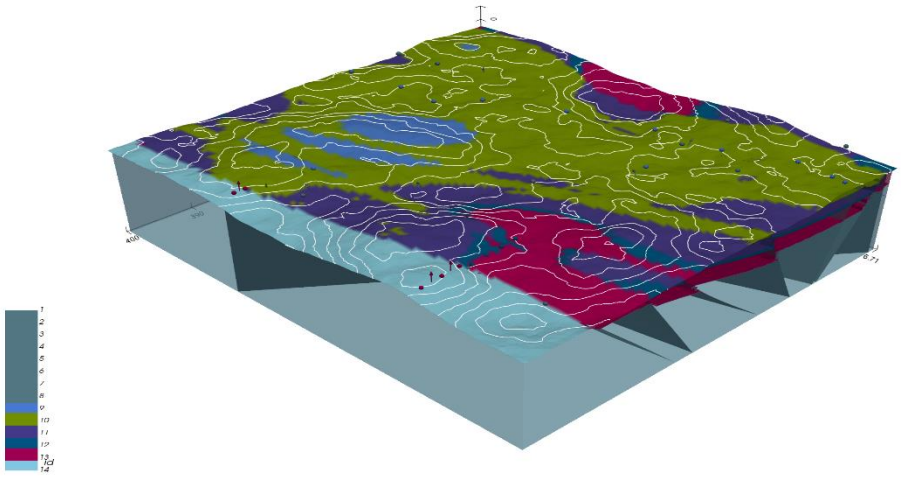
TRƯỜNG ĐẠI HỌC MỎ-ĐỊA CHẤT

KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO HỌC THUẬT

NGHIÊN CỨU MÔ HÌNH CẤU TRÚC ĐỊA CHẤT 3D- GEMPY

PHẠM AN CƯƠNG



Hà Nội, 6/2023

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN VỀ GEMPY

1.1 Kiến thức cơ bản về mô hình địa chất với GemPy

Importing GemPy

```
import gempy as gp

# Importing auxiliary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import os

# Setting options
np.random.seed(1515)
pd.set_option('precision', 2)
```

1.2 Nhập và tạo dữ liệu đầu vào

Dữ liệu được sử dụng để xây dựng mô hình trong GemPy được lưu trữ trong các đối tượng Python. Các lớp dữ liệu chính là:

- Surface_points
- Orientations
- Grid
- Surfaces
- Series
- Additional data
- Faults

Hầu hết dữ liệu cũng có thể được tạo từ dữ liệu thô ở dạng tệp CSV (CSV = các giá trị được phân tách bằng dấu phẩy). Các tệp như vậy có thể đạt được bằng cách xuất dữ liệu mô hình từ một chương trình khác như GeoModeller hoặc chỉ cần tạo nó trong phần mềm bảng tính như Microsoft Excel hoặc LibreOffice Calc.

Trong hướng dẫn này, tất cả dữ liệu đầu vào được tạo bằng cách nhập các tệp CSV như vậy. Các tệp mẫu này có thể được tìm thấy trong thư mục input_data trong thư mục gốc của GemPy. Dữ liệu bao gồm các giá trị vị trí xx-, yy- và zz cho tất cả các điểm bề mặt và phép đo định hướng. Đối với loại thứ hai, các cực, phương vị và cực được bao gồm thêm. Các điểm bề mặt cũng được chỉ định một đội hình. Đây có thể là một đơn vị thạch học như "Sa thạch" hoặc một đặc điểm cấu trúc như "Đứt gãy chính". Điều quan trọng cần nhớ là, trong GemPy, các điểm vị trí giao diện đánh dấu phần dưới cùng của một lớp. Nếu những điểm như vậy là cần thiết để giống với đỉnh của hệ thống (ví dụ: khi lập mô hình một sự xâm nhập), thì điều này có thể đạt được bằng cách xác định phép đo hướng đảo ngược tương ứng.

Khi tạo Dữ liệu từ tệp CSV, cũng phải xác định phạm vi thực của mô hình theo xx, yy và zz, cũng như khai báo độ phân giải mong muốn cho mỗi trục. Độ phân giải này sẽ lần lượt xác định số lượng voxels được sử dụng trong quá trình mô hình hóa. Ở đây, chúng tôi dựa trên độ phân giải trung bình là 50x50x50, lên tới 125.000 voxels. Phạm vi mô hình nên được chọn theo cách chứa tất cả dữ liệu liên quan trong một không gian đại diện. Vì voxels mô hình của chúng tôi không phải là hình khối mà là lăng kính, độ phân giải có thể có hình dạng khác với mức độ. Chúng tôi khuyên bạn không nên sử dụng nhiều hơn 100 ô theo mọi hướng (1.000.000 voxels), vì độ phân giải cao hơn sẽ ngày càng trở nên đắt đỏ hơn để tính toán.

```
geo_model = gp.create_model('Tutorial_ch1_1_Basics')
```

```
data_path = 'https://raw.githubusercontent.com/cgre-aachen/gempy_data/master/'
# Importing the data from CSV-files and setting extent and resolution
gp.init_data(geo_model, [0, 2000., 0, 2000., 0, 750.], [50, 50, 50],
             path_o=data_path + "/data/input_data/getting_started/"
                                "simple_fault_model_orientations.csv",
             path_i=data_path + "/data/input_data/getting_started/"
                                "simple_fault_model_points.csv",
             default_values=True)
```

Out:

```
Active grids: ['regular']

Tutorial_ch1_1_Basics  2021-04-18 11:28
```

```
geo_model.surfaces
```

	surface	series	order_surfaces	color	id
0	Shale	Default series	1	#015482	1
1	Sandstone_1	Default series	2	#9f0052	2
2	Siltstone	Default series	3	#ffbe00	3
3	Sandstone_2	Default series	4	#728f02	4
4	Main_Fault	Default series	5	#443988	5
5	basement	Basement	1	#ff3f20	6

Dữ liệu đầu vào sau đó có thể được liệt kê bằng cách sử dụng lệnh get_data.

```
gp.get_data(geo_model, 'surface_points').head()
```

	X	Y	Z	smooth	surface
0	1000	50	450.00	2.00e-06	Shale
1	1000	150	433.33	2.00e-06	Shale
2	1000	300	433.33	2.00e-06	Shale
3	1000	500	466.67	2.00e-06	Shale
4	1000	1000	533.33	2.00e-06	Shale

```
gp.get_data(geo_model, 'orientations').head()
```

	X	Y	Z	G_x	G_y	G_z	smooth	surface
0	1000	1000	300	0.32	1.00e-12	0.95	0.01	Shale
1	400	1000	420	0.32	1.00e-12	0.95	0.01	Sandstone_2
2	500	1000	300	-0.95	1.00e-12	0.32	0.01	Main_Fault

Khai báo trình tự tuần tự của các thành tạo địa chất

```
geo_model.surfaces
```

	surface	series	order_surfaces	color	id
0	Shale	Default series	1	#015482	1
1	Sandstone_1	Default series	2	#9f0052	2
2	Siltstone	Default series	3	#ffbe00	3
3	Sandstone_2	Default series	4	#728f02	4
4	Main_Fault	Default series	5	#443988	5
5	basement	Basement	1	#ff3f20	6

```
gp.map_stack_to_surfaces(geo_model,
                         {"Fault_Series": 'Main_Fault',
                          "Strat_Series": ('Sandstone_2', 'Siltstone',
                                           'Shale', 'Sandstone_1', 'basement')},
                         remove_unused_series=True)
```

	surface	series	order_surfaces	color	id
4	Main_Fault	Fault_Series	1	#443988	1
0	Shale	Strat_Series	1	#015482	2
1	Sandstone_1	Strat_Series	2	#9f0052	3
2	Siltstone	Strat_Series	3	#ffbe00	4
3	Sandstone_2	Strat_Series	4	#728f02	5
5	basement	Strat_Series	5	#ff3f20	6

```
geo_model.surfaces
```

	surface	series	order_surfaces	color	id
4	Main_Fault	Fault_Series	1	#443988	1
0	Shale	Strat_Series	1	#015482	2
1	Sandstone_1	Strat_Series	2	#9f0052	3
2	Siltstone	Strat_Series	3	#ffbe00	4
3	Sandstone_2	Strat_Series	4	#728f02	5
5	basement	Strat_Series	5	#ff3f20	6

```
geo_model.stack
```

	order_series	BottomRelation	isActive	isFault	isFinite
Fault_Series	1	Erosion	True	False	False
Strat_Series	2	Erosion	True	False	False

```
geo_model.set_is_fault(['Fault_Series'])
```

Out:

```
Fault colors changed. If you do not like this behavior, set change_color to False.
```

	order_series	BottomRelation	isActive	isFault	isFinite
Fault_Series	1	Fault	True	True	False
Strat_Series	2	Erosion	True	False	False

```
geo_model.faults.faults_relations_df
```

	Fault_Series	Strat_Series
Fault_Series	False	True
Strat_Series	False	False

```
geo_model.faults
```

	order_series	BottomRelation	isActive	isFault	isFinite
Fault_Series	1	Fault	True	True	False
Strat_Series	2	Erosion	True	False	False

```
geo_model.faults.faults_relations_df
```

	Fault_Series	Strat_Series
Fault_Series	False	True
Strat_Series	False	False

1.3 Trả về thông tin từ dữ liệu đầu vào

Dữ liệu đầu vào mô hình ở đây có tên là "geo_model", chứa tất cả thông tin cần thiết cho việc xây dựng mô hình. Có thể truy cập các loại thông tin khác nhau bằng cách sử dụng gp.get_data hoặc đơn giản bằng cách truy cập vào atrribues.

```
geo_model.grid
```

Out:

```
Grid Object. Values:
array([[  20. ,   20. ,    7.5],
       [  20. ,   20. ,   22.5],
       [  20. ,   20. ,   37.5],
       ...,
       [1980. , 1980. ,  712.5],
       [1980. , 1980. ,  727.5],
       [1980. , 1980. ,  742.5]])
```

Thuật toán cốt lõi của GemPy dựa trên phép nội suy của hai loại dữ liệu: - điểm_mặt_mặt và - phép đo định hướng (nếu muốn biết thêm về cách hoạt động của thuật toán nội suy này, hãy xem bài báo: https://www.geosci-model-dev.net/12/1/2019/gmd-12-1-2019.pdf).

Chúng tôi đã giới thiệu hàm get \ _data ở trên. Bạn cũng có thể chỉ định loại dữ liệu bạn muốn gọi, bằng cách khai báo thuộc tính chuỗi "dtype" là 'surface_points' (giao diện) hoặc 'định hướng'.

1.4 Khung dữ liệu giao diện (Interfaces Dataframe)

```
gp.get_data(geo_model, 'surface_points').head()
```

	X	Y	Z	smooth	surface
52	700	1000	300.0	2.00e-06	Main_Fault
53	600	1000	200.0	2.00e-06	Main_Fault
54	500	1000	100.0	2.00e-06	Main_Fault
55	1000	1000	600.0	2.00e-06	Main_Fault
56	1100	1000	700.0	2.00e-06	Main_Fault

1.5 Khung dữ liệu định hướng (Orientations Dataframe)

```
gp.get_data(geo_model, 'orientations')
```

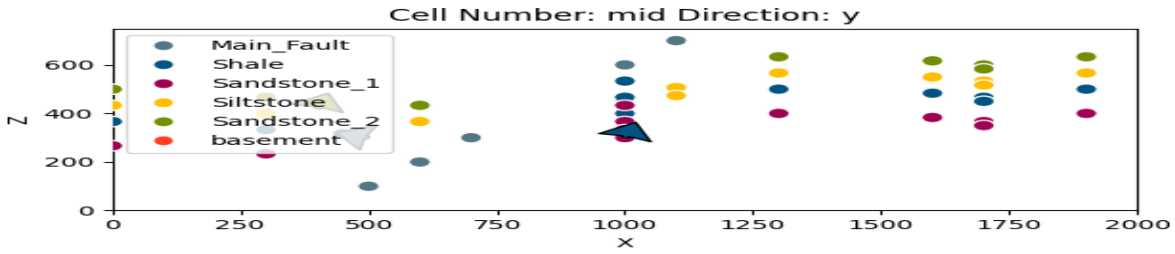
	X	Y	Z	G_x	G_y	G_z	smooth	surface
2	500	1000	300	-0.95	1.00e-12	0.32	0.01	Main_Fault
0	1000	1000	300	0.32	1.00e-12	0.95	0.01	Shale
1	400	1000	420	0.32	1.00e-12	0.95	0.01	Sandstone_2

Lưu ý rằng bây giờ tất cả các bề mặt đã được gán cho một chuỗi và được hiển thị theo đúng thứ tự (từ trẻ đến già).

1.6 Trực quan hóa dữ liệu đầu vào

Sử dụng hàm plot_data, để tạo hình chiếu 2D của các điểm dữ liệu lên một mặt phẳng có hướng đã chọn (có thể chọn thuộc tính này là xx, yy hoặc zz).

```
plot = gp.plot_2d(geo_model, show_lith=False, show_boundaries=False)
plt.show()
```

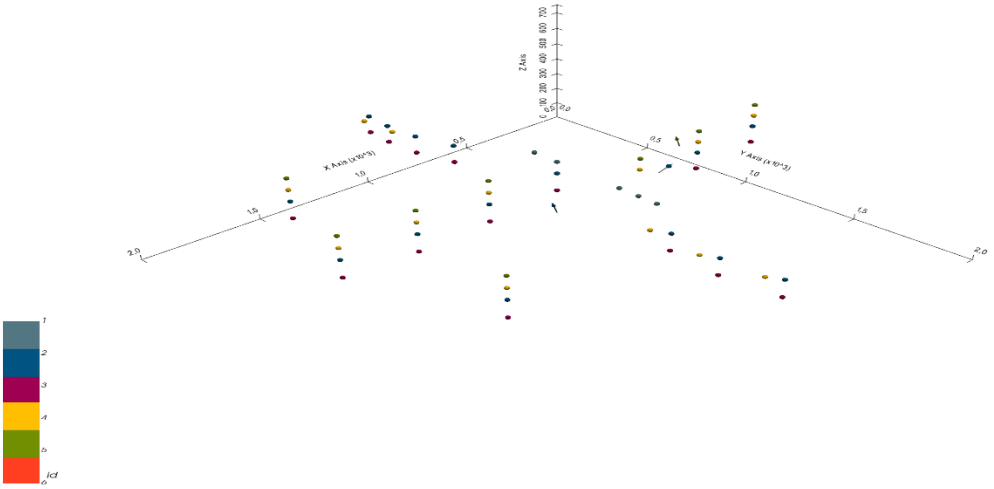


Sử dụng plot_data_3D, cũng có thể trực quan hóa dữ liệu này dưới dạng 3D. Lưu ý rằng trực quan hóa 3D trong GemPy yêu cầu phải cài đặt Bộ công cụ trực quan hóa (VTK).

Tất cả các ô 3D trong GemPy đều có tính tương tác. Điều này có nghĩa là có thể kéo và thả bất kỳ điểm dữ liệu và phép đo nào. Các hình chiếu trục vuông góc trong VTK đặc biệt hữu ích để di chuyển các điểm chỉ trên một mặt phẳng 2D mong muốn. Mọi thay đổi sau đó sẽ được lưu trữ vĩnh viễn trong khung dữ liệu "InputData". Nếu muốn đặt lại các điểm dữ liệu sẽ cần tải lại dữ liệu đầu vào ban đầu.

Việc thực thi ô bên dưới sẽ mở ra một cửa sổ mới với biểu đồ tương tác 3D của dữ liệu.

```
gpv = gp.plot_3d (geo_model, image = False, plotter_type = 'basic')
```



1.7 Tạo mô hình-Model generation

Khi chắc chắn rằng đã xác định tất cả thông tin chính như mong muốn trong đối tượng DataManagement.InputData (được đặt tên là geo_data trong các hướng dẫn), có thể tiếp tục với bước tiếp theo để tạo mô hình địa chất: chuẩn bị dữ liệu đầu vào cho phép nội suy.

Điều này được thực hiện bằng cách tạo một đối tượng InterpolatorData (có tên là interp_data trong các hướng dẫn) từ đối tượng InputData thông qua hàm sau::

```
gp.set_interpolator(geo_model,
                    compile_theano=True,
                    theano_optimizer='fast_compile',
                    )
```

Out:

```
Setting kriging parameters to their default values.
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                  values
range              2926.17
$C_o$            203869.05
drift equations     [3, 3]

<gempy.core.interpolator.InterpolatorModel object at 0x7fcb8ab2c5e0>
```

Hàm này thay đổi tỷ lệ phạm vi và tọa độ của dữ liệu gốc (và lưu trữ nó trong thuộc tính geo_data_res hoạt động như một đối tượng InputData thông thường) và thêm các tham số toán học cần thiết để tiến hành nội suy. Việc tính toán bước này có thể mất một lúc, vì nó cũng biên dịch một hàm theano cần thiết cho tính toán mô hình. Tuy nhiên, nếu điều này không cần thiết, chúng ta có thể bỏ qua nó bằng cách khai báo compile_theano = False trong hàm.

Hơn nữa, quá trình chuẩn bị này bao gồm việc ấn định số lượng cho mỗi hệ tầng. Lưu ý rằng GemPy's luôn tạo một hệ tầng cơ bản mặc định là số hệ tầng cuối cùng. Sau đó, các số được phân bổ từ trẻ nhất đến già nhất được xác định bởi trình tự của chuỗi và hệ tầng. Trên các dạng thuộc tính trên dữ liệu nội suy của có thể tìm ra số nào đã được chỉ định cho dạng nào:
Các tham số được sử dụng cho phép nội suy có thể được trả về bằng cách sử dụng hàm get_kriging_parameters. Chúng được tạo tự động từ dữ liệu gốc, nhưng có thể được thay đổi nếu cần. Tuy nhiên, người dùng nên cẩn thận làm như vậy, nếu họ không hiểu hết ý nghĩa của chúng.

```
gp.get_data(geo_model, 'kriging')
```

	values
range	2926.17
C_o	203869.05
drift equations	[3, 3]

Để tính toán mô hình đầy đủ thông qua compute_model. Theo mặc định, sẽ trả về hai giải pháp riêng biệt ở dạng mảng. Đầu tiên cung cấp thông tin về các thành tạo thạch học, thứ hai về mạng lưới đứt gãy trong mô hình. Các mảng này bao gồm hai mảng con là các mục nhập mỗi mảng:
Giải pháp mô hình khối thạch học:
Entry [0]: Mảng này cho biết loại hình thành thạch học nào được tìm thấy trong mỗi voxel, như được chỉ ra bởi một số hình thành tương ứng.
Entry [1]: Mảng trường tiềm năng thể hiện định hướng của các đơn vị và lớp thạch học trong mô hình khối.
Giải pháp mô hình khối mạng lưới đứt gãy:
Entry [0]: Mảng trong đó tất cả các vùng được phân tách bằng đứt gãy của mô hình được biểu diễn bằng một số riêng biệt có trong mỗi voxel.
Entry [1]: Mảng trường tiềm năng liên quan đến mạng lưới đứt gãy trong mô hình khối.
Dưới đây, là minh họa các giải pháp mô hình khác nhau này và cách chúng có thể được sử dụng.

```
sol = gp.compute_model(geo_model)
sol
```

Out:

```
Lithology ids
  [6. 6. 6. ... 2. 2. 2.]
```

```
geo_model.solutions
```

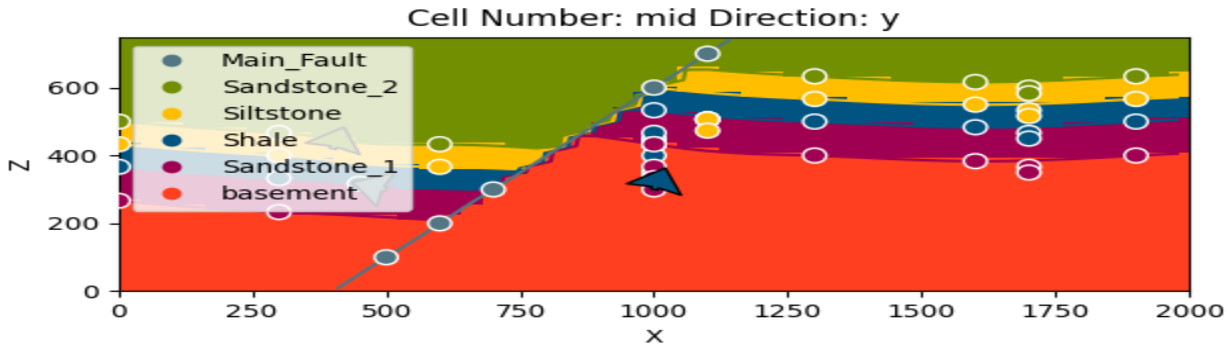
Out:

```
Lithology ids
  [6. 6. 6. ... 2. 2. 2.]
```

1.8 Trực quan hóa mô hình trong GemPy

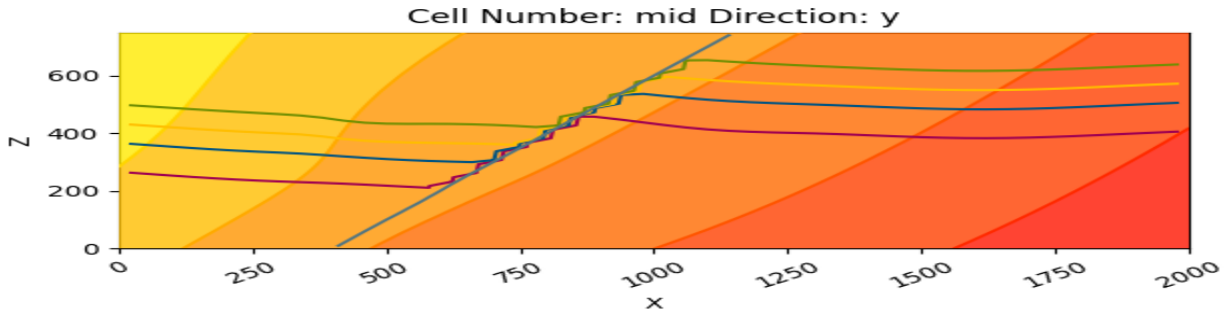
Các giải pháp mô hình có thể dễ dàng được hình dung trực tiếp trong các phần 2D trong GemPy. Hãy xem xét khối thạch học trong mô hình:

```
gp.plot_2d(geo_model, show_data=True)
plt.show()
```

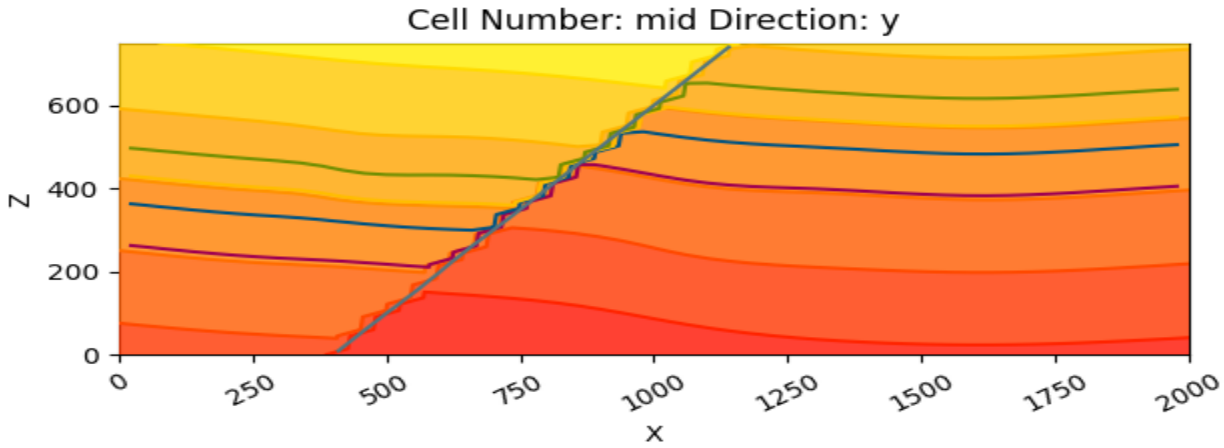


Với cell_number = 25 và nhớ rằng đã xác định độ phân giải là 50 ô theo mỗi hướng, chọn một phần đi qua giữa khối. Di chuyển 25 ô theo hướng = 'y', do đó, biểu đồ mô tả một mặt phẳng song song với các trục xx- và yy. Đặt plot_data = True, có thể vẽ dữ liệu gốc cùng với kết quả. Thay đổi các giá trị cho cell_number và hướng, có thể di chuyển qua mô hình khối 3D và khám phá nó bằng cách nhìn vào các mặt phẳng 2D khác nhau.
Có thể làm điều tương tự với giải pháp trường vô hướng thạch học:

```
gp.plot_2d(geo_model, show_data=False, show_scalar=True, show_lith=False)
plt.show()
```




```
gp.plot_2d(geo_model, series_n=1, show_data=False, show_scalar=True, show_lith=False)
plt.show()
```



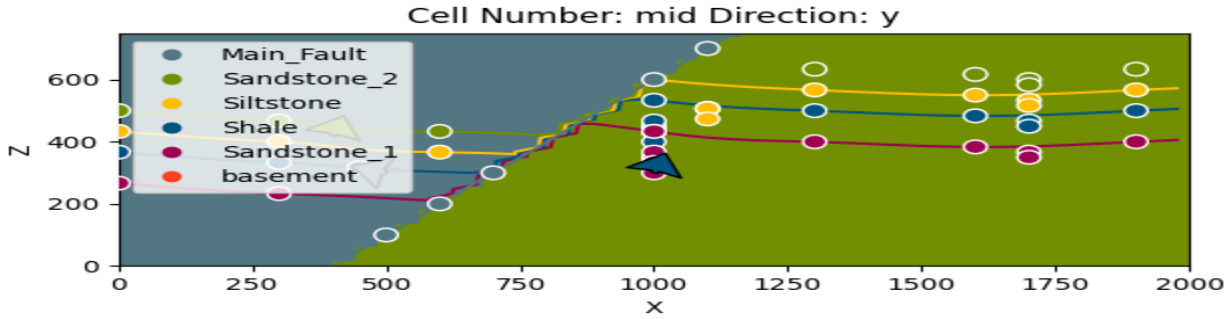
Điều này minh họa rõ ràng sự biến dạng liên quan đến nếp uốn của địa tầng, cũng như cách các lớp chịu ảnh hưởng của đứt gãy.
Các giải pháp mô hình hóa mạng lưới đứt gãy có thể được hình dung theo cùng một cách:

```
geo_model.solutions.scalar_field_at_surface_points
```

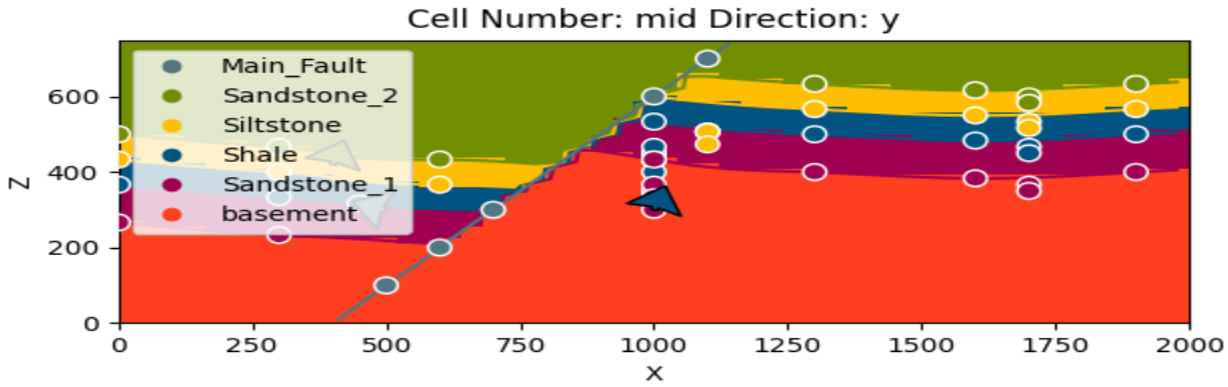
Out:

```
array([[0.03075848, 0.        , 0.        , 0.        , 0.        ],
       [0.        , 0.77174354, 0.72471042, 0.80357372, 0.83598092]])
```

```
gp.plot_2d(geo_model, show_block=True, show_lith=False)
plt.show()
```



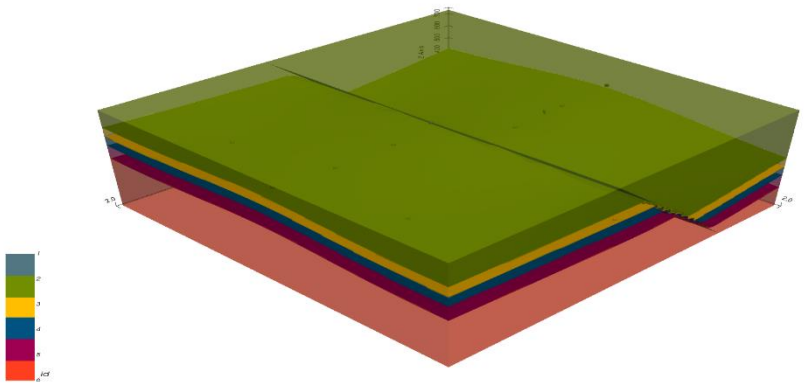

```
gp.plot_2d(geo_model, series_n=1, show_block=True, show_lith=False)
plt.show()
```



1.9 Biểu diễn hình khối và trực quan hóa vtk

Ngoài các phần 2D, có thể trích xuất các bề mặt để hình dung trong trình kết xuất 3D. Các bề mặt có thể được hình dung dưới dạng phức hợp tam giác 3D trong VTK (xem hàm plot_surfaces_3D). Để tạo ra những hình tam giác này, cần trích xuất các đỉnh và đỉnh tương ứng từ các trường thạch học và đứt gãy tiềm ẩn. Quá trình này được tự động hóa trong GemPy với hàm get_surface.

```
ver, sim = gp.get_surfaces(geo_model)
gpv = gp.plot_3d(geo_model, image=False, plotter_type='basic')
```



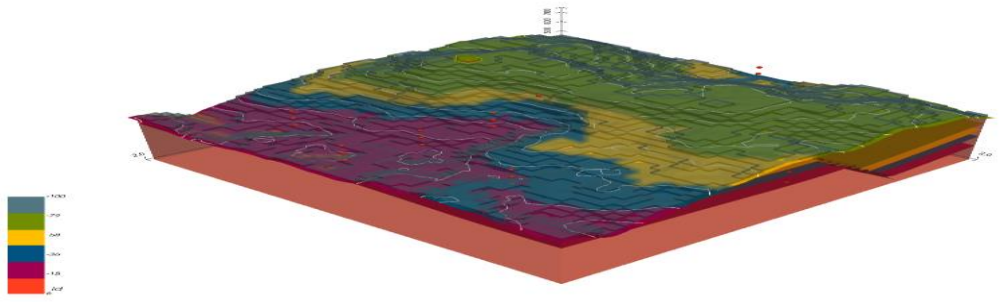
Sử dụng dữ liệu nội suy đã thay đổi tỷ lệ, cũng có thể chạy trực quan hóa VTK 3D trong một chế độ tương tác cho phép thay đổi và cập nhật mô hình trong thời gian thực. Tương tự như hình ảnh 3D tương tác của dữ liệu đầu vào, các thay đổi được lưu vĩnh viễn (trong đối tượng InterpolationInput.dataframe). Ngoài ra, những thay đổi kết quả trong các mô hình địa chất được tính toán lại theo thời gian thực.

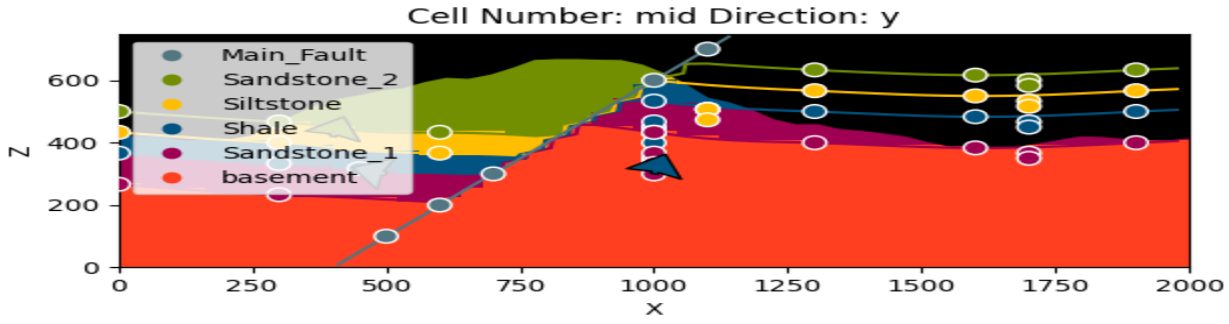
1.10 Thêm địa hình-Adding topography

```
geo_model.set_topography(d_z=(350, 750))
```

Out:

```
Active grids: ['regular' 'topography']

Grid Object. Values:
array([[  20.        ,   20.        ,    7.5       ],
       [  20.        ,   20.        ,   22.5       ],
       [  20.        ,   20.        ,   37.5       ],
       ...,
       [2000.        , 1918.36734694,  423.48951452],
       [2000.        , 1959.18367347,  430.25455308],
       [2000.        , 2000.        ,  431.07163663]])
gp.compute_model(geo_model)
gp.plot_2d(geo_model, show_topography=True)
plt.show()


# sphinx_gallery_thumbnail_number = 9
gpv = gp.plot_3d(geo_model, plotter_type='basic', show_topography=True,
show_surfaces=True,
                 show_lith=True,
                 image=False)
```





Out:

```
/WorkSSD/PythonProjects/gempy/gempy/core/solution.py:173: VisibleDeprecationWarning:
Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or
ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  self.geological_map = np.array(
```

1.11 Tính toán tại một vị trí nhất định

Điều này được thực hiện bằng cách sửa đổi lưới thành lưới tùy chỉnh và tính toán lại. Lưu ý rằng các kết quả được đưa ra dưới dạng: *grid + surfaces_points_ref + surface_points_rest locations*

```
x_i = np.array([[3, 5, 6]])
sol = gp.compute_model(geo_model, at=x_i)
```

Out:

```
Active grids: ['custom']
/WorkSSD/PythonProjects/gempy/gempy/core/solution.py:168: VisibleDeprecationWarning:
Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or
ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  self.custom = np.array(
```

Therefore if we just want the value at **x_i**:

```
sol.custom
```

Out:

```
array([array([[6.]]), array([[0.18630133],
                             [0.63163565]])], dtype=object)
```

This return the id, and the scalar field values for each series

1.12 Lưu mô hình

GemPy sử dụng Python [pickle] để lưu trữ nhanh các đối tượng tạm thời (https://docs.python.org/3/library/pickle.html). Tuy nhiên, cần có tính nhất quán của phiên bản mô-đun. Để tải một pickle vào GemPy, bạn phải đảm bảo rằng bạn đang sử dụng cùng một phiên bản pickle và các mô-đun phụ thuộc (ví dụ: Pandas, NumPy) như đã được sử dụng khi dữ liệu được lưu trữ ban đầu.

Để lưu trữ lâu dài an toàn hơn, có thể xuất pandas.DataFrames sang csv bằng cách sử dụng:

```
gp.save_model(geo_model)
```

Out:

```
True
```

1.13 Thêm địa hình vào mô hình địa chất

```
import gempy as gp
import numpy as np
import matplotlib.pyplot as plt
import os
```

1.14 Quy trình phổ biến để thiết lập một mô hình

```
data_path = 'https://raw.githubusercontent.com/cgre-aachen/gempy_data/master/'

geo_model = gp.create_model('Single_layer_topo')
gp.init_data(geo_model, extent=[450000, 460000, 70000, 80000, -1000, 500],
             resolution=[50, 50, 50],
             path_i=data_path + "/data/input_data/tut-ch1-7/onelayer_interfaces.csv",
             path_o=data_path + "/data/input_data/tut-ch1-7/onelayer_orient.csv")
```

Out:

```
Active grids: ['regular']

Single_layer_topo  2021-04-18 11:28
```

use happy spring colors!

```
geo_model.surfaces.colors.change_colors({'layer1': '#ff8000', 'basement': '#88cc60'})
gp.map_stack_to_surfaces(geo_model, {'series': ('layer1', 'basement')})
```

	surface	series	order_surfaces	color	id
0	layer1	series	1	#ff8000	1
1	basement	series	2	#88cc60	2

```
s = {'s1': ([450000, 75000], [460000, 75500], [100, 100])}
geo_model.set_section_grid(s)
```

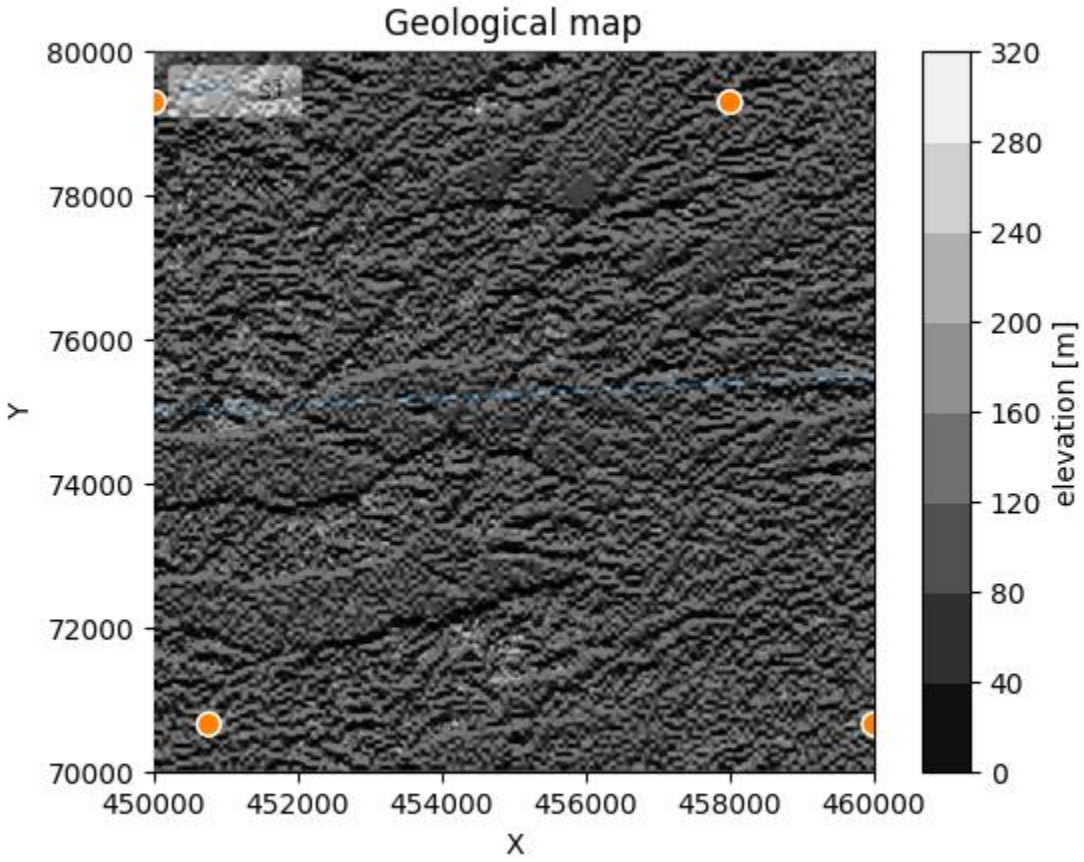
Out:

```
Active grids: ['regular' 'sections']
```

	start	stop	resolution	dist
s1	[450000, 75000]	[460000, 75500]	[100, 100]	10012.49

CHƯƠNG 2: TẠO ĐỊA HÌNH-ADDING TOPOGRAPHY
2.1 Tải từ file raster

```
cwd = os.getcwd()
if not 'examples' in cwd:
    path_dir = os.getcwd() + '/examples/tutorials/ch5_probabilistic_modeling'
else:
    path_dir = cwd

fp = path_dir + "/../../data/input_data/tut-ch1-7/bogota.tif"
geo_model.set_topography(source='gdal', filepath=fp)
gp.plot_2d(geo_model,          show_topography=True,          section_names=['topography'],
show_lith=False,
           show_boundaries=False,
           kwargs_topography={'cmap': 'gray', 'norm': None}
           )
plt.show()
```



Out:

```
Cropped raster to geo_model.grid.extent.
depending on the size of the raster, this can take a while...
```

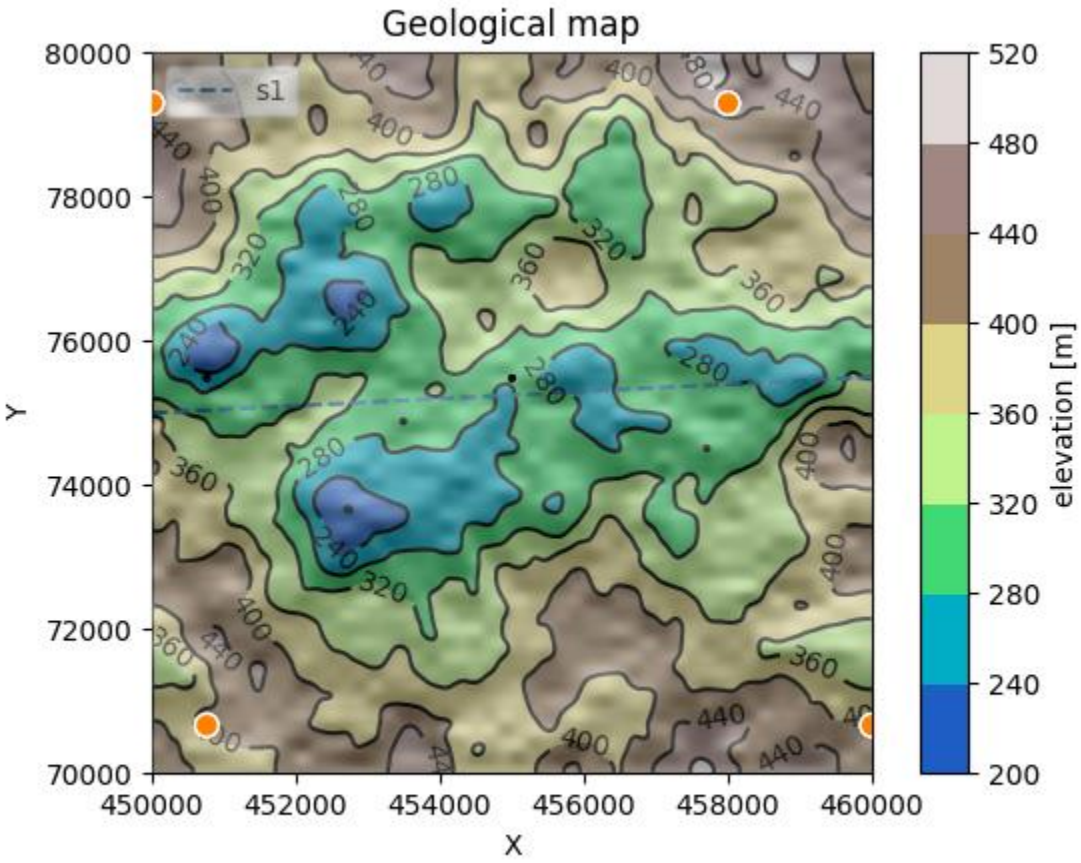
```
storing converted file...
Active grids: ['regular' 'topography' 'sections']
```

2.2 Tạo địa hình

Sử dụng hàm gempys set_topography để tạo địa hình ngẫu nhiên dựa trên lưới Fractal:

sphinx_gallery_thumbnail_number = 2

```
geo_model.set_topography(source='random')
gp.plot_2d(geo_model, show_topography=True, section_names=['topography'])
plt.show()
```



Out:

```
[200. 500.]
Active grids: ['regular' 'topography' 'sections']
```

Các từ khóa bổ sung:

• fd: kích thước fractal:

mặc định là 2.0. Càng cao (thử 2.9), cảnh quan sẽ càng thô.

• d_z: chênh lệch độ cao:

Nếu không, hãy kéo dài 20% mô hình theo hướng z.

• Mức độ:

mức độ theo hướng xy. Nếu không, geo_model.grid.extent sẽ được sử dụng.

•Độ phân giải:

độ phân giải của mảng địa hình. Nếu không, geo_model.grid.resoution sẽ được sử dụng. Tăng độ phân giải dẫn đến các bản đồ địa chất đẹp hơn.

```
geo_model.set_topography(source='random', fd=1.9, d_z=np.array([0, 250]),
                         resolution=np.array([200, 200]))
```

Out:

```
Active grids: ['regular' 'topography' 'sections']

Grid Object. Values:
array([[450100.        ,  70100.        ,   -985.        ],
       [450100.        ,  70100.        ,   -955.        ],
       [450100.        ,  70100.        ,   -925.        ],
       ...,
       [460000.        ,  75500.        ,    469.6969697 ],
       [460000.        ,  75500.        ,    484.84848485],
       [460000.        ,  75500.        ,    500.        ]])
```

Lưu ý rằng mỗi khi hàm này được gọi, một địa hình ngẫu nhiên mới sẽ được tạo ra. Nếu bạn đặc biệt thích địa hình đã tạo hoặc nếu bạn đã tải một tệp lớn bằng gdal, bạn có thể lưu đối tượng địa hình và tải lại sau:

save:

```
geo_model._grid.topography.save('test_topo')
```

load:

```
geo_model.set_topography(source='saved', filepath='test_topo.npy')
```

Out:

```
Active grids: ['regular' 'topography' 'sections']

Grid Object. Values:
array([[450100.        ,  70100.        ,   -985.        ],
       [450100.        ,  70100.        ,   -955.        ],
       [450100.        ,  70100.        ,   -925.        ],
       ...,
       [460000.        ,  75500.        ,    469.6969697 ],
       [460000.        ,  75500.        ,    484.84848485],
       [460000.        ,  75500.        ,    500.        ]])
```

2.3 Mô hình máy tính

```
gp.set_interpolator(geo_model)
```

Out:

```
Setting kriging parameters to their default values.
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  0
Compilation Done!
Kriging values:
                values
range         14221.46
```

```
$C_o$           4815476.19
drift equations         [3]


<gempy.core.interpolator.InterpolatorModel object at 0x7fcb8ae537f0>
```

```
gp.compute_model(geo_model, compute_mesh=False, set_solutions=True)
```

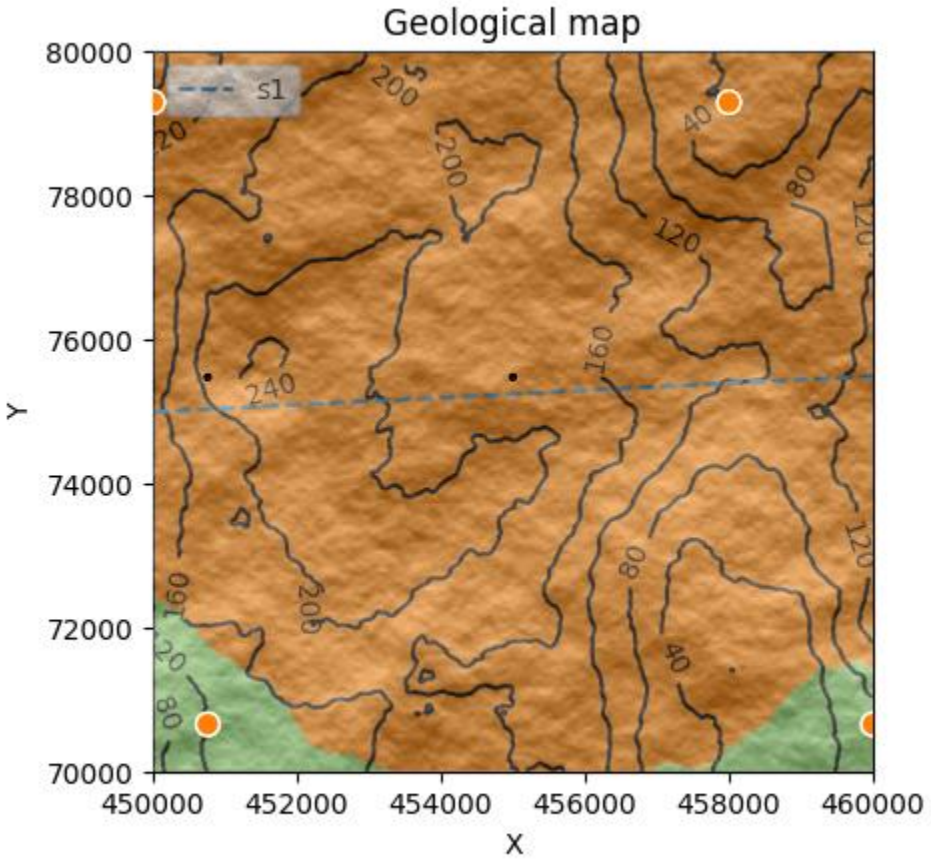
Out:

```
Lithology ids
  [2. 2. 2. ... 1. 1. 1.]
```

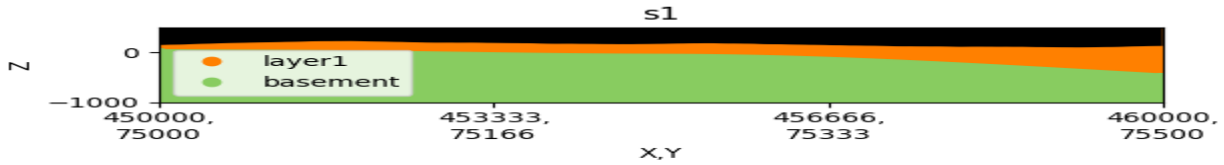
2.4 Trực quan hóa

Đối tượng chứa bản đồ địa chất được tính toán. Nó có thể được trực quan hóa bằng cách sử dụng chức năng vẽ đồ thị 2D và 3D:

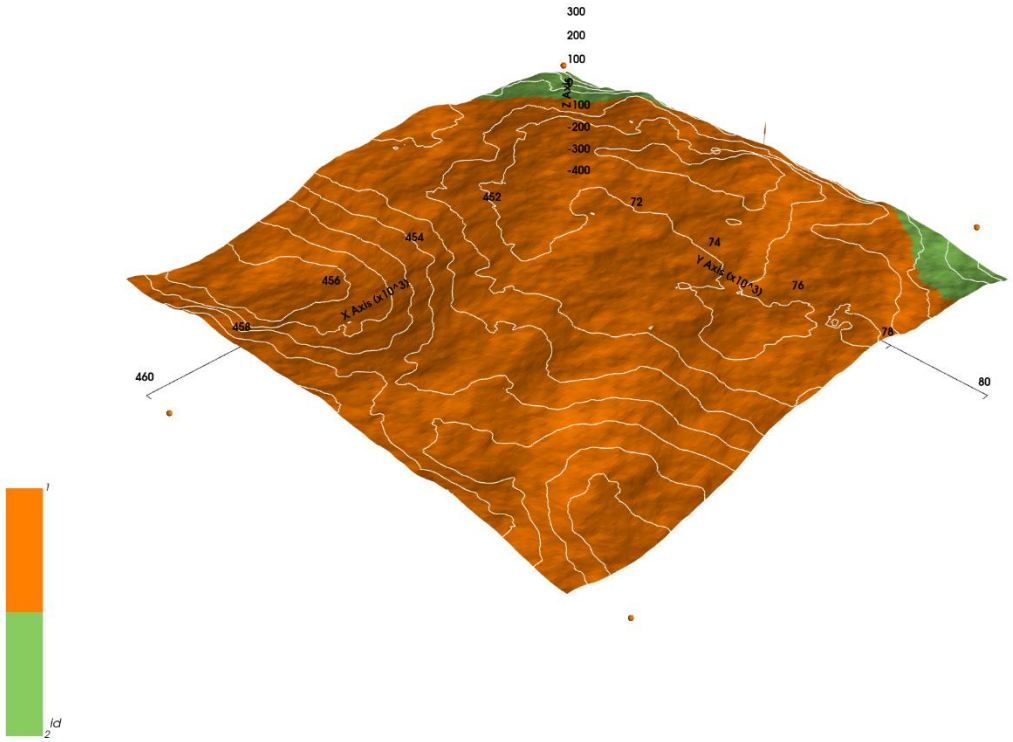
```
gp.plot_2d(geo_model, show_topography=True, section_names=['topography'], show_boundaries=False, show_data=True)
plt.show()
```




```
gp.plot_2d(geo_model, show_topography=True, section_names=['s1'])
plt.show()
```

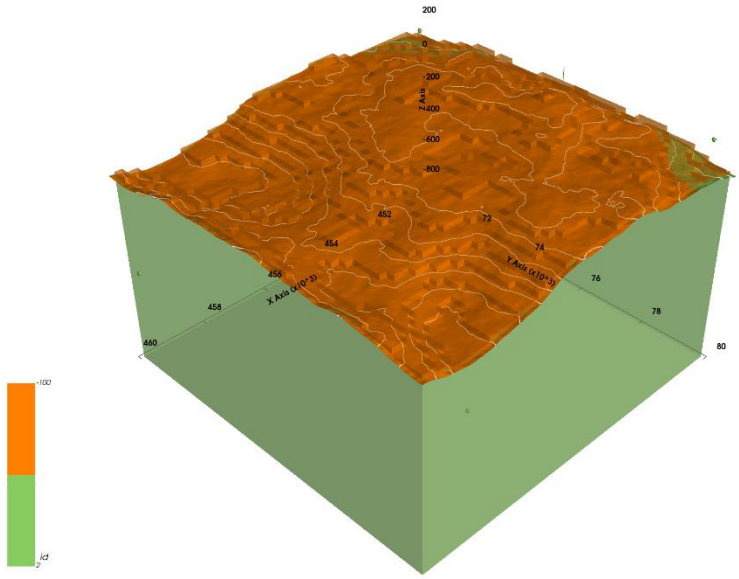



```
g3d = gp.plot_3d(geo_model,
                 show_topography=True,
                 show_lith=False,
                 show_surfaces=False,
                 show_results=False,
                 ve=5)
```




```
sphinx_gallery_thumbnail_number = 3
g3d = gp.plot_3d(geo_model,
```

```
            show_topography=True,
            show_lith=True,
            show_surfaces=True,
            ve=5)
```



2.5 Mặt cắt 2-D

Importing

```
import gempy as gp
import numpy as np
import matplotlib.pyplot as plt
np.random.seed(1234)
```

Setup the model

```
geo_model = gp.create_model('Tutorial_ch1-1_Basics')

# Importing the data from CSV-files and setting extent and resolution
data_path = 'https://raw.githubusercontent.com/cgre-aachen/gempy_data/master/'

gp.init_data(geo_model, [0, 2000., 0, 2000., 0, 2000.], [5, 5, 5],
             path_o=data_path + "/data/input_data/tut_chapter1/simple_fault_model_orientations.csv",
             path_i=data_path + "/data/input_data/tut_chapter1/simple_fault_model_points.csv",
default_values=True)
gp.map_stack_to_surfaces(geo_model,
                         {"Fault_Series": 'Main_Fault',
                          "Strat_Series": ('Sandstone_2', 'Siltstone',
                                           'Shale', 'Sandstone_1', 'basement')}, remove_unused_series=True)
geo_model.set_is_fault(['Fault_Series'])
```

Out:

```
Active grids: ['regular']
```

Fault colors changed. If you do not like this behavior, set change_color to False.

	order_series	BottomRelation	isActive	isFault	isFinite
Fault_Series	1	Fault	True	True	False
Strat_Series	2	Erosion	True	False	False

2.6 Thêm mặt cắt

Kiểm tra điểm đầu, điểm cuối và độ phân giải cho từng mặt cắt:

```
section_dict = {'section1': ([0, 0], [2000, 2000], [100, 80]),
                'section2': ([800, 0], [800, 2000], [150, 100]),
                'section3': ([0, 200], [1500, 500], [200, 150])}  # p1,p2,resolution
geo_model.set_section_grid(section_dict)
```

Out:

```
Active grids: ['regular' 'sections']
```

	start	stop	resolution	dist
section1	[0, 0]	[2000, 2000]	[100, 80]	2828.43
section2	[800, 0]	[800, 2000]	[150, 100]	2000.00
section3	[0, 200]	[1500, 500]	[200, 150]	1529.71

2.7 Thêm địa hình

```
geo_model.set_topography(fd=1.2, d_z=np.array([600, 2000]), resolution=np.array([50, 50]))
```

Out:

```
Active grids: ['regular' 'topography' 'sections']

Grid Object. Values:
array([[ 200.        ,  200.        ,  200.        ],
       [ 200.        ,  200.        ,  600.        ],
       [ 200.        ,  200.        , 1000.        ],
       ...,
       [1500.        ,  500.        , 1973.15436242],
       [1500.        ,  500.        , 1986.57718121],
       [1500.        ,  500.        , 2000.        ]])
```

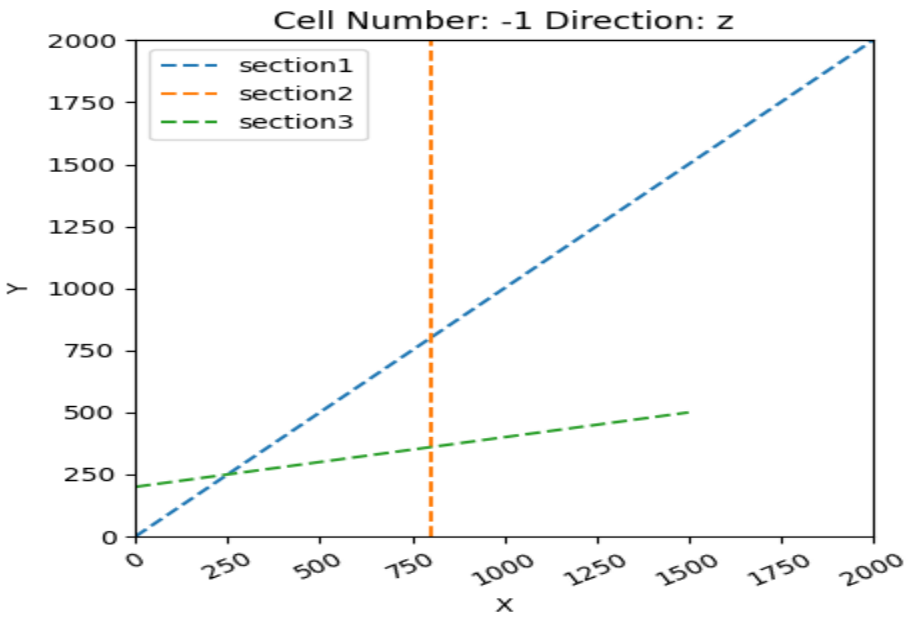
Active grids:

```
geo_model.get_active_grids()
```

Out:

```
array(['regular', 'topography', 'sections'], dtype='<U10')
```

```
gp.plot.plot_section_traces(geo_model)
plt.show()
```




```
gp.set_interpolator(geo_model)
```

Out:

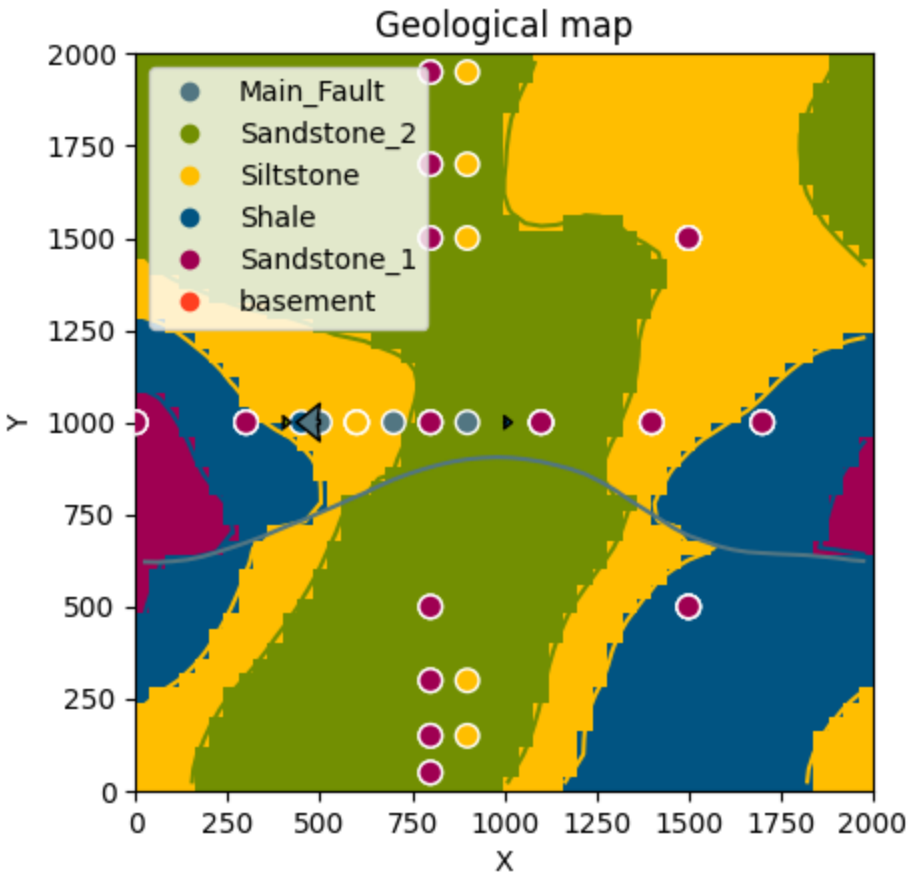
```
Setting kriging parameters to their default values.
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                values
range             3464.1
$C_o$           285714.29
drift equations     [3, 3]

<gempy.core.interpolator.InterpolatorModel object at 0x7fcc46b9b7c0>
```

```
sol = gp.compute_model(geo_model, compute_mesh=False)
```

Out:

```
/WorkSSD/PythonProjects/gempy/gempy/core/solution.py:173:       VisibleDeprecationWarning:
Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or
ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  self.geological_map = np.array(
/WorkSSD/PythonProjects/gempy/gempy/core/solution.py:178:       VisibleDeprecationWarning:
Creating an ndarray from ragged nested sequences (which is a list-or-tuple of lists-or-tuples-or
ndarrays with different lengths or shapes) is deprecated. If you meant to do this, you must specify
'dtype=object' when creating the ndarray.
  self.sections = np.array(
```

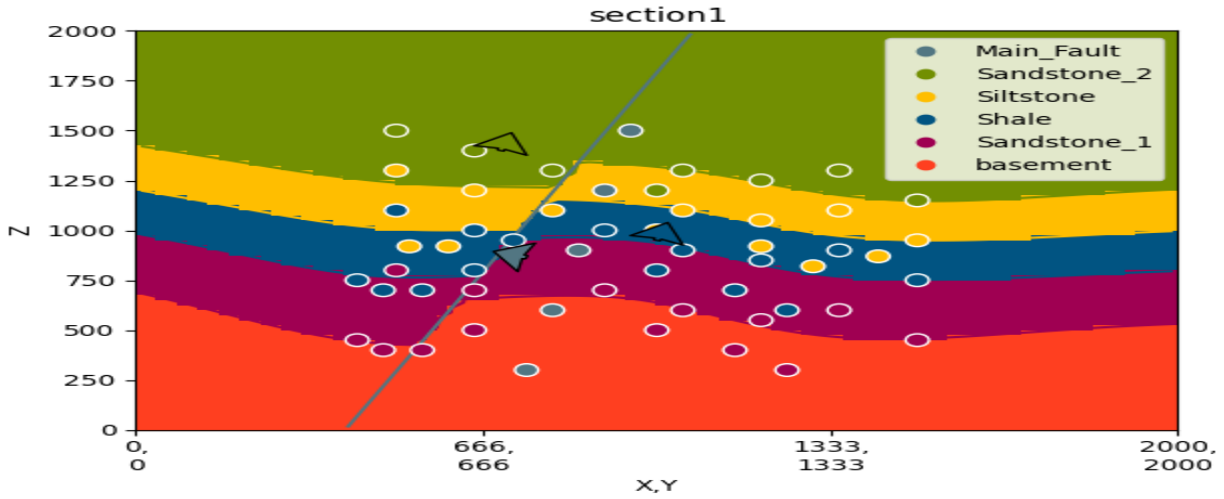
```
gp.plot_2d(geo_model, section_names=['topography'])
```



Out:

```
<gempy.plot.visualization_2d.Plot2D object at 0x7fcc46f04580>
```

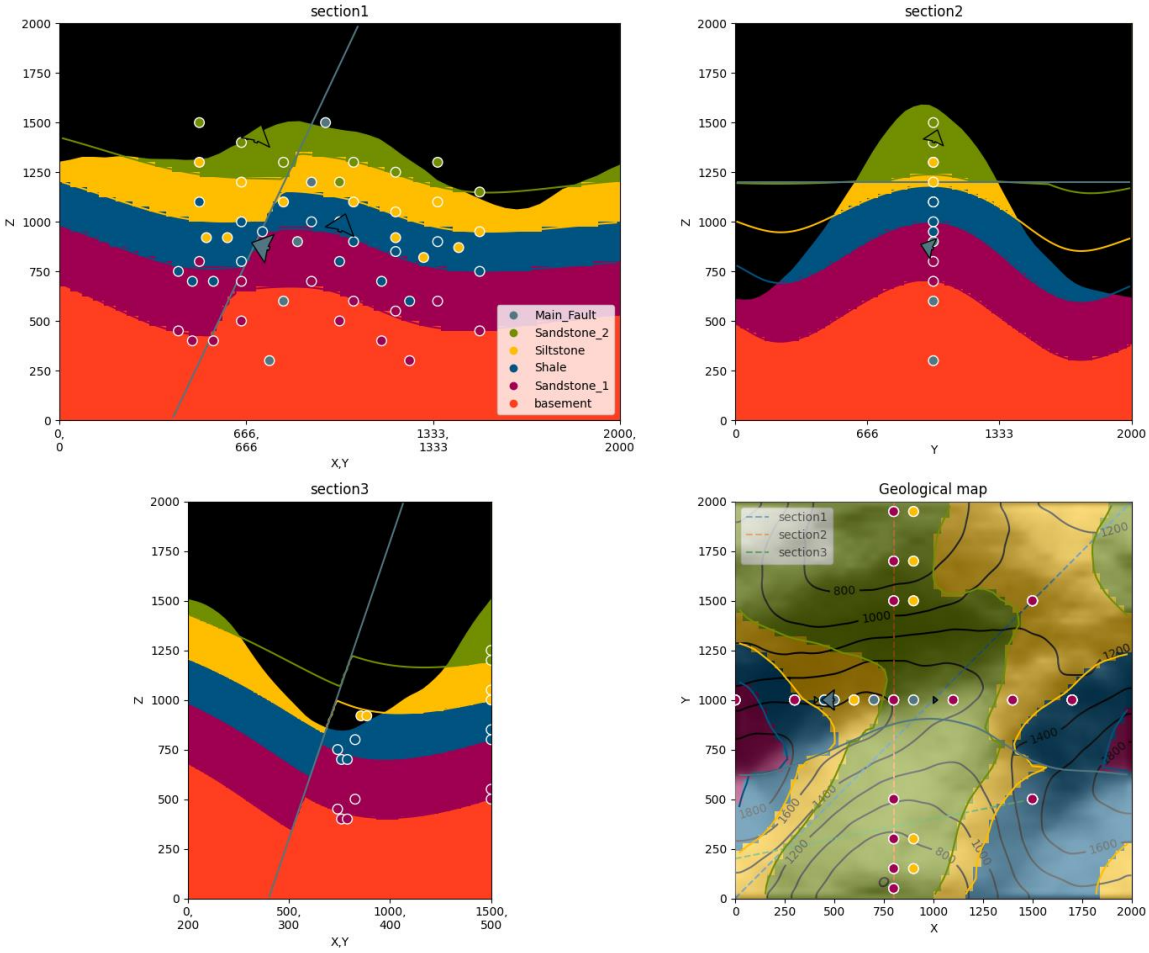
```
gp.plot_2d(geo_model, section_names=['section1'])
plt.show()
```



sphinx_gallery_thumbnail_number = 4

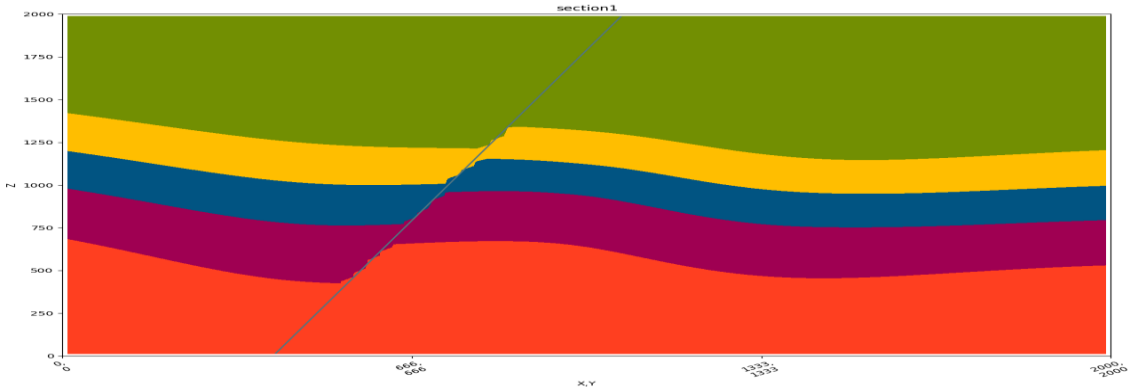
```
gp.plot_2d(geo_model, section_names=['section1', 'section2',
```

```
                        'section3', 'topography'],
          show_topography=True)
plt.show()
```



2.8 Tạo hình đa giác trong các mặt cắt

```
from gempy.core.grid_modules import section_utils
polygondict, cdict, extent = section_utils.get_polygon_dictionary(geo_model, 'section1')
```



this stores the xy points in the sections for every surface.

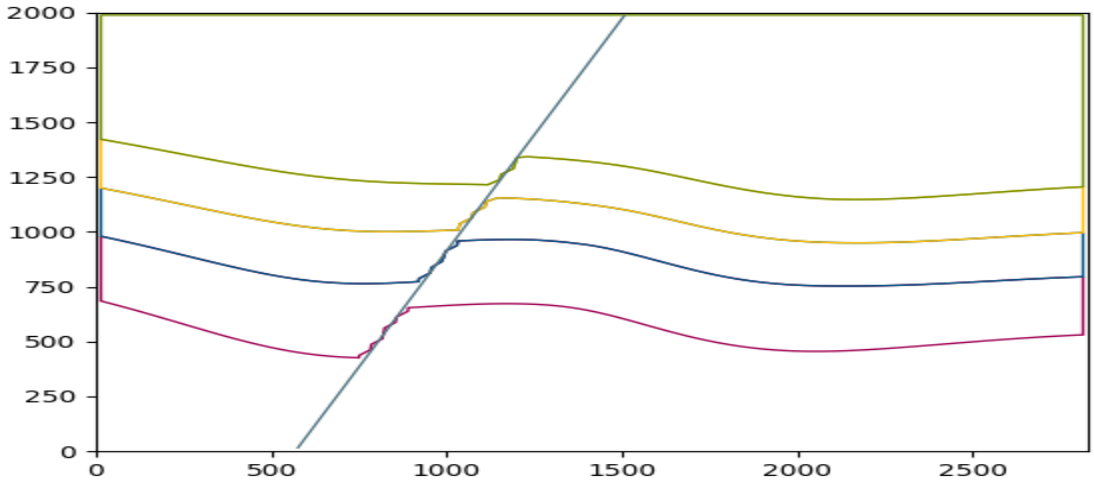
```
polygondict
```

Out:

```
{'Main_Fault': [Path(array([[ 574.17067095,   12.5       ],
       [ 579.82756057,   24.46977031],
       [ 585.98562123,   37.5       ],
       [ 597.80057151,   62.5       ],
       [ 608.11183182,   84.31824749],
       [ 609.61552173,   87.5       ],
       [ 621.43047197,  112.5       ],
       [ 633.24542214,  137.5       ],
```

2.9 Xem kết quả của các đa giác

```
import matplotlib.path
import matplotlib.patches as patches


def plot_pathdict(pathdict, cdict, extent, ax=None, surfaces=list(geo_model.surfaces.df['surface'])[:-1][::-1]):
    if ax == None:
        fig, ax = plt.subplots()
    for formation in surfaces:
        for path in pathdict.get(formation):
            if path !=[]:
                if type(path) == matplotlib.path.Path:
                    patch = patches.PathPatch(path, fill=False, lw=1, edgecolor=cdict.get(formation, 'k'))
                    ax.add_patch(patch)
                elif type(path) == list:
                    for subpath in path:
                        assert type(subpath == matplotlib.path.Path)
                        patch = patches.PathPatch(subpath, fill=False, lw=1, edgecolor=cdict.get(formation, 'k'))
                        ax.add_patch(patch)
    ax.set_ylim(extent[2:4])
    ax.set_xlim(extent[:2])
    plt.show()
plot_pathdict(polygondict, cdict, extent)
```




```
plot_pathdict(polygondict, cdict, extent, surfaces=['basement', 'Main_Fault'])

gp.save_model(geo_model)
```

Out:

```
True
```

CHƯƠNG 3: PHÂN TÍCH CẤU TRÚC LIÊN KẾT ĐỊA MÔ HÌNH

3.1 Tải mô hình ví dụ

```
import gempy as gp
from gempy.assets import topology as tp

import numpy as np
import matplotlib.pyplot as plt

import warnings
warnings.filterwarnings("ignore")
```

Đầu tiên, hãy thiết lập một mô hình ví dụ rất đơn giản. Để làm được điều đó, khởi tạo đối tượng geo_data với phạm vi mô hình chính xác và độ phân giải. Sau đó, tải các điểm dữ liệu từ tệp csv và thiết lập chuỗi và các lớp địa tầng (cột địa tầng).

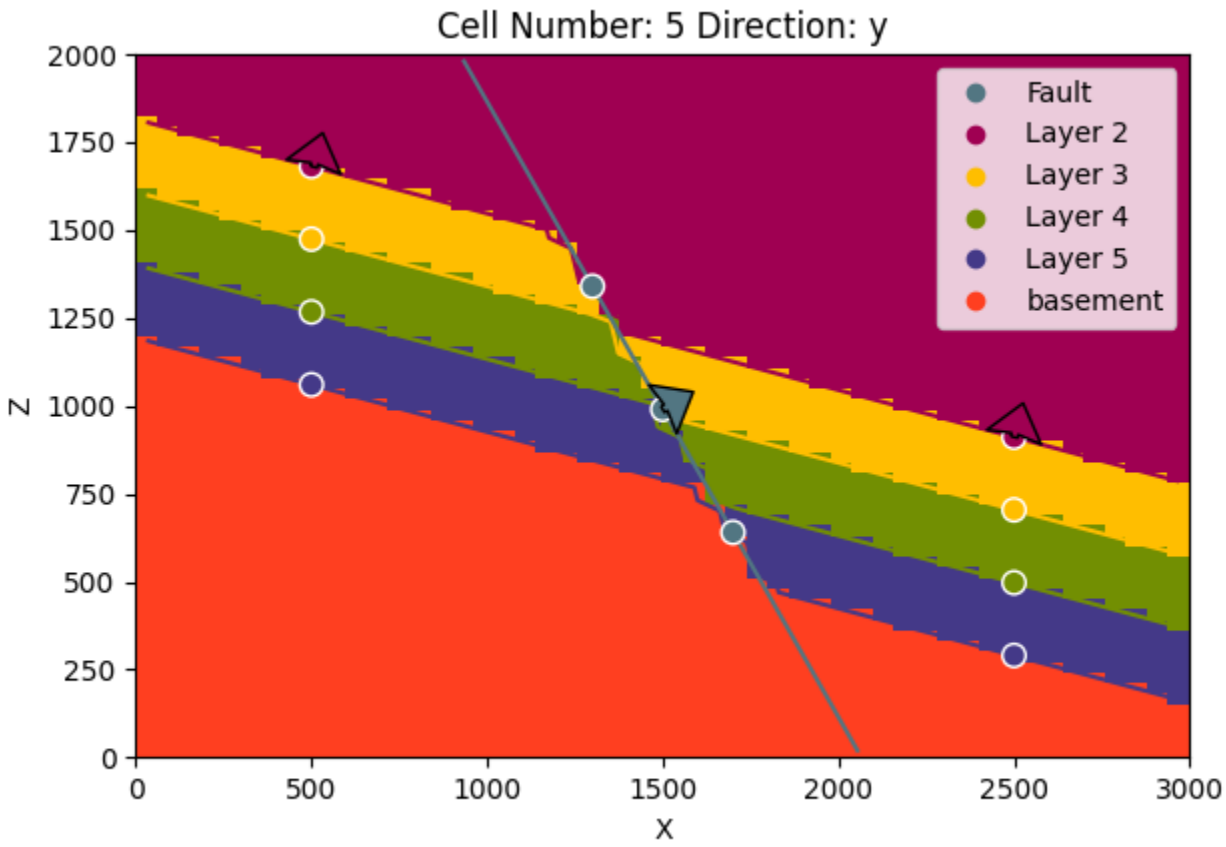
```
geo_model = gp.create_model("Model_Tutorial6")

data_path = 'https://raw.githubusercontent.com/cgre-aachen/gempy_data/master/'
gp.init_data(
    geo_model, [0, 3000, 0, 20, 0, 2000], [50, 10, 67],
    path_i=data_path+"data/input_data/tut_chapter6/ch6_data_interf.csv",
    path_o=data_path+"data/input_data/tut_chapter6/ch6_data_fol.csv"
)
gp.map_stack_to_surfaces(
    geo_model,
    {
        "fault": "Fault",
        "Rest": ('Layer 2', 'Layer 3', 'Layer 4', 'Layer 5')
    }
)
geo_model.set_is_fault(["fault"]);
gp.set_interpolator(geo_model)
sol = gp.compute_model(geo_model, compute_mesh=True)
```

Out:

```
Active grids: ['regular']
Fault colors changed. If you do not like this behavior, set change_color to False.
Setting kriging parameters to their default values.
Compiling theano function...
Level of Optimization:  fast_compile
Device:  cpu
Precision:  float64
Number of faults:  1
Compilation Done!
Kriging values:
                values
range             3605.61
```

```
$C_o$           309533.33
drift equations  [3, 3, 3]
```

```
gp.plot_2d(geo_model, cell_number=[5])
```



Out:

```
<gempy.plot.visualization_2d.Plot2D object at 0x7fcb84e4ff70>
```

3.2 Phân tích cấu trúc địa hình

GemPy có chức năng tích hợp để phân tích cấu trúc liên kết địa hình của các mô hình của nó. Tất cả những gì cần cho việc này là đối tượng geo_data, khối thạch học và khối đứt gãy. Nhập chúng vào gp.topology_compute và nhận được một số kết quả đầu ra hữu ích:

- Một đồ thị kề G, đại diện cho các mối quan hệ tôpô của mô hình
- Trọng tâm của tất cả các vùng tôpô duy nhất trong mô hình (tọa độ x, y, z)
- Danh sách tất cả các nhãn duy nhất (label_unique)
- Hai bảng tra cứu từ id thạch học đến nhãn nút và ngược lại

```
edges, centroids = tp.compute_topology(geo_model)
```

Đầu ra đầu tiên của hàm cấu trúc liên kết là tập hợp các cạnh đại diện cho các mối quan hệ cấu trúc liên kết giữa các thể địa chất duy nhất của mô hình khối. Một cạnh được biểu thị bằng một bộ hai nhãn int geobody (hoặc nút):

```
edges
```

Out:

{(9, 10), (4, 10), (1, 2), (3, 4), (1, 8), (3, 10), (2, 3), (2, 9), (1, 7), (4, 5), (3, 9), (5, 10), (6, 7), (8, 9), (1, 6), (7, 8), (2, 8)}

Đầu ra thứ hai là centroid dict, ánh xạ id đơn vị địa chất duy nhất (id nút biểu đồ) đến vị trí tâm thể địa chất trong tọa độ lưới:

```
centroids
```

Out:

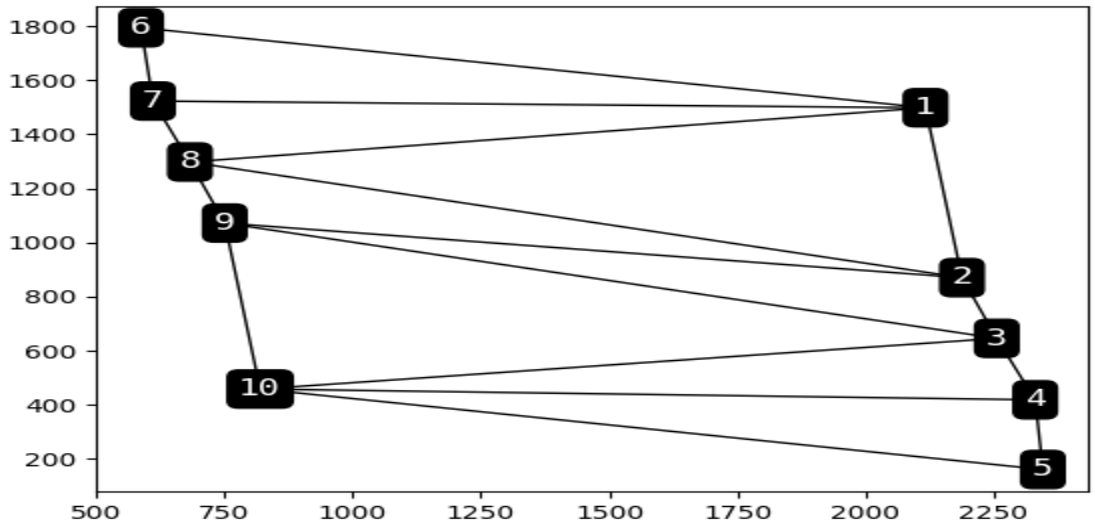
```
{1: array([35.27893175,  4.5       , 50.19485658]), 2: array([36.46666667,  4.5       , 29.14444444]),
3: array([37.59756098,  4.5       , 21.62195122]), 4: array([38.84563758,  4.5       , 14.00671141]),
5: array([39.09550562,  4.5       ,  5.37640449]), 6: array([ 9.79081633,  4.5       , 60.10204082]),
7: array([10.17687075,  4.5       , 51.02721088]), 8: array([11.37804878,  4.5       , 43.47560976]),
9: array([12.51098901,  4.5       , 35.90659341]), 10: array([13.659857  ,  4.5       , 15.34320735])}
```

Sau khi tính toán cấu trúc liên kết mô hình, chúng ta có thể phủ biểu đồ cấu trúc liên kết lên phần mô hình

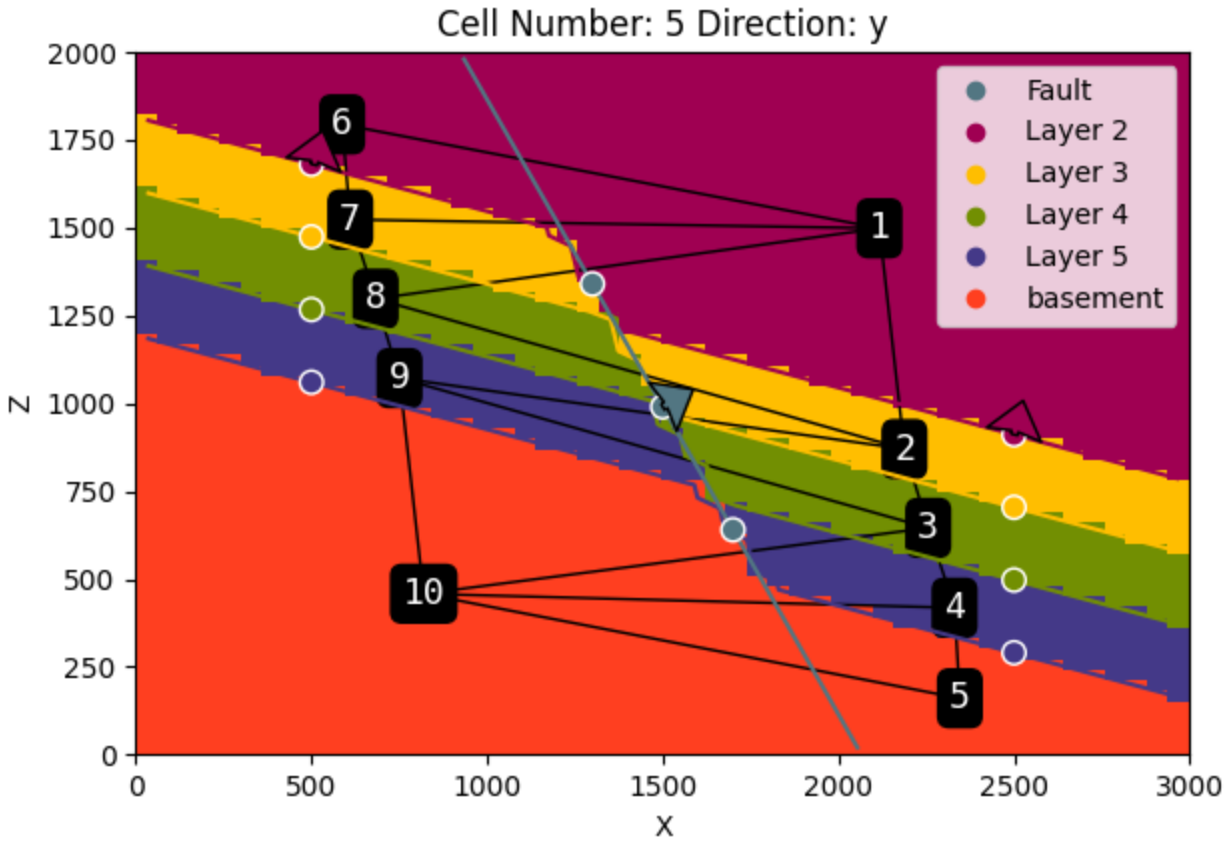
3.3 Trực quan hóa cấu trúc địa hình

Trực quan hóa đồ thị topo 2D

```
gp.plot.plot_topology(geo_model, edges, centroids)
plt.show()
```




```
gp.plot_2d(geo_model, cell_number=[5], show=False)
gp.plot.plot_topology(geo_model, edges, centroids, scale=True)
plt.show()
```



3.4 Ma trận kề

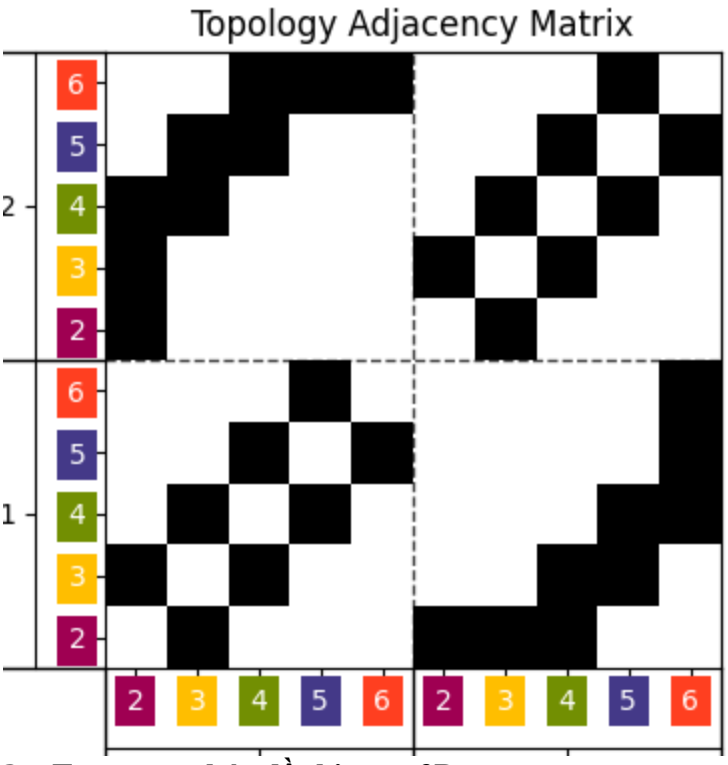
Một cách khác để mã hóa và trực quan hóa cấu trúc liên kết mô hình địa hình là sử dụng biểu đồ kề:

```
M = tp.get_adjacency_matrix(geo_model, edges, centroids)
print(M)
```

Out:

```
[[False  True False False False  True  True  True False False]
 [ True False  True False False False False  True  True False]
 [False  True False  True False False False False  True  True]
 [False False  True False  True False False False False  True]
 [False False False  True False False False False False  True]
 [ True False False False False False  True False False False]
 [ True False False False False  True False  True False False]
 [ True  True False False False False  True False  True False]
 [False  True  True False False False False  True False  True]
 [False False  True  True  True False False False  True False]]
```

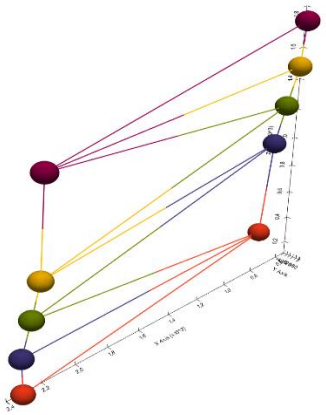
```
tp.plot_adjacency_matrix(geo_model, M)
```



3.5 Trực quan hóa đồ thị topo 3D

Có thể vẽ cấu trúc liên kết trong 3-D bằng cách sử dụng bộ công cụ trực quan hóa 3-D của GemPy được cung cấp bởi pyvista:

```
from gempy.plot._vista import Vista
gpv = Vista(geo_model)
gpv.plot_topology(edges, centroids)
gpv.show()
```



Out:

```
surface
Fault      #527682
Layer 2    #9f0052
Layer 3    #ffbe00
```

```
Layer 4     #728f02
Layer 5     #443988
basement    #ff3f20
Name: color, dtype: object
```

3.6 Bảng tra cứu

Nội dung cấu trúc liên kết cung cấp một số bảng tra cứu để làm việc với id cấu trúc liên kết đơn vị địa chất duy nhất.

```
lith_lot = tp.get_lot_node_to_lith_id(geo_model, centroids)
lith_lot
```

Out:

```
{1: 2, 2: 3, 3: 4, 4: 5, 5: 6, 6: 2, 7: 3, 8: 4, 9: 5, 10: 6}
```

Tìm ra các nút nằm trong khối đứt gãy nào:

```
fault_lot = tp.get_lot_node_to_fault_block(geo_model, centroids)
fault_lot
```

Out:

```
{1: 0, 2: 0, 3: 0, 4: 0, 5: 0, 6: 1, 7: 1, 8: 1, 9: 1, 10: 1}
```

Có thể dễ dàng ánh xạ id thạch học với id cấu trúc liên kết tương ứng:

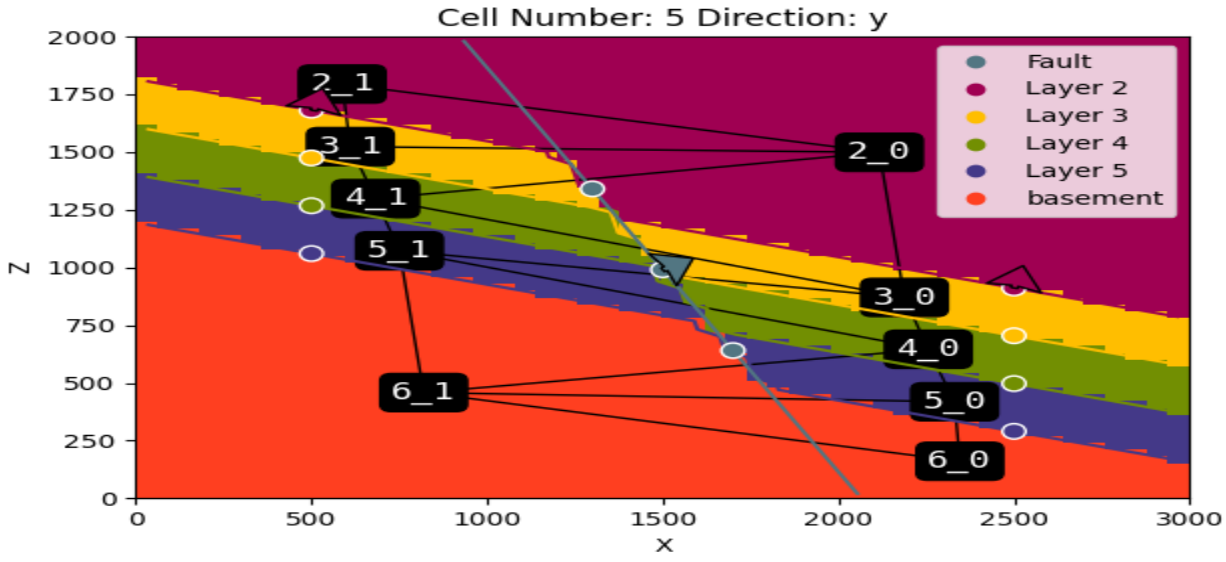
```
tp.get_lot_lith_to_node_id(lith_lot)
```

Out:

```
{2: [1, 6], 3: [2, 7], 4: [3, 8], 5: [4, 9], 6: [5, 10]}
```

3.7 Ghi nhãn nút chi tiết

```
sphinx_gallery_thumbnail_number = 4
dedges, dcentroids = tp.get_detailed_labels(geo_model, edges, centroids)
gp.plot_2d(geo_model, cell_number=[5], show=False)
gp.plot.plot_topology(geo_model, dedges, dcentroids, scale=True)
plt.show()
```




```
dedges
```

Out:

```
{('3_0', '4_0'), ('3_1', '4_1'), ('2_0', '2_1'), ('4_1', '5_1'), ('6_0', '6_1'), ('4_0', '6_1'), ('4_0', '5_1'), ('2_0', '4_1'), ('2_0', '3_1'), ('2_0', '3_0'), ('5_0', '6_1'), ('3_0', '5_1'), ('5_0', '6_0'), ('2_1', '3_1'), ('3_0', '4_1'), ('5_1', '6_1'), ('4_0', '5_0')}
```

```
dcentroids
```

Out:

```
{'2_0': array([35.27893175,  4.5       , 50.19485658]), '3_0': array([36.46666667,  4.5       , 29.14444444]), '4_0': array([37.59756098,  4.5       , 21.62195122]), '5_0': array([38.84563758,  4.5       , 14.00671141]), '6_0': array([39.09550562,  4.5       ,  5.37640449]), '2_1': array([ 9.79081633,  4.5       , 60.10204082]), '3_1': array([10.17687075,  4.5       , 51.02721088]), '4_1': array([11.37804878,  4.5       , 43.47560976]), '5_1': array([12.51098901,  4.5       , 35.90659341]), '6_1': array([13.659857  ,  4.5       , 15.34320735])}
```

3.8 Kiểm tra độ gần kề

Giả sử muốn kiểm tra xem lớp màu tím (id 5) có được kết nối qua đứt gãy với lớp màu vàng (id 3) hay không. Để làm được điều này, có thể dễ dàng sử dụng nhãn chi tiết và hàm check_adjacency:

```
tp.check_adjacency(dedges, "5_1", "3_0")
```

Out:

```
True
```

Có thể kiểm tra tất cả các thể địa chất tiếp giáp với lớp màu tím (id 5) ở phía bên trái của lỗi (đứt gãy id 1):

```
tp.get_adjacencies(dedges, "5_1")
```

Out:

```
{'6_1', '4_1', '4_0', '3_0'}
```

TÀI LIỆU THAM KHẢO

[1]. Miguel de la Varga, Alexander Schaaf, and Florian Wellmann (2019) *GemPy 1.0: open-source stochastic geological modeling and inversion*. Institute for Computational Geoscience and Reservoir Engineering, Geosci. Model Dev. Discuss., https://doi.org/10.5194/gmd-2018-61, Manuscript under review for journal Geosci. Model Dev.

[2]. https://www.gempy.org/: GemPy-Open-source 3D geological modeling

[3]. Calcagno, P., Chilès, J. P., Courrioux, G., & Guillen, A. (2008). *Geological modelling from field data and geological knowledge: Part I. Modelling method coupling 3D potential-field interpolation and geological rules.* Physics of the Earth and Planetary Interiors, 171(1-4), 147-157.

[4]. Lajaunie, C., Courrioux, G., & Manuel, L. (1997). *`Foliation fields and 3D cartography in geology: principles of a method based on potential interpolation.* Mathematical Geology, 29(4), 571-584.