius, e.g. `tsurface=10`, default is `10` degrees C
- **gradient** (*Union[float, int]*) – Geothermal gradient in degrees celsius per meter, e.g. `gradient=0.03`, default is `0.03` degrees C per m

Returns **mesh** – PolyData dataset including a temperature data array

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and Files
>>> import gemgis as gg
>>> import rasterio
>>> import pyvista as pv
>>> dem = rasterio.open(fp='raster.tif')
>>> mesh = pv.read(filename='mesh1.vtk')
>>> mesh
PolyData    Information
N Cells     4174
N Points    2303
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    0
```

```
>>> # Creating temperature map
>>> mesh = gg.visualization.create_temperature_map(dem=dem, mesh=mesh)
>>> mesh
Header
UnstructuredGrid    Information
N Cells     3946
N Points    2130
X Bounds    9.720e+00, 9.623e+02
Y Bounds    1.881e+02, 9.491e+02
Z Bounds    3.050e+02, 7.250e+02
N Arrays    2
Data Arrays
Name                Field   Type    N Comp  Min         Max
Thickness [m]       Points  float64 1       9.321e-02   2.020e+02
Temperature [°C]    Points  float64 1       1.000e+01   1.606e+01
```

See also:

create_depth_map Creating depth map from PolyData dataset

create_depth_maps_from_gempy Creating depth maps from GemPy Model Surfaces

create_thickness_maps Creating thickness map from PolyData datasets

gemgis.visualization.**create_thickness_maps**(*top_surface: pyvista.core.pointset.PolyData*, *base_surface: pyvista.core.pointset.PolyData*) → pyvista.core.pointset.PolyData

Creating a thickness map using https://docs.pyvista.org/examples/01-filter/distance-between-surfaces.html#sphx-glr-examples-01-filter-distance-between-surfaces-py

Parameters

- **top_surface** (*pv.core.pointset.PolyData*) – Mesh representing the top of the layer
- **base_surface** (*pv.core.pointset.PolyData*) – Mesh representing the base of the layer

Returns **thickness** – Mesh with scalars representing the thickness of the layer

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and creating thickness map
>>> import gemgis as gg
>>> dict_all = gg.visualization.create_depth_maps_from_gempy(geo_model=geo_model,␣
↪surfaces=['Sand1', 'Ton'])
>>> thickness_map = gg.visualization.create_thickness_maps(top_surface=dict_all[
↪'Sand1'][0], base_surface=dict_all['Ton'][0])
>>> thickness_map
Header
PolyData    Information
N Cells     5111
N Points    2739
X Bounds    9.720e+00, 9.623e+02
Y Bounds    3.578e+02, 1.058e+03
Z Bounds    3.050e+02, 7.265e+02
N Arrays    3
Data Arrays
Name            Field   Type    N Comp  Min             Max
Data            Points  float64 1       3.050e+02   7.265e+02
Normals         Points  float32 3       -9.550e-01  6.656e-01
Thickness [m]   Points  float64 1       4.850e+01   8.761e+01
```

See also:

create_depth_map Creating depth map from PolyData dataset

create_depth_maps_from_gempy Creating depth maps from GemPy Model Surfaces

create_temperature_map Creating temperature map from PolyData datasets

gemgis.visualization.**drape_array_over_dem**(*array: numpy.ndarray*, *dem: Union[rasterio.io.DatasetReader, numpy.ndarray]*, *extent: List[Union[int, float]] = None*, *zmax: Union[float, int] = 10000*, *resize_array: bool = True*)

Creating grid and texture to drape array over a digital elevation model

Parameters

- **array** (*np.ndarray*) – Array containing the map data such as a WMS Map
- **dem** (*Union[rasterio.io.DatasetReader, np.ndarray]*) – Digital elevation model where the array data is being draped over
- **extent** (*List[Union[float, int]]*) – List containing the bounds of the raster, e.g. `extent=[0, 972, 0, 1069]`
- **zmax** (*Union[float, int]*) – Maximum z value to limit the elevation data, e.g. `zmax=1000`
- **resize_array** (*bool*) – Whether to resize the array or the dem if the shape of the dem does not match the shape of the array Options include: `True` or `False`, default set to `True`

Returns

- **mesh** (*pyvista.core.pointset.PolyData*) – Mesh containing the Digital elevation model data

- **texture** (*pyvista.core.objects.Texture*) – PyVista Texture containing the map data

New in version 1.0.x.

Changed in version 1.1: Function now allows rasters with different sizes and resizes one of the rasters automatically

Changed in version 1.1.2: Edit zmax value and fixing a bug with the scikit-image resize function, see https://github.com/cgre-aachen/gemgis/issues/303

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> array
array([[[ 93,  93, 126],
[104, 104, 133],
[227, 227, 234],
...,
[152, 178, 204],
[160, 188, 201],
[206, 200, 181]],
[[247, 246, 248],
[241, 240, 246],
[243, 241, 241],
...,
[150, 177, 205],
[175, 187, 177],
[232, 228, 219]]], dtype=uint8)
```

```
>>> # Inspecting Digital Elevation Model values
>>> dem
array([[  0.  ,   0.  ,   0.  , ...,  40.1 ,  40.09,  44.58],
[  0.  ,   0.  ,   0.  , ...,  40.08,  40.07,  44.21],
[  0.  ,   0.  ,   0.  , ...,  40.14,  44.21,  43.98],
...,
[100.56, 102.14, 102.17, ...,   0.  ,   0.  ,   0.  ],
[ 99.44,  99.85,  99.77, ...,   0.  ,   0.  ,   0.  ],
[ 88.32,  91.76,  98.68, ...,   0.  ,   0.  ,   0.  ]],
dtype=float32)
```

```
>>> # Draping mesh over array
>>> mesh, texture = gg.visualization.drape_array_over_dem(array=array, dem=dem)
>>> mesh
Header
StructuredGrid  Information
N Cells         5595201
N Points        5600000
X Bounds        3.236e+07, 3.250e+07
Y Bounds        5.700e+06, 5.800e+06
Z Bounds        0.000e+00, 5.038e+02
Dimensions      2000, 2800, 1
N Arrays        1
```

(continues on next page)

(continued from previous page)

```
Data Arrays
Name                 Field   Type    N Comp  Min         Max
Texture Coordinates Points  float32 2       -7.077e-06  1.000e+00
```

```
>>> # Inspecting the texture
>>> texture
(Texture)00000151B91F3AC0
```

See also:

read_raster Reading Digital Elevation Model as xarray

convert_to_rgb Converting bands to RGB values for plotting

gemgis.visualization.**get_batlow_cmap**() → matplotlib.colors.ListedColormap

Returning the Batlow cmap from https://github.com/callumrollo/cmcrameri/blob/master/cmcrameri/cmaps/batlow.txt

Returns **cmap_batlow** – Batlow color map

Return type matplotlib.colors.ListedColormap

New in version 1.0.x.

gemgis.visualization.**get_color_lot**(*geo_model*, *lith_c: pandas.core.frame.DataFrame = None*, *index='surface'*, *is_faults: bool = True*, *is_basement: bool = False*) → pandas.core.series.Series

Method to get the right color list depending on the type of plot. Borrowed from https://github.com/cgre-aachen/gempy/blob/6aed72a4dfa26830df142a0461294bd9d21a4fa4/gempy/plot/vista.py#L133-L167

Parameters

- **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model
- **lith_c** (*pd.DataFrame*) – Pandas Series with index surface names and values hex strings with the colors
- **index** (*str*) – Index provided as string, e.g. `index='surface'`, default is `'surface'`
- **is_faults** (*bool*) – Return the colors of the faults. This should be true for surfaces and input data and false for scalar values. Options include `True` and `False`, default is `True`
- **is_basement** (*bool*) – Return or not the basement. This should be true for the lith block and false for surfaces and input data. Options include `True` and `False`, default is `False`

New in version 1.0.x.

gemgis.visualization.**get_mesh_geological_map**(*geo_model*) → Tuple[pyvista.core.pointset.PolyData, matplotlib.colors.ListedColormap, bool]

Getting the geological map of a GemPy Model draped over the topography as mesh. Borrowed from https://github.com/cgre-aachen/gempy/blob/6aed72a4dfa26830df142a0461294bd9d21a4fa4/gempy/plot/vista.py#L512-L604

Parameters **geo_model** (*gp.core.model.Project*) – Previously calculated GemPy Model

Returns

- **polydata** (*pv.core.PolyData*) – PyVista Mesh containing the geological map draped over the topography
- **cm** (*matplotlib.colors.ListedColormap*) – Colormap for plotting
- **rgb** (*bool*) – Boolean to use `rgb=True` when plotting

New in version 1.0.x.

`gemgis.visualization.get_petrel_cmap()` → matplotlib.colors.ListedColormap

Returning the Petrel cmap

Returns **cmap_seismic** – Seismic color map

Return type matplotlib.colors.ListedColormap

New in version 1.0.x.

`gemgis.visualization.get_points_along_spline(spline: pyvista.core.pointset.PolyData, dist: numpy.ndarray)`

Returning the closest point on the spline a given a length along a spline.

Parameters

- **spline** (*pv.core.pointset.PolyData*) – Spline with the resampled vertices
- **dist** (*np.ndarray*) – np.ndarray containing the measured depths (MD) of values along the well path

Returns **spline.points[idx_list]** – PyVista Array containing the selected points

Return type pv.core.pyvista_ndarray.pyvista_ndarray

New in version 1.0.x.

`gemgis.visualization.get_seismic_cmap()` → matplotlib.colors.ListedColormap

Returning the seismic cmap from https://github.com/lperozzi/Seismic_colormaps/blob/master/colormaps.py

Returns **cmap_seismic** – Seismic color map

Return type matplotlib.colors.ListedColormap

New in version 1.0.x.

`gemgis.visualization.group_borehole_dataframe(df: pandas.core.frame.DataFrame)` → List[pandas.core.frame.DataFrame]

Grouping Borehole DataFrame by Index

Parameters **df** (*pd.DataFrame*) – Pandas DataFrame containing the borehole data

Returns **df_groups**

Return type List[pd.DataFrame]

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pandas as pd
>>> df = pd.read_csv('file.csv')
```

```
>>> # Creating groups
>>> df_groups = gg.visualization.group_borehole_dataframe(df=df)
```

gemgis.visualization.**plane_through_hypocenters**(*spheres: pyvista.core.composite.MultiBlock*) → pyvista.core.pointset.PolyData

Fitting a plane through the hypocenters of earthquakes using Eigenvector analysis

Parameters **spheres** (*pv.core.composite.MultiBlock*) – PyVista MultiBlock object containing the hypocenters stored as spheres

Returns **plane** – Plane fitting through the hypocenters using Eigenvector analysis

Return type pv.core.pointset.PolyData

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import pyvista as pv
>>> spheres = pv.read(filename='spheres.vtk')
```

```
>>> # Fitting plane through spheres
>>> plane = gg.visualization.plane_through_hypocenters(spheres=spheres)
>>> plane
Header
PolyData    Information
N Cells     100
N Points    121
X Bounds    3.230e+07, 3.231e+07
Y Bounds    5.618e+06, 5.620e+06
Z Bounds    -1.113e+04, -8.471e+03
N Arrays    2
Data Arrays
Name                Field   Type    N Comp  Min         Max
Normals             Points  float32 3       0.000e+00   1.000e+00
TextureCoordinates  Points  float32 2       0.000e+00   1.000e+00
```

gemgis.visualization.**plot_data**(*geo_data*, *show_basemap: bool = False*, *show_geolmap: bool = False*, *show_topo: bool = False*, *show_interfaces: bool = False*, *show_orientations: bool = False*, *show_customsections: bool = False*, *show_wms: bool = False*, *show_legend: bool = True*, *show_hillshades: bool = False*, *show_slope: bool = False*, *show_aspect: bool = False*, *show_contours: bool = False*, *add_to_extent: float = 0*, *hide_topo_left: bool = False*, ***kwargs*)

Plotting Input Data

Parameters

- **geo_data** – GemPy Geo Data Class containing the raw data
- **show_basemap** (*bool*) – Showing the basemap. Options include `True` and `False`, default is `False`
- **show_geolmap** (*bool*) – Showing the geological map. Options include `True` and `False`, default is `False`
- **show_topo** (*bool*) – Showing the topography/digital elevation model. Options include `True` and `False`, default is `False`
- **show_interfaces** (*bool*) – Showing the interfaces. Options include `True` and `False`, default is `False`
- **show_orientations** (*bool*) – Showing orientations. Options include `True` and `False`, default is `False`
- **show_customsections** (*bool*) – Showing custom sections. Options include `True` and `False`, default is `False`
- **show_wms** (*bool*) – Showing a WMS layer. Options include `True` and `False`, default is `False`
- **show_legend** (*bool*) – Showing the legend of interfaces. Options include `True` and `False`, default is `False`
- **show_hillshades** (*bool*) – Showing hillshades. Options include `True` and `False`, default is `False`
- **show_slope** (*bool*) – Showing the slope of the DEM. Options include `True` and `False`, default is `False`
- **show_aspect** (*bool*) – Showing the aspect of the DEM. Options include `True` and `False`, default is `False`
- **show_contours** (*bool*) – Showing the contours of the DEM
- **add_to_extent** (*float*) – Number of meters to add to the extent of the plot in each direction, e.g. `add_to_extent=10`, default is `0`
- **hide_topo_left** (*bool*) – If set to `True`, the topography will not be shown in the left plot. Options include `True` and `False`, default is `False`
- **cmap_basemap** (*str*) – Cmap for basemap
- **cmap_geolmap** (*str*) – Cmap for geological map
- **cmap_topo** (*str*) – Cmap for topography
- **cmap_hillshades** (*str*) – Cmap for hillshades
- **cmap_slope** (*str*) – Cmap for slope
- **cmap_aspect** (*str*) – Cmap for aspect
- **cmap_interfaces** (*str*) – Cmap for interfaces
- **cmap_orientations** (*str*) – Cmap for orientations
- **cmap_wms** (*str*) – Cmap for WMS Service
- **cmap_contours** (*str*) – Cmap for contour lines

New in version 1.0.x.

gemgis.visualization.**plot_orientations**(*gdf: Union[geopandas.geodataframe.GeoDataFrame, pandas.core.frame.DataFrame]*, *show_planes: bool = True*, *show_density_contours: bool = True*, *show_density_contourf: bool = False*, *formation: str = None*, *method: str = 'exponential_kamb'*, *sigma: Union[float, int] = 1*, *cmap: str = 'Blues_r'*)

Plotting orientation values of a GeoDataFrame with mplstereonet

Parameters

- **gdf** (*Union[gpd.geodataframe.GeoDataFrame, pd.DataFrame]*) – (Geo-)DataFrame containing columns with orientations values
- **show_planes** (*bool*) – Variable to show planes of orientation values. Options include: `True` or `False`, default set to `True`
- **show_density_contours** (*bool*) – Variable to display density contours. Options include: `True` or `False`, default set to `True`
- **show_density_contourf** (*bool*) – Variable to display density contourf. Options include: `True` or `False`, default set to `False`
- **formation** (*str*) – Name of the formation for which the contourf plot is shown, e.g. `formation='Layer1'`, default is `None`
- **method** (*str*) – Method to estimate the orientation density distribution, `method='exponential_kamb'`, default is `'exponential_kamb'`
- **sigma** (*Union[float, int]*) – Expected count in standard deviations, e.g. `sigma=1`, default is 1
- **cmap** (*str*) – Name of the colormap for plotting the orientation densities, e.g. `cmap='Blues_r'`, default is `'Blues_r'`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and File
>>> import gemgis as gg
>>> import geopandas as gpd
>>> gdf = gpd.read_file(filename='file.shp')
>>> gdf
    id      formation   dip     azimuth geometry
0   None    Sand        25.00   310     POINT (49.249 1033.893)
1   None    Sand        30.00   315     POINT (355.212 947.557)
2   None    Sand        15.00   330     POINT (720.248 880.912)
3   None    Clay        10.00   135     POINT (526.370 611.300)
4   None    Clay        25.00   140     POINT (497.591 876.368)
5   None    Clay        35.00   50      POINT (394.593 481.039)
```

```
>>> # Creating plot
>>> gg.visualization.plot_orientations(gdf=gdf, show_planes=True, show_density_
↪contours=False, show_density_contourf=False)
```

gemgis.visualization.**polyline_from_points**(*points: numpy.ndarray*) → pyvista.core.pointset.PolyData

Creating PyVista PolyLine from points

Parameters **points** (*np.ndarray*) – Points defining the PolyLine

Returns **poly**

Return type pv.core.pointset.PolyData

New in version 1.0.x.

gemgis.visualization.**read_raster**(*path=<class 'str'>*, *nodata_val: typing.Union[float, int] = None*, *name: str = 'Elevation [m]'*) → pyvista.core.pointset.PolyData

Reading a raster and returning a mesh

Parameters

- **path** (*str*) – Path to the raster, e.g. `path='raster.tif'`
- **nodata_val** (*Union[float, int]*) – Nodata value of the raster, e.g. `nodata_val=9999.0`
- **name** (*str*) – Name of the data array, e.g. `name='Elevation [m]`, default is `'Elevation [m]'`

Returns **mesh** – PyVista mesh containing the raster values

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

Changed in version 1.1.1: Set nodata value manually if no data value is provided and raster does not contain nodata values

Example

```
>>> # Loading Libraries and outputting mesh
>>> import gemgis as gg
>>> polydata = gg.visualization.read_raster(path='raster.tif', nodata_val=9999.0,␣
↪name='Elevation [m]')
>>> polydata
Header
StructuredGrid  Information
N Cells         5595201
N Points        5600000
X Bounds        3.236e+07, 3.250e+07
Y Bounds        5.700e+06, 5.800e+06
Z Bounds        0.000e+00, 0.000e+00
Dimensions      2000, 2800, 1
N Arrays        1
Data Arrays
Name            Field   Type    N Comp  Min         Max
Elevation [m]   Points  float32 1       0.000e+00   5.038e+02
```

See also:

convert_to_rgb Converting bands to RGB values for plotting

drape_array_over_dem Draping an array of the Digital Elevation Model

gemgis.visualization.**resample_between_well_deviation_points**(*coordinates: numpy.ndarray*) → numpy.ndarray

Resampling between points that define the path of a well

Parameters **coordinates** (*np.ndarray*) – Nx3 Numpy array containing the X, Y, and Z coordinates that define the path of a well

Returns **points_resampled** – Resampled points along a well

Return type np.ndarray

New in version 1.0.x.

gemgis.visualization.**seismic_to_array**(*seismic_data*, *cdp_start: Optional[int] = None*, *cdp_end: Optional[int] = None*, *max_depth: Optional[Union[int, float]] = None*) → numpy.ndarray

Converting seismic data loaded with segysak to a NumPy array

Parameters

- **seismic_data** (*xarray.core.dataset.Dataset*) – seismic data loaded with the segysak package
- **cdp_start** (*Union[int, type(None)]*) – Value for the start CDP number, e.g. cdp_start=100, default is None
- **cdp_end** (*Union[int, type(None)]*) – Value for the end CDP number, e.g. cdp_start=200, default is None
- **max_depth** (*Union[int, float, type(None)]*) – Maximum depth of the seismic, e.g. max_depth=200, default is None

Returns **df_seismic_data_values_reshaped_selected** – NumPy array containing the seismic data

Return type np.ndarray

New in version 1.0.x.

gemgis.visualization.**seismic_to_mesh**(*seismic_data*, *cdp_start: Optional[int] = None*, *cdp_end: Optional[int] = None*, *max_depth: Union[int, float] = None*, *sampling_rate: Optional[int] = None*, *shift: Union[int, float] = 0*, *source_crs: Union[str, pyproj.crs.crs.CRS] = None*, *target_crs: Union[str, pyproj.crs.crs.CRS] = None*, *cdp_coords=None*) → pyvista.core.pointset.StructuredGrid

Converting seismic data loaded with segysak to a PyVista Mesh

Parameters

- **seismic_data** (*xarray.core.dataset.Dataset*) – seismic data loaded with the segysak package
- **cdp_start** (*Union[int, type(None)]*) – Value for the start CDP number, e.g. cdp_start=100, default is None
- **cdp_end** (*Union[int, type(None)]*) – Value for the end CDP number, e.g. cdp_start=200, default is None
- **max_depth** (*Union[int, float, type(None)]*) – Maximum depth of the seismic, e.g. max_depth=200, default is None
- **sampling_rate** (*Union[int, type(None)]*) – Sampling rate of the seismic, e.g. sampling_rate=2, default is None

- **shift** (*Union[int, float]*) – Shift of the seismic, e.g. `shift=100`, default is `0`
- **source_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Source CRS of the seismic, e.g. `source_crs='EPSG:25832'`, default is `None`
- **target_crs** (*Union[str, pyproj.crs.crs.CRS]*) – Target CRS of the seismic, e.g. `target_crs='EPSG:3034'`, default is `None`
- **cdp_coords** (*Union[int, float, type(None)]*) – CDP coordinates of the seismic if no CDP columns are present in the segysak object, default is `None`

Returns **grid** – PyVista Structured grid containing the seismic data

Return type pv.core.pointset.StructuredGrid

New in version 1.0.x.

gemgis.visualization.**show_well_log_along_well**(*coordinates: numpy.ndarray*, *dist: numpy.ndarray*, *values: numpy.ndarray*, *radius_factor: Union[int, float] = 2*) → pyvista.core.pointset.PolyData

Function to return a tube representing well log values along a well path

Parameters

- **coordinates** (*np.ndarray*) – Nx3 Numpy array containing the X, Y, and Z coordinates that define the path of a well
- **dist** (*np.ndarray*) – np.ndarray containing the measured depths (MD) of values along the well path
- **values** (*np.ndarray*) – np.ndarray containing the measured well log values along the well path
- **radius_factor** (*int, float*) – Radius factor to adjust the diameter of the tube, e.g. `radius_factor=2`, default is `2`

Returns **tube_along_spline** – PyVista PolyData Pointset representing the the measured well log values along the well path

Return type pyvista.core.pointset.PolyData

New in version 1.0.x.

9.1.9 gemgis.web module

Contributors: Alexander Jüstel, Arthur Endlein Correia, Florian Wellmann, Marius Pischke

GemGIS is a Python-based, open-source spatial data processing library. It is capable of preprocessing spatial data such as vector data raster data, data obtained from online services and many more data formats. GemGIS wraps and extends the functionality of packages known to the geo-community such as GeoPandas, Rasterio, OWSLib, Shapely, PyVista, Pandas, and NumPy.

GemGIS is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

GemGIS is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License (LICENSE) for more details.

gemgis.web.**create_request**(*wcs_url: str*, *version: str*, *identifier: str*, *form: str*, *extent: List[Union[int, float]]*, *name: str = 'test.tif'*) → str

Creating URL to request data from WCS Server

Parameters

- **wcs_url** (*str*) – Url of the WCS server, e.g. `url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'`
- **version** (*str*) – Version number of the WCS as string, e.g. `version='2.0.1'`
- **identifier** (*str*) – Name of the layer, e.g. `identifier='nw_dgm'`
- **form** (*str*) – Format of the layer, e.g. `form='image/tiff'`
- **extent** (*List[Union[float,int]]*) – Extent of the tile to be downloaded, size may be restricted by server, e.g. `extent=[0, 972, 0, 1069]`
- **name** (*str*) – Name of file, e.g. `name='tile1.tif'`, default is `'test.tif'`

Returns **url** – Url for the WCS request

Return type str

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

```
>>> # Creating Request for WCS Service
>>> url = gg.web.create_request(url=wcs.url, version=wcs.version, identifier='nw_dgm
↪', form='image/tiff', extent=[0, 1000, 0, 1000], name='test.tif'])
```

See also:

load_wcs Load WCS Service

load_as_file Download WCS data file

load_as_files Download WCS data files

gemgis.web.**load_as_array**(*url: str*, *layer: str*, *style: str*, *crs: Union[str, dict]*, *bbox: List[Union[int, float]]*, *size: List[int]*, *filetype: str*, *transparent: bool = True*, *save_image: bool = False*, *path: str = None*, *overwrite_file: bool = False*, *create_directory: bool = False*) → numpy.ndarray

Loading a portion of a WMS as array

Parameters

- **url** (*str*) – Link of the WMS Service, e.g. `url='https://ows.terrestris.de/osm/service?'`
- **layer** (*str*) – Name of layer to be requested, e.g. `layer='OSM-WMS'`
- **style** (*str*) – Name of style of the layer, e.g. `style='default'`
- **crs** (*str*) – String or dict containing the CRS, e.g. `crs='EPSG:4647'`
- **bbox** (*List[Union[float,int]]*) – List of bounding box coordinates, e.g. `bbox=[0, 972, 0, 1069]`

- **size** (*List[int]*) – List of x and y values defining the size of the image, e.g. `size=[1000, 1000]`
- **filetype** (*str*) – String of the image type to be downloaded, e.g. 'filetype='image/png'``
- **transparent** (*bool*) – Variable if layer is transparent. Options include: `True` or `False`, default set to `True`
- **save_image** (*bool*) – Variable to save image. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path and file name of the file to be saved, e.g. `path=map.tif`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **wms_array** – OWSlib map object loaded as np.ndarray

Return type np.ndarray

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WMS Service as array
>>> import gemgis as gg
>>> wms_map = gg.web.load_as_array(url='https://ows.terrestris.de/osm/service?',␣
↪layer='OSM-WMS', style='default', crs='EPSG:4647', bbox=[32286000,32328000,␣
↪5620000,5648000], size=[4200, 2800], filetype='image/png')
>>> wms_map
array([[[0.8039216 , 0.7647059 , 0.65882355],
[0.85882354, 0.8784314 , 0.6627451 ],
[0.87058824, 0.91764706, 0.6666667 ],
...,
[0.78431374, 0.7647059 , 0.65882355],
[0.8862745 , 0.9019608 , 0.81960785],
[0.9529412 , 0.93333334, 0.9019608 ]]], dtype=float32)
```

See also:

load_wms Load WMS Service

load_as_map Load Map from WMS Service

gemgis.web.**load_as_file**(*url: str*, *path: str*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Executing WCS request and downloading file into specified folder

Parameters

- **url** (*str*) – Url for request
- **path** (*str*) – Path where file is saved, e.g. `path='tile.tif'`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`

- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

```
>>> # Creating Request for WCS Service
>>> url = gg.web.create_request(url=wcs.url, version=wcs.version, identifier='nw_dgm
↪', form='image/tiff', extent=[0, 1000, 0, 1000], name='test.tif'])
```

```
>>> # Downloading file from WCS Service
>>> gg.web.load_as_file(url=url, path='tile.tif')
```

See also:

load_wcs Load WCS Service

create_request Create request for WCS

load_as_files Download WCS data files

gemgis.web.**load_as_files**(*wcs_url: str*, *version: str*, *identifier: str*, *form: str*, *extent: List[Union[int, float]]*, *size: int*, *path: str = ''*, *create_directory: bool = False*)

Executing WCS requests and downloading files into specified folder

Parameters

- **wcs_url** (*str*) – Url of the WCS server, e.g. `url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'`
- **version** (*str*) – Version number of the WCS as string, e.g. `version='2.0.1'`
- **identifier** (*str*) – Name of the layer, e.g. `identifier='nw_dgm'`
- **form** (*str*) – Format of the layer, e.g. `form='image/tiff'`
- **extent** (*List[Union[float,int]]*) – Extent of the tile to be downloaded, size may be restricted by server, e.g. `extent=[0, 972, 0, 1069]`
- **size** (*int*) – Size of the quadratic tile that is downloaded, e.g. `size=2000`
- **path** (*str*) – Path where the file is going to be downloaded, e.g. `name='tile1'`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

```
>>> # Downloading files from WCS Service
>>> gg.web.load_as_files(wcs_url=wcs.url, version=wcs.version, form='image/tiff',␣
↪extent=[0, 10000, 0, 10000], size=2000, path='tile.tif')
```

See also:

load_wcs Load WCS Service

create_request Create request for WCS

load_as_file Download WCS data file

gemgis.web.**load_as_gpd**(*url: str*, *typename: str = None*, *outputformat: str = None*) → geopandas.geodataframe.GeoDataFrame

Requesting data from a WFS Service

Parameters

- **url** (*str*) – URL of the Web Feature Service, e.g. `url="https://nibis.lbeg.de/net3/public/ogc.ashx?NodeId=476&Service=WFS&"`
- **typename** (*str*) – Name of the feature layer, e.g. `typename='iwan:L383'`, default is `None`
- **outputformat** (*str*) – Output format of the feature layer, e.g. `outputformat='xml/gml2'`, default is `None`

Returns **feature** – GeoDataFrame containing the feature data of the WFS Service

Return type gpd.geodataframe.GeoDataFrame

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WFS Service as GeoDataFrame
>>> import gemgis as gg
>>> wfs = gg.web.load_as_gpd(url="https://nibis.lbeg.de/net3/public/ogc.ashx?
↪NodeId=476&Service=WFS&")
>>> wfs
    gml_id      OBJECTID    ID  SURVEYNAME      ARCHIV  MESSJAHR        OPERATOR  ␣
↪                                    OP_NACHFOL                          MESSFIRMA␣
↪                                MESSPUNKTE  UP_DATE                     geometry
0       1541    1541        112 Jemgum 2007     0127494 2007        GdF Produktion␣
↪Exploration Deutschland GmbH Neptune Energy Deutschland GmbH Geophysik und␣
↪Geotechnik Leipzig GmbH   1340        2020-01-20T00:00:00+01:00   MULTIPOLYGON␣
↪(((32395246.839 5907777.660, 3239...
```

See also:

load_wfs Load WFS Service

gemgis.web.**load_as_map**(*url: str*, *layer: str*, *style: str*, *crs: Union[str, dict]*, *bbox: List[Union[int, float]]*, *size: List[int]*, *filetype: str*, *transparent: bool = True*, *save_image: bool = False*, *path: str = None*, *overwrite_file: bool = False*, *create_directory: bool = False*)

Loading a portion of a WMS as array

Parameters

- **url** (*str*) – Link of the WMS Service, e.g. `url='https://ows.terrestris.de/osm/service?'`
- **layer** (*str*) – Name of layer to be requested, e.g. `layer='OSM-WMS'`
- **style** (*str*) – Name of style of the layer, e.g. `style='default'`
- **crs** (*str*) – String or dict containing the CRS, e.g. `crs='EPSG:4647'`
- **bbox** (*List[Union[float,int]]*) – List of bounding box coordinates, e.g. `bbox=[0, 972, 0, 1069]`
- **size** (*List[int]*) – List of x and y values defining the size of the image, e.g. `size=[1000, 1000]`
- **filetype** (*str*) – String of the image type to be downloaded, e.g. `filetype='image/png'`
- **transparent** (*bool*) – Variable if layer is transparent. Options include: `True` or `False`, default set to `True`
- **save_image** (*bool*) – Variable to save image. Options include: `True` or `False`, default set to `False`
- **path** (*str*) – Path and file name of the file to be saved, e.g. `path=map.tif`
- **overwrite_file** (*bool*) – Variable to overwrite an already existing file. Options include: `True` or `False`, default set to `False`
- **create_directory** (*bool*) – Variable to create a new directory of directory does not exist Options include: `True` or `False`, default set to `False`

Returns **wms_map** – OWSlib map object

Return type owslib.util.ResponseWrapper

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WMS Service as Map
>>> import gemgis as gg
>>> wms_map = gg.web.load_as_map(url='https://ows.terrestris.de/osm/service?',␣
↪layer='OSM-WMS', style='default', crs='EPSG:4647', bbox=[32286000,32328000,␣
↪5620000,5648000], size=[4200, 2800], filetype='image/png')
>>> wms_map
<owslib.util.ResponseWrapper at 0x261d348cc10>
```

See also:

load_wms Load WMS Service

load_as_array Load Map as array from WMS Service

gemgis.web.**load_wcs**(*url: str*)

Loading Web Coverage Service

Parameters **url** (*str*) – Link of the Web Coverage Service, e.g. url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm'

Returns **wcs** – OWSLib Web Coverage Object

Return type owslib.coverage.wcs201.WebCoverageService_2_0_1

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WCS Service
>>> import gemgis as gg
>>> wcs = gg.web.load_wms(url='https://www.wcs.nrw.de/geobasis/wcs_nw_dgm')
>>> wcs
<owslib.coverage.wcs201.WebCoverageService_2_0_1 at 0x27fc64783d0>
```

See also:

create_request Create request for WCS

load_as_file Download WCS data file

load_as_files Download WCS data files

gemgis.web.**load_wfs**(*url: str*)

Loading a WFS Service by URL

Parameters **url** (*str*) – Link of the WFS Service, e.g. url="https://nibis.lbeg.de/net3/public/ogc.ashx?NodeId=476&Service=WFS&"

Returns **wfs** – OWSLib Feature object

Return type owslib.feature.wfs100.WebFeatureService_1_0_0

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WFS Service
>>> import gemgis as gg
>>> wfs = gg.web.load_wfs(url="https://nibis.lbeg.de/net3/public/ogc.ashx?
↪NodeId=476&Service=WFS&")
>>> wfs
<owslib.feature.wfs100.WebFeatureService_1_0_0 at 0x19260e21340>
```

See also:

load_as_gpd Load information of a WFS Service as GeoDataFrame

gemgis.web.**load_wms**(*url: str*)

Loading a WMS Service by URL

Parameters **url** (*str*) – Link of the WMS Service, e.g. url='https://ows.terrestris.de/osm/service?'

Returns **wms** – OWSLib WebMapService Object

Return type owslib.map.wms111.WebMapService

New in version 1.0.x.

Example

```
>>> # Loading Libraries and WMS Service
>>> import gemgis as gg
>>> wms = gg.web.load_wms(url='https://ows.terrestris.de/osm/service?')
>>> wms
<owslib.map.wms111.WebMapService_1_1_1 at 0x1c434eb6370>
```

See also:

load_as_map Load Map from WMS Service

load_as_array Load Map as array from WMS Service

9.1.10 Module contents

Top-level package for GemGIS.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

g

INDEX

D

E

G

I

L

M

O

P

R

S

T

U