pygeomod Documentation

Release 0.2

Florian Wellmann

March 25, 2015

Contents

Contents:

Contents

CHAPTER 1

pygeomod

1.1 Python wrapper for Geomodeller

pygeomod is a Python package with several modules to manipulate and control geological models created with Geomodeller. It contains methods to automate model computation and export (for structured and unstructured grids), to change input files, and to adapt exported and imported data sets around geological models.

The current version of the repository can be accessed on github:

https://github.com/flohorovicic/pygeomod

No guarantee for any working order :-) However, if you have any questions, feel free to email me!

1.2 Getting Started

The best way to get started (after installation, that is) is to go through the IPython tutorial and example notebooks. I will try to keep them as self-consistent as possible (with all data sets provided in the package, as well).

1.3 Documentation

The best way to view the documentation online is through the ReadTheDocs page:

http://pygeomod.readthedocs.org/en/latest/

The documentation is also available in PDF and HTML versions as part of the repository: HTML: https://github.com/flohorovicic/pygeomod/tree/master/doc/*build/html/index.html PDF: https://github.com/flohorovicic/pygeomod/tree/master/doc/*build/latex/pygeomod.pdf

1.4 Installation

Simplest way (if you have pip installed): download from Python Package Index and install:

pip install pygeomod

For more information on pip, see http://www.pip-installer.org/en/latest/quickstart.html

NOTE: to be on the safe side, it is a good idea to install experiemntal Python packages into a virtual environment first:

https://python-packaging-user-guide.readthedocs.org/en/latest/tutorial.html

Alternatively, you can download the distribution (either from PyPI or from github) and simply run

python setup.py install

1.5 License

pygeomodeller is free software and published under an MIT License (basically, you can do anything you want with the code, as long as you provide attribution back to me and don't hold me liable).

If you use this work in any form in a publication, please cite our article as reference:

J. Florian Wellmann, Stefan Finsterle, Adrian Croucher, Integrating Structural Geological Data into the Inverse Modelling Framework of iTOUGH2, Computers & Geosciences, Available online 31 October 2013, ISSN 0098-3004, http://dx.doi.org/10.1016/j.cageo.2013.10.014.

The manuscript is available online at: http://www.sciencedirect.com/science/article/pii/S0098300413002781

Disclaimer: the code 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.

Have fun!

CHAPTER 2

Analysis and modification of exported 3D Structural Data

All structural data from an entire GeoModeller project can be exported into ASCII files using the function in the GUI:

Export -> 3D Structural Data

This method generates files for defined geological parameters:

- "Points" (i.e. formation contact points) and
- "Foliations" (i.e. orientations/ potential field gradients).

Exported parameters include all those defined in sections as well as 3D data points.

The `struct_data` package contains methods to check, visualise, and extract/modify parts of these exported data sets, for example to import them into a different Geomodeller project. Several of these methods will be explained and tested in this notebook.

The export function generates files with the endings `_Foliations.csv` and `_Points.csv`. Classes for both of these data sets are available to handle them accordingly.

```
# import module
import sys
# as long as not in Pythonpath, we have to set directory:
sys.path.append(r'/Users/flow/git/pygeomod')
import struct_data

%pylab inline

Populating the interactive namespace from numpy and matplotlib

WARNING: pylab import has clobbered these variables: ['norm']
%matplotlib prevents importing * from pylab and numpy
```

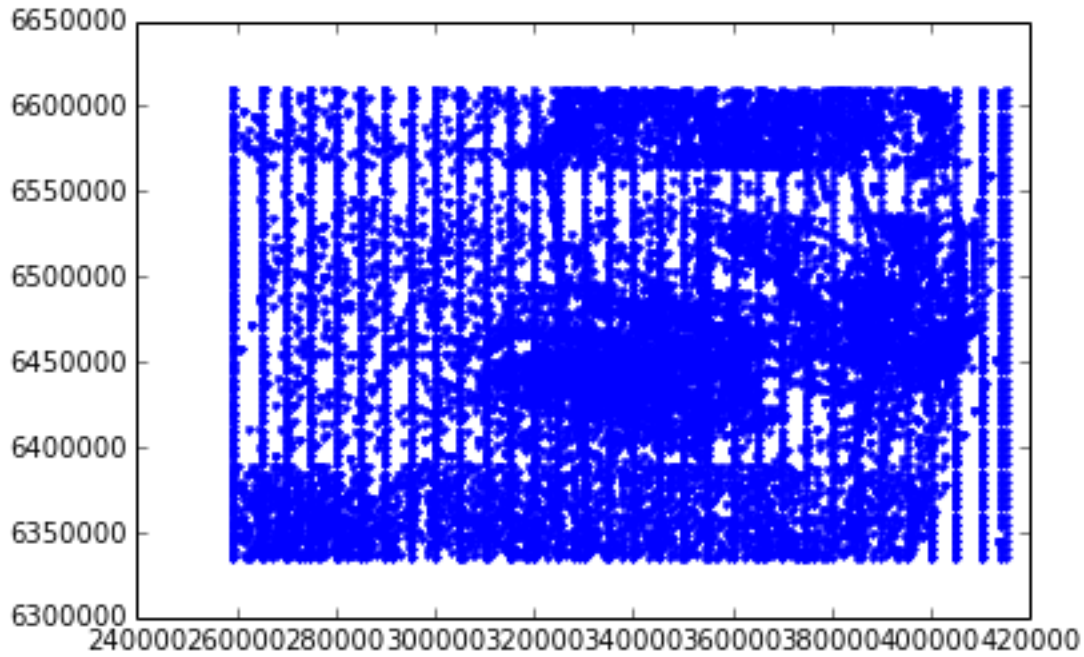
2.1 Point data sets

We will first start with reading and analysing the point data sets:

```
# only required during testing stage
reload(struct_data)
# use example data
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')

pts.get_formation_names()
pts.formation_names
```

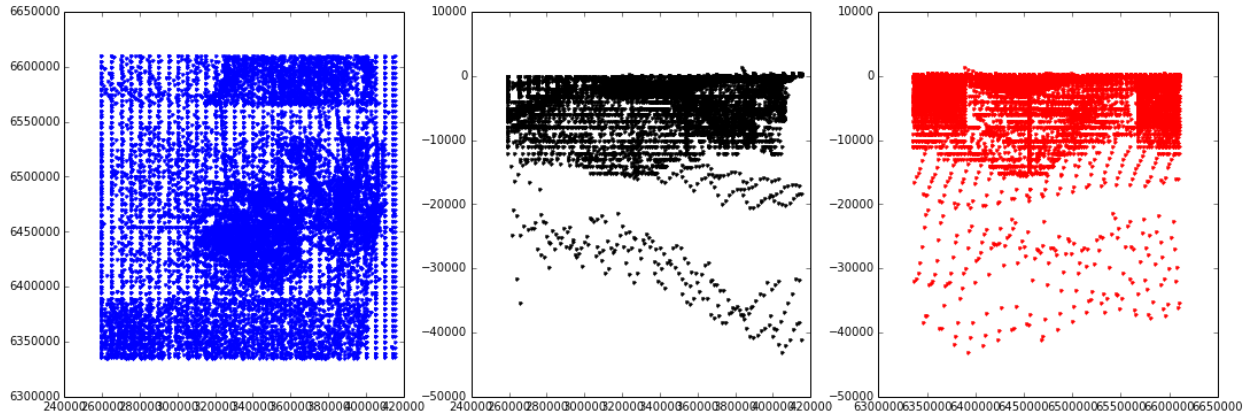
```
array(['Basement', 'CPB_Fault_01', 'CPB_Fault_02', 'CPB_Fault_03',
       'CPB_Fault_04', 'CPB_Fault_05', 'CPB_Fault_06', 'CPB_Fault_07',
       'CPB_Fault_08', 'CPB_Fault_09', 'CPB_Fault_10',
       'CPB_Fault_11_Serpentine', 'CPB_Fault_12_Darling', 'CPB_Fault_13',
       'CPB_Fault_14', 'CPB_Fault_15', 'CPB_Fault_16',
       'CPB_Fault_Transverse_02', 'CPB_Fault_Transverse_03',
       'CPB_Fault_Transverse_04', 'CPB_Fault_Transverse_05',
       'Cattamarra_Coal_Measures', 'Eneabba_Fm', 'Gage_Ss',
       'Kockatea_Shale', 'Late_Triassic', 'Leederville_Fm', 'Lesueur_Ss',
       'Lower_crust', 'Moho', 'Neocomian_Unc', 'Permian', 'Sea_level',
       'South_Perth_Sh', 'Topo', 'Yarragadee_Fm', 'Yilgarn'],
      dtype='|S32')
```

```
pts.plot_plane()
```



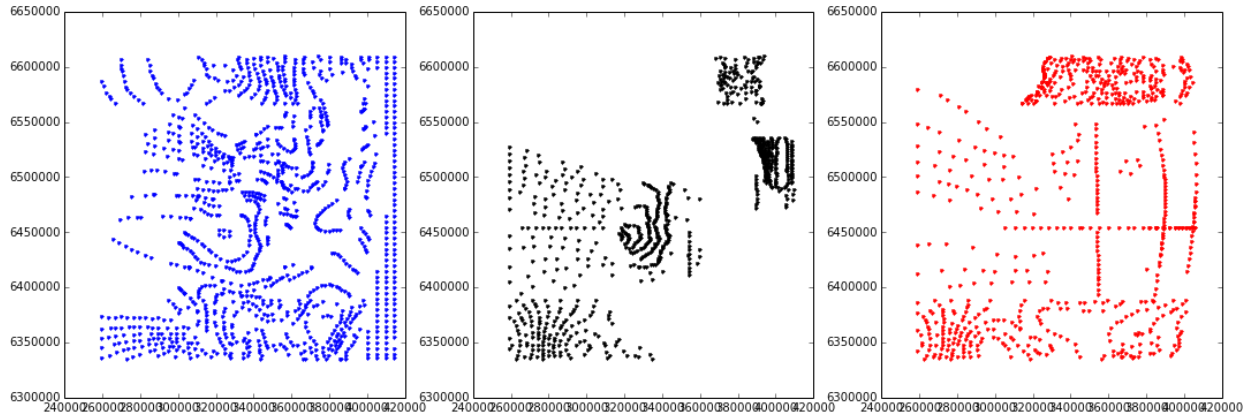
Or, a bit more complex plotting options:

```
fig = plt.figure(figsize = (18,6))
ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)
pts.plot_plane(ax = ax1)
pts.plot_plane(plane = ('x','z'), ax = ax2, color = 'k')
pts.plot_plane(plane = ('y','z'), ax = ax3, color = 'r')
```



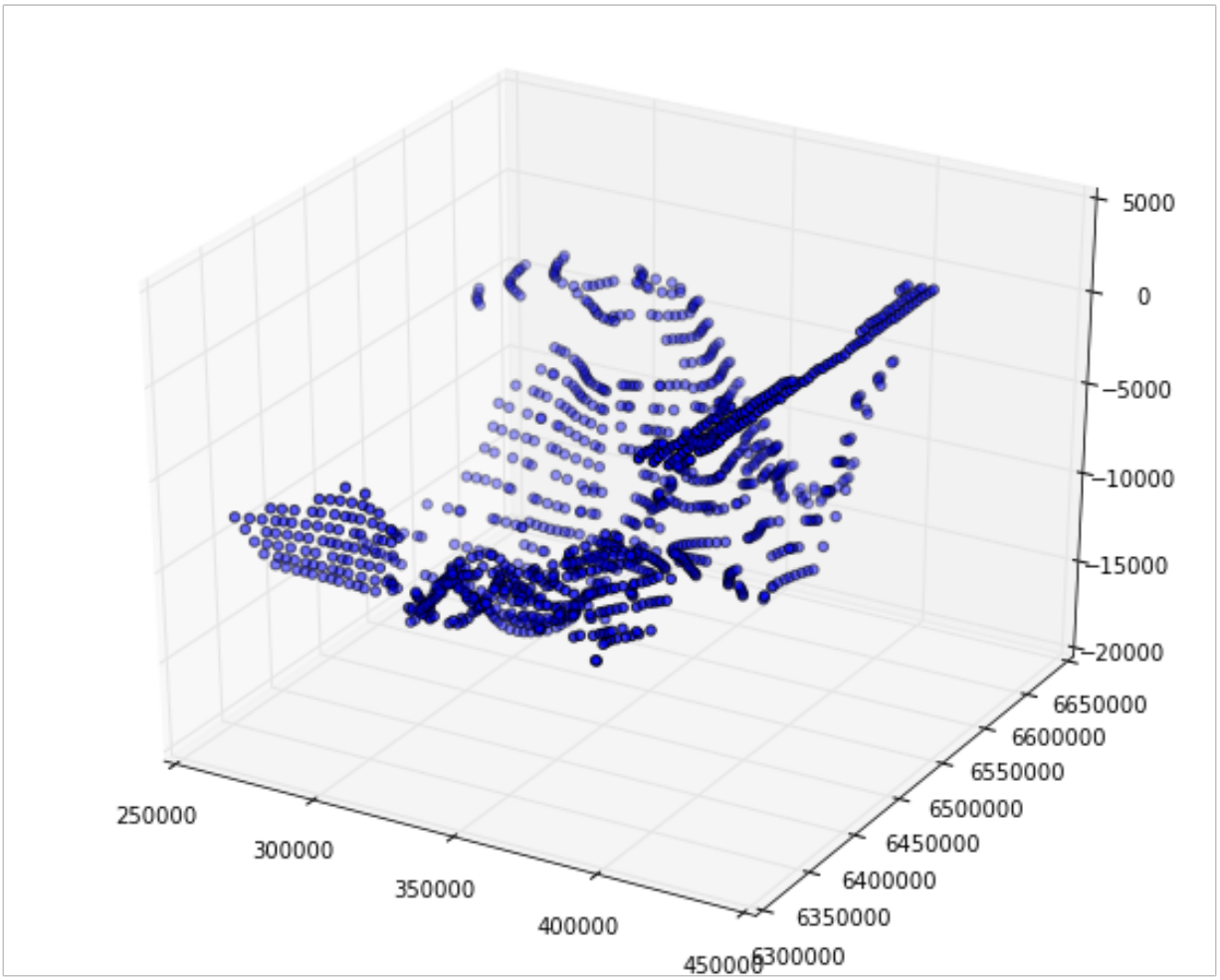
We can also create plots of points for specified formations only:

```
fig = plt.figure(figsize = (18,6))
ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)
pts.plot_plane(ax = ax1, formation_names = ['Basement'])
pts.plot_plane(ax = ax2, formation_names = ['Yarragadee_Fm'], color = 'k')
pts.plot_plane(ax = ax3, formation_names = ['Lesueur_Ss'], color = 'r')
```



It is also possible to create 3-D perspective plots (not many options yet, but a lot more could be included):

```
pts.plot_3D(formation_names = ['Basement'])
```




```
pts.formation_names
```

```
array(['Basement', 'CPB_Fault_01', 'CPB_Fault_02', 'CPB_Fault_03',
       'CPB_Fault_04', 'CPB_Fault_05', 'CPB_Fault_06', 'CPB_Fault_07',
       'CPB_Fault_08', 'CPB_Fault_09', 'CPB_Fault_10',
       'CPB_Fault_11_Serpentine', 'CPB_Fault_12_Darling', 'CPB_Fault_13',
       'CPB_Fault_14', 'CPB_Fault_15', 'CPB_Fault_16',
       'CPB_Fault_Transverse_02', 'CPB_Fault_Transverse_03',
       'CPB_Fault_Transverse_04', 'CPB_Fault_Transverse_05',
       'Cattamarra_Coal_Measures', 'Eneabba_Fm', 'Gage_Ss',
       'Kockatea_Shale', 'Late_Triassic', 'Leederville_Fm', 'Lesueur_Ss',
       'Lower_crust', 'Moho', 'Neocomian_Unc', 'Permian', 'Sea_level',
       'South_Perth_Sh', 'Topo', 'Yarragadee_Fm', 'Yilgarn'],
      dtype='|S32')
```

Generate subset for defined formations

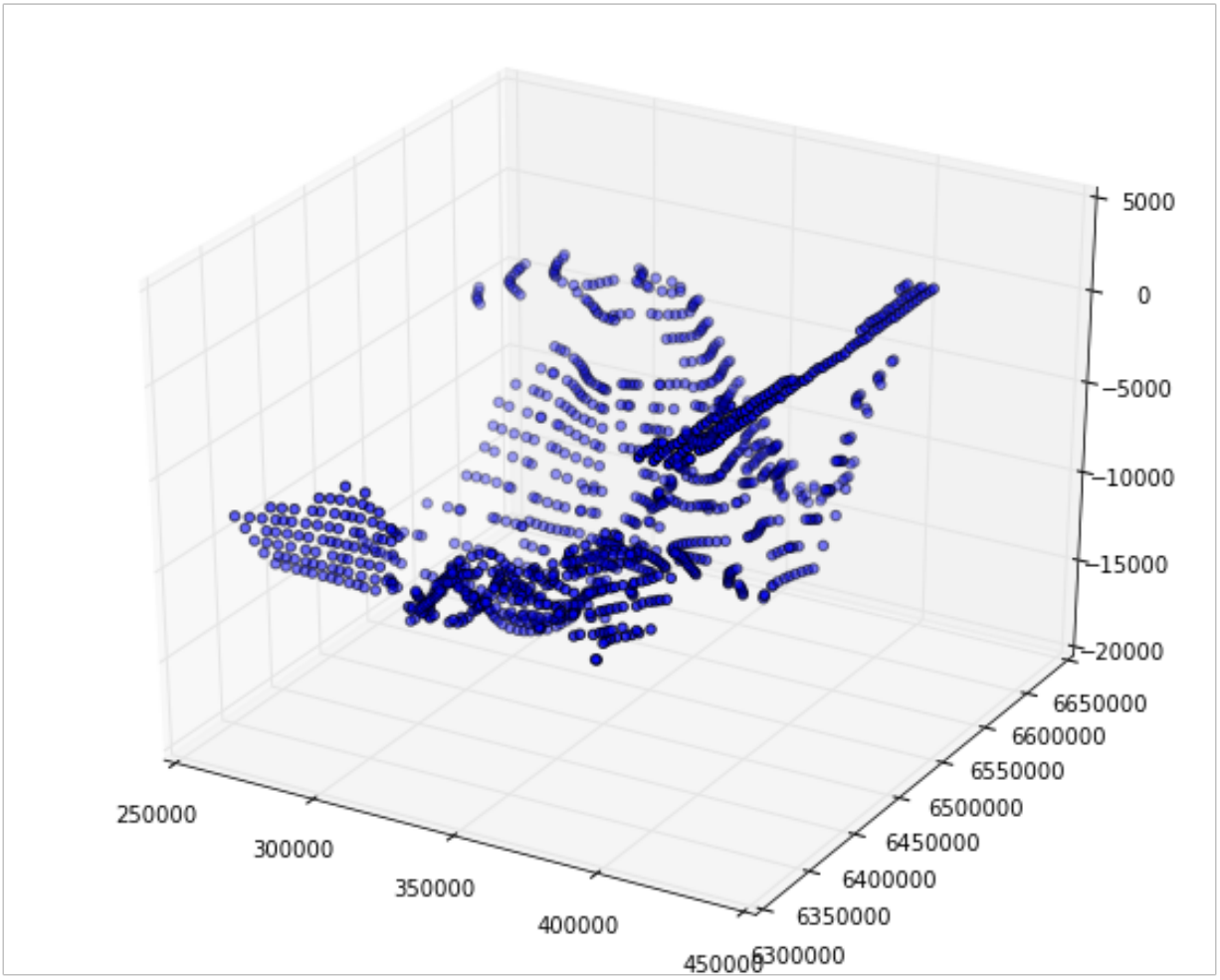
It is often required to generate a subset of the points, either for specified formations only (although that could also be done during the input in GeoModeller...), or for a defined range/ extent. These options are possible with the package, and selections can be stored to new files:

```
# only required during testing stage
reload(struct_data)
# use example data
```

```
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')
pts_subset = pts.create_formation_subset(['Basement'])

pts_subset.len
```

```
1209
```

```
pts_subset.plot_3D()
```



Generating a subset for a subvolume

It is also possible to extract data in only a specified range. The range is defined with keywords, e.g. `from_x = xx`, `to_x = xx`. All other ranges (not stated) are simply kept as before.

Here an example:

```
# only required during testing stage
reload(struct_data)
# use example data
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')

pts.xmin, pts.xmax, pts.ymin, pts.ymax
```

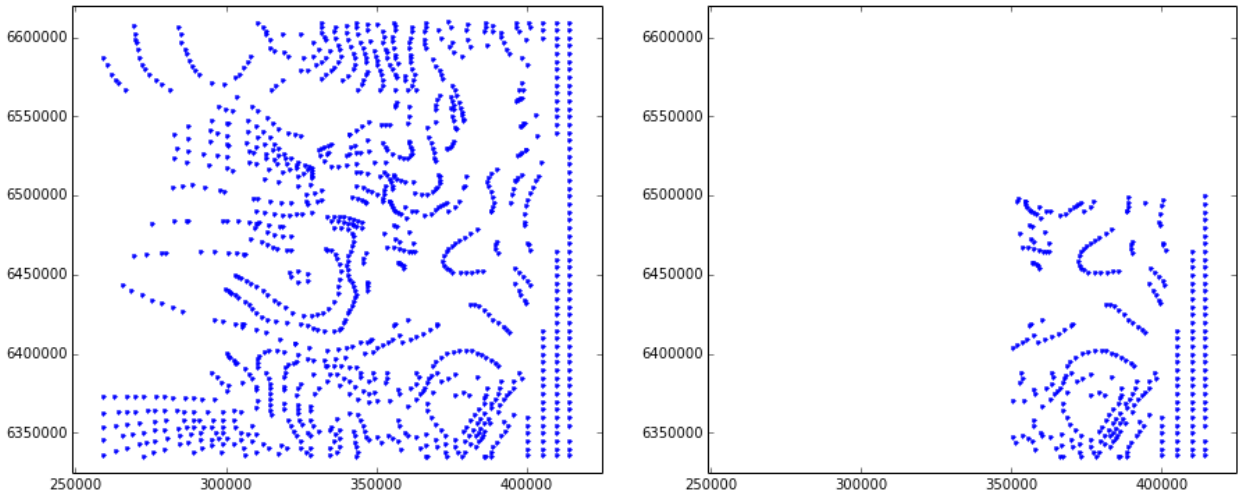
```
(259000.0, 415000.0, 6335020.0, 6610000.0)
```

```
pts_subset = pts.extract_range(from_x = 350000., to_y = 6500000)
```

Now let's compare the two sets in a plot:

```
fig = plt.figure(figsize = (15,6))
ax1 = fig.add_subplot(121)
ax2 = fig.add_subplot(122)
pts.plot_plane(ax = ax1, formation_names = ['Basement'])
pts_subset.plot_plane(ax = ax2, formation_names = ['Basement'])
# set ranges to compare plots
ax1.set_xlim((pts.xmin-10000, pts.xmax+10000))
ax1.set_ylim((pts.ymin-10000, pts.ymax+10000))
ax2.set_xlim((pts.xmin-10000, pts.xmax+10000))
ax2.set_ylim((pts.ymin-10000, pts.ymax+10000))
```

```
(6325020.0, 6620000.0)
```



Thin data

In many cases, a lot of data has been digitised for specific locations. This can lead to problems with the kriging interpolation in Geomodeller. A simple function to thin data is implemented here to avoid this problem. The method is really simple and can be quite compute intense (for smalll grids), so check what you are doing!

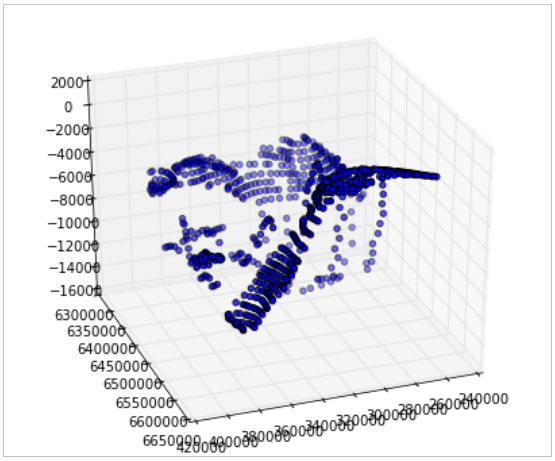
Note: If spatial data is imported, Geomodeller actually has a functionality for a type of "thinning" - if you work with this data (e.g. digitised from MapInfo), then this might be the function to use...

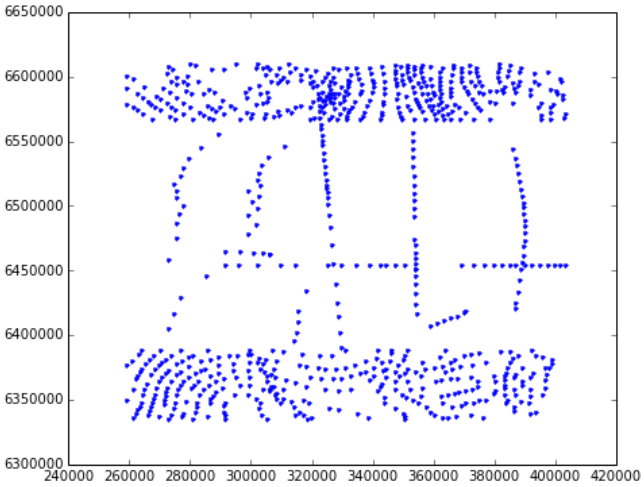
The thinning is performed for **the entire set** and **not aware of formations** (so far...) and works on a grid/ raster base for defined number of cells in each axis direction of `nx, ny, nz`. The best procedure is therefore to first create a subset for one formation, and then to perform the thinning for this subset:

```
# only required during testing stage
reload(struct_data)
# use example data
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')
pts_sub1 = pts.create_formation_subset('Permian')
```

Let's have a look at one formation, the Permian. We can clearly see the dense data in the overlapping regions and the highly defined traces through the model:

```
fig = plt.figure(figsize = (16,6))
ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122)
ax1.view_init(elev=30, azim=70)
pts_sub1.plot_3D(ax = ax1)
pts_sub1.plot_plane(ax = ax2)
```



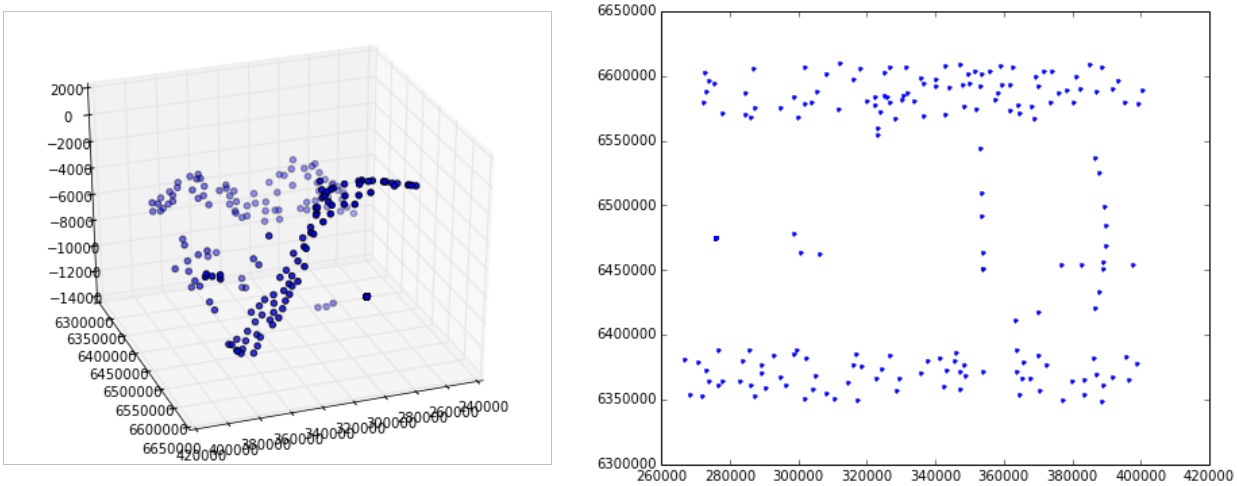


We want to thin this now with a grid which is `n` times smaller than the extent in each direction:

```
nx = 20
ny = 20
nz = 20

pts_sub2 = pts_sub1.thin(nx, ny, nz)

pts_sub1.len, pts_sub2.len

(810, 319)
```

And compare those to the set above in a plot:

```
fig = plt.figure(figsize = (16,6))
ax1 = fig.add_subplot(121, projection='3d')
ax2 = fig.add_subplot(122)
ax1.view_init(elev=30, azim=70)
pts_sub2.plot_3D(ax = ax1)
pts_sub2.plot_plane(ax = ax2)
```



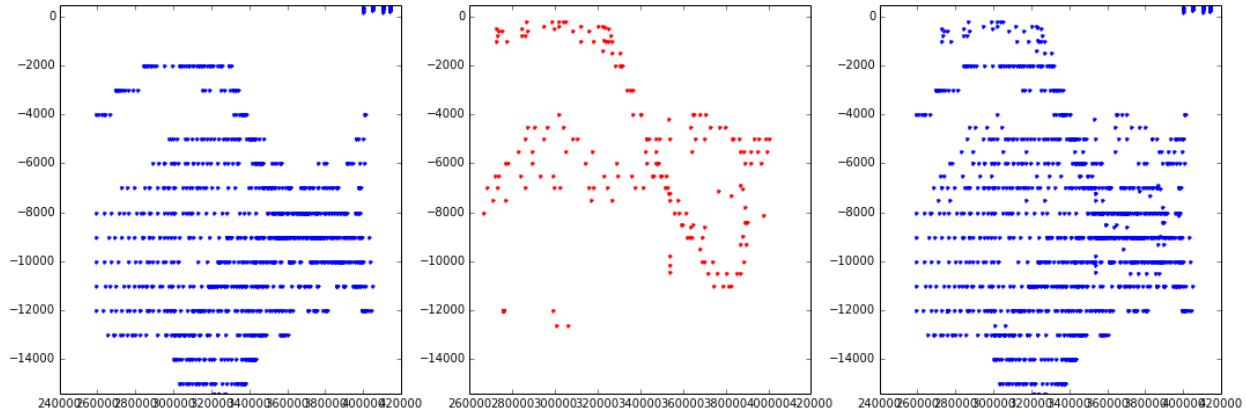
Nice, it seems to work (surprised myself...)

Of course, it's up to the user to evaluate if the thinning is creating "wrong" Geomodels afterwards... good luck!

Combine two point sets

It might be important to combine datasets, for example after thinning two different geological formations. As an example, we will combine the thinned Permian data set from above with the (complete) set for the Basement:

```
# only required during testing stage
reload(struct_data)
# use example data
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')
pts_sub3 = pts.create_formation_subset('Basement')
# create a copy for comparison:
pts_sub3b = pts.create_formation_subset('Basement')

pts_sub3.combine_with(pts_sub2)

# create a plot of all three point sets:
fig = plt.figure(figsize = (18,6))
ax1 = fig.add_subplot(131)
ax2 = fig.add_subplot(132)
ax3 = fig.add_subplot(133)
pts_sub3b.plot_plane(('x','z'), ax = ax1)
pts_sub2.plot_plane(('x','z'), ax = ax2, color='r')
pts_sub3.plot_plane(('x','z'), ax = ax3)
ax1.set_ylim((pts_sub3.zmin, pts_sub3.zmax))
ax2.set_ylim((pts_sub3.zmin, pts_sub3.zmax))
ax3.set_ylim((pts_sub3.zmin, pts_sub3.zmax))

(-15400.0, 454.0)
```



Note: This function changes the pointset in place and does not return a new one (as for thinning, subsetting, etc.)

Remove formation from point set

It is also possible to remove one or multiple formations from the set. This functionality can be used in combination with thinning and combining to thin only the data set for one formation, and then combine it back into the original set:

```
# only required during testing stage
reload(struct_data)
# use example data
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')

print("Original length of point set: %d" % pts.len)


# formation to perform thinning:
formation = 'Permian'

# first step: extract formation
pts_perm = pts.create_formation_subset(formation)

# now remove from original
pts.remove_formations(formation)

# perform thinning
nx = ny = nz = 20
pts_perm_thinned = pts_perm.thin(nx, ny, nz)

# and now combine back with original:
pts.combine_with(pts_perm_thinned)

print("New length of point set (after thinning): %d" % pts.len)
```

```
Original length of point set: 14184
New length of point set (after thinning): 13693
```

Save new set to file

Of course, we want to save this new and adapted data set to a file! This is also simply possible with:

```
# only required during testing stage
reload(struct_data)
# use example data
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')
```

```
# extract defined range and only Basement points:
pts_subset = pts.create_formation_subset(['Basement'])
pts_subset_1 = pts_subset.extract_range(from_x = 350000., to_y = 6500000)

pts_subset_1.save("new_Points.csv")
```

Change formation names

Sometime it is required to adjust the name of a formation or unit in the point set, for example for combination with another point set. This operation is possible with a dictionary for one or more formations with a mapping old -> new name:

```
# only required during testing stage
reload(struct_data)
# use example data
pts = struct_data.Struct3DPoints(filename = r'../data/wt_Points.csv')

print pts.formation_names

rename_dict = {'Basement' : 'Basement_new', 'CPB_Fault_01' : 'Euler'}

pts.rename_formations(rename_dict)

print pts.formation_names
```

```
['Basement' 'CPB_Fault_01' 'CPB_Fault_02' 'CPB_Fault_03' 'CPB_Fault_04'
 'CPB_Fault_05' 'CPB_Fault_06' 'CPB_Fault_07' 'CPB_Fault_08' 'CPB_Fault_09'
 'CPB_Fault_10' 'CPB_Fault_11_Serpentine' 'CPB_Fault_12_Darling'
 'CPB_Fault_13' 'CPB_Fault_14' 'CPB_Fault_15' 'CPB_Fault_16'
 'CPB_Fault_Transverse_02' 'CPB_Fault_Transverse_03'
 'CPB_Fault_Transverse_04' 'CPB_Fault_Transverse_05'
 'Cattamarra_Coal_Measures' 'Eneabba_Fm' 'Gage_Ss' 'Kockatea_Shale'
 'Late_Triassic' 'Leederville_Fm' 'Lesueur_Ss' 'Lower_crust' 'Moho'
 'Neocomian_Unc' 'Permian' 'Sea_level' 'South_Perth_Sh' 'Topo'
 'Yarragadee_Fm' 'Yilgarn']
Change name from CPB_Fault_01 to Euler
Change name from Basement to Basement_new
['Basement_new' 'CPB_Fault_02' 'CPB_Fault_03' 'CPB_Fault_04' 'CPB_Fault_05'
 'CPB_Fault_06' 'CPB_Fault_07' 'CPB_Fault_08' 'CPB_Fault_09' 'CPB_Fault_10'
 'CPB_Fault_11_Serpentine' 'CPB_Fault_12_Darling' 'CPB_Fault_13'
 'CPB_Fault_14' 'CPB_Fault_15' 'CPB_Fault_16' 'CPB_Fault_Transverse_02'
 'CPB_Fault_Transverse_03' 'CPB_Fault_Transverse_04'
 'CPB_Fault_Transverse_05' 'Cattamarra_Coal_Measures' 'Eneabba_Fm' 'Euler'
 'Gage_Ss' 'Kockatea_Shale' 'Late_Triassic' 'Leederville_Fm' 'Lesueur_Ss'
 'Lower_crust' 'Moho' 'Neocomian_Unc' 'Permian' 'Sea_level'
 'South_Perth_Sh' 'Topo' 'Yarragadee_Fm' 'Yilgarn']
```

More ideas

The methods could be used to check different geological interpretations quickly, i.e. different data sets for Moho structures, etc.: Create one model, load different data sets and compute the models.

It would also be possible to use the method directly to reproduce the "data thinning" example from Martin Putz' 2001 paper - might be interesting as an indication for interpolation stability.

In extension: it should be possible to perform a kind of "bootstrapping" method to test interpolation uncertainty with respect to data density!

I didn't check, but it might be possible to load 3D structural data through the API (although not sure, as a lot of checks are performed in the GUI). If this is possible, then all of the previous methods could easily be automated and/or

included in model validity and uncertainty estimation steps!

And with a bit of coding:

It should be relatively simple to add some more functionality, for example to create a simple data density plot, as a "zero order" estimation of model uncertainty with respect to available data (i.e. no data, high uncertainty).

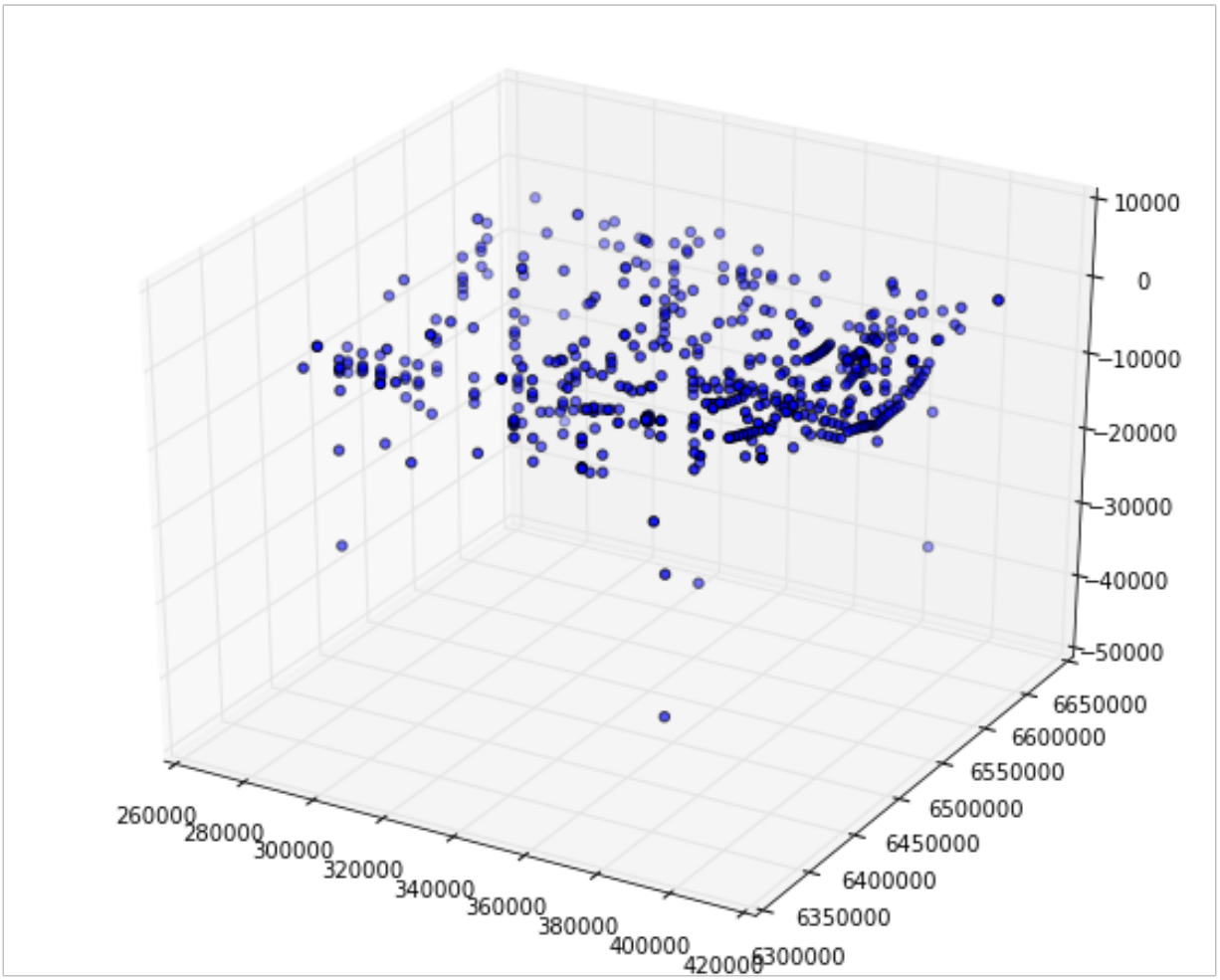
CHAPTER 3

Orientation Data Sets

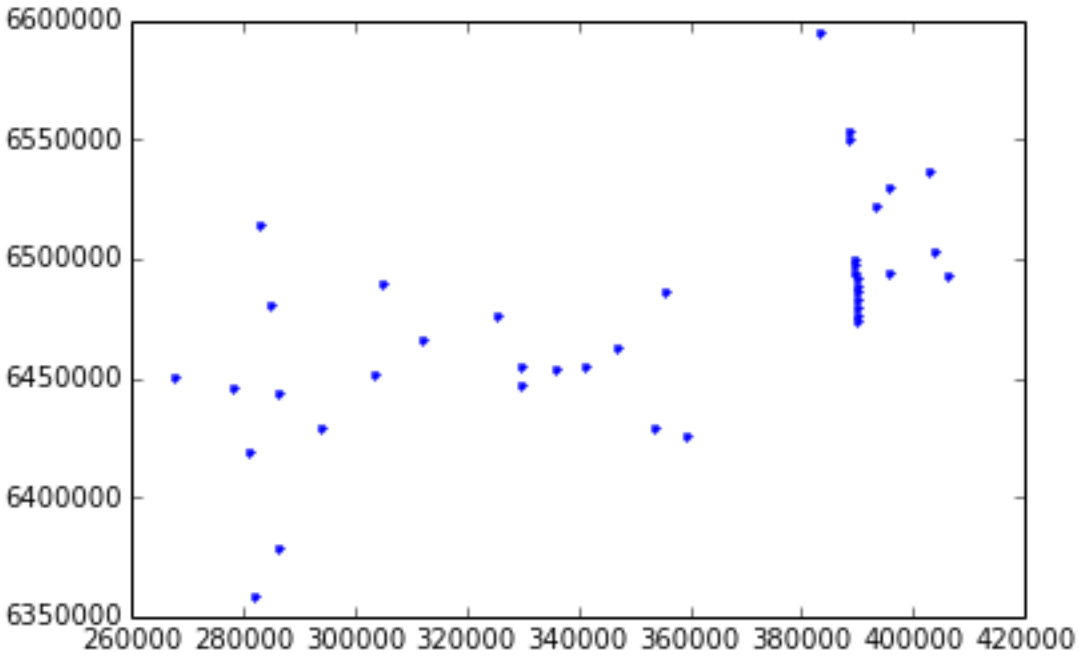
The functionality for orientation data sets is very similar to the point data sets as the main functionality is really for sorting and adapting parameters according to location and formation, and not so much for actual operations that affect the vectorial information (althoguh it might be interesting to include a vector-specific upscaling/ averaging, etc. - but this is not implemented to date).

Loading a data set is exactly as before, and the plotting commands now show the location of orientation data:

```
# only required during testing stage
reload(struct_data)
# use example data
fol = struct_data.Struct3DFoliations(filename = r'../data/wt_Foliations.csv')

fol.plot_3D()
```




```
fol.plot_plane(formation_names = 'Yarragadee_Fm')
```


6600000
6550000
6500000
6450000
6400000
6350000
260000
280000
300000
320000
340000
360000
380000
400000
420000

CHAPTER 4

Creating an irregular mesh from Geomodeller for SHEMAT simulations

Regular meshes can be exported directly from within the Geomodeller GUI. However, in many cases, a more flexible solution is required, for example to:

- update a mesh automatically (without using the GUI), or
- create an irregular mesh with refined regions

These steps can easily be performed with a set of Python scripts and C programs that access Geomodellers funcionality through the API.

The main funcionality required here is combined in the Python package `pygeomod`. Two main packages are required: `geogrid.py` is the most recent development and contains a (relatively general) class definition for rectangular grids in general, with the link to Geomodeller in particular. The package ´geomodeller_xml_obj.py‘ contains methods to access and modify information stored in the Geomodeller xml Project-files. This functionality can be used, for example, to change geological input parameters (e.g. dip of a fault) directly from the Python script.

A note on installation:

The most tricky part is to get the API properly installed, all libraries linked, and compiled on a system. On esim39, the required library path settings are defined in

`adjust_to_jni.sh`

Another important point (for now, should be fixed at some stage...) is that the shared object `libgeomod.so` has to be located in the current directory... time to write a proper make file, but to date that's the stage the project is in.

We will first start here with an example for the generation of an rectilinear refined mesh for a simulation with SHEMAT.

```
# first step: import standard libraries and set pylab for plotting functionalities
%pylab inline
import numpy as np
import matplotlib.pyplot as plt
import sys, os
```

```
Welcome to pylab, a matplotlib-based Python environment [backend: module://IPython.zmq.pylab.backend_
For more information, type 'help(pylab)'.
```

```
# Add path to pygeomod and import module (note: this is only required because it can't be installed p
sys.path.append(r'/home/jni/git/tmp/pygeomod_tmp')
import geogrid
```

4.1 Creating a regular grid

The geogrid module contains a variety of methods to generate grids. In combinbation wtih Geomodeller, the easiest thing to do is to create a regular mesh from a Geomodeller project:

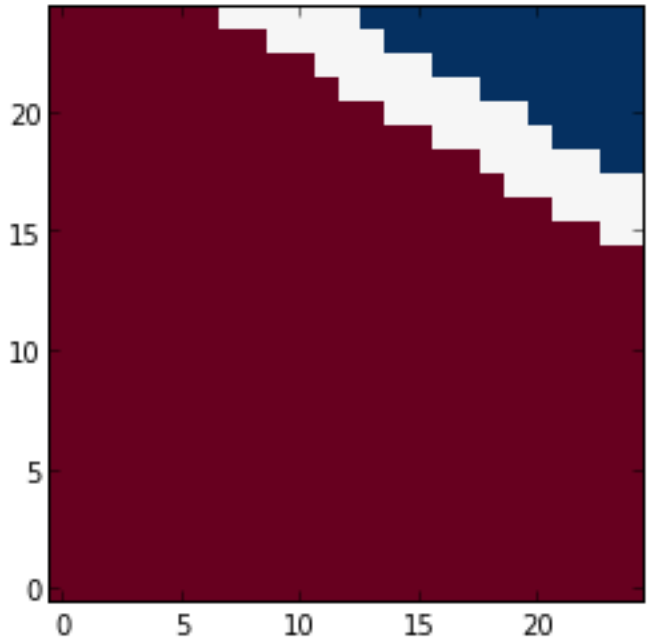
```
# Define path to geomodeller model file:
geomodel = r'/home/jni/git/tmp/geomuce/gemuce_tmp/examples/simple_three_layer/simple_three_layer.xml'

reload(geogrid) # only required for development stage - can be removed afterwards
# Now: define a GeoGrid object:
G1 = geogrid.GeoGrid()
# and set the boundaries/ model extent according to the Geomodeller model:
G1.get_dimensions_from_geomodeller_xml_project(geomodel)

# and create a regular grid for a defined number of cells in each direction:
nx = 25
ny = 2
nz = 25
G1.define_regular_grid(nx, ny, nz)

# ...and, finally, update the grid properties on the base of the Geomodeller model:
G1.update_from_geomodeller_project(geomodel)
```

The grid is stored in the object variable `G1.grid` as a numpy array.

```
type(G1.grid)

numpy.ndarray
```

So the grid can directly be used to create slices, plots, further caluclations, etc. However, a lot of functionality is alread implemented in the geogrid package. For example, slice plots through the model can simply be generated with:

```
G1.plot_section('y', colorbar=False, cmap='RdBu') # more plotting options possible, generally follow
```



It is also possible to export the model directly to VTK - however, this requires an installation of the pyevtk package which is not installed on esim for now:

```
G1.export_to_vtk()
```

```
---------------------------------------------------------------------------
ImportError                               Traceback (most recent call last)

<ipython-input-71-972ad06a1420> in <module>()
----> 1 G1.export_to_vtk()


/home/jni/git/tmp/pygeomod_tmp/geogrid.py in export_to_vtk(self, vtk_filename, real_coords, **kwds)
    327         grid = kwds.get("grid", self.grid)
    328         var_name = kwds.get("var_name", "Geology")
--> 329         from evtk.hl import gridToVTK
    330         # define coordinates
    331         x = np.zeros(self.nx + 1)


ImportError: No module named evtk.hl
```

4.2 Rectilinear grids

Creating a rectilinear grid requires only that the cell spacings are explicitly defined. Everything else is exactly the same as before. Note that it is (at the moment) your responsibility to assing proper spacings - if you go beyond the bounds of the Geomodel, the function will not crash, but return the standard Geomodeller "out" value (usually the number of stratigraphic units + 1).

One way to create meshes in the correct range is, of course, to use the extent of the Geomodel, determined with the function:

```
reload(geogrid) # only required for development stage - can be removed afterwards
# Now: define a GeoGrid object:
G1 = geogrid.GeoGrid()
# and set the boundaries/ model extent according to the Geomodeller model:
G1.get_dimensions_from_geomodeller_xml_project(geomodel)

# The extent of the Geomodeller model can be obtained with:
G1.xmin, G1.xmax
```

```
(0, 1000)
```

```
# and the extent with:
G1.extent_x
```

```
1000
```

Let's be a bit fancy and create the horizontal (x,y) grid with a core region of high refinement and increasing mesh sizes towards the boundary. First, we define the geometry:

```
core_region = 100 # m
# define cell width in core region:
cell_width_core = 25 # m
del_core = np.ones(int(core_region / cell_width_core)) * cell_width_core
# and the number of cells in the boundary regions (the innermost cell has the size of the core cells,
n_boundary = 10
```

```
# now determine the boundary width on both sides of the core region:
width_boundary_x = (G1.extent_x - core_region) / 2.
width_boundary_y = (G1.extent_y - core_region) / 2.
```

A little helper function in the `geogrid` package can be used to determine an optimal cell increase factor for the boundary cells for a given width an a number of cells, and a fixed inner cell width which we take as the width of the core cells for a neat transition:

```
dx_boundary = geogrid.optimial_cell_increase(cell_width_core, n_boundary, width_boundary_x)
dy_boundary = geogrid.optimial_cell_increase(cell_width_core, n_boundary, width_boundary_y)
```

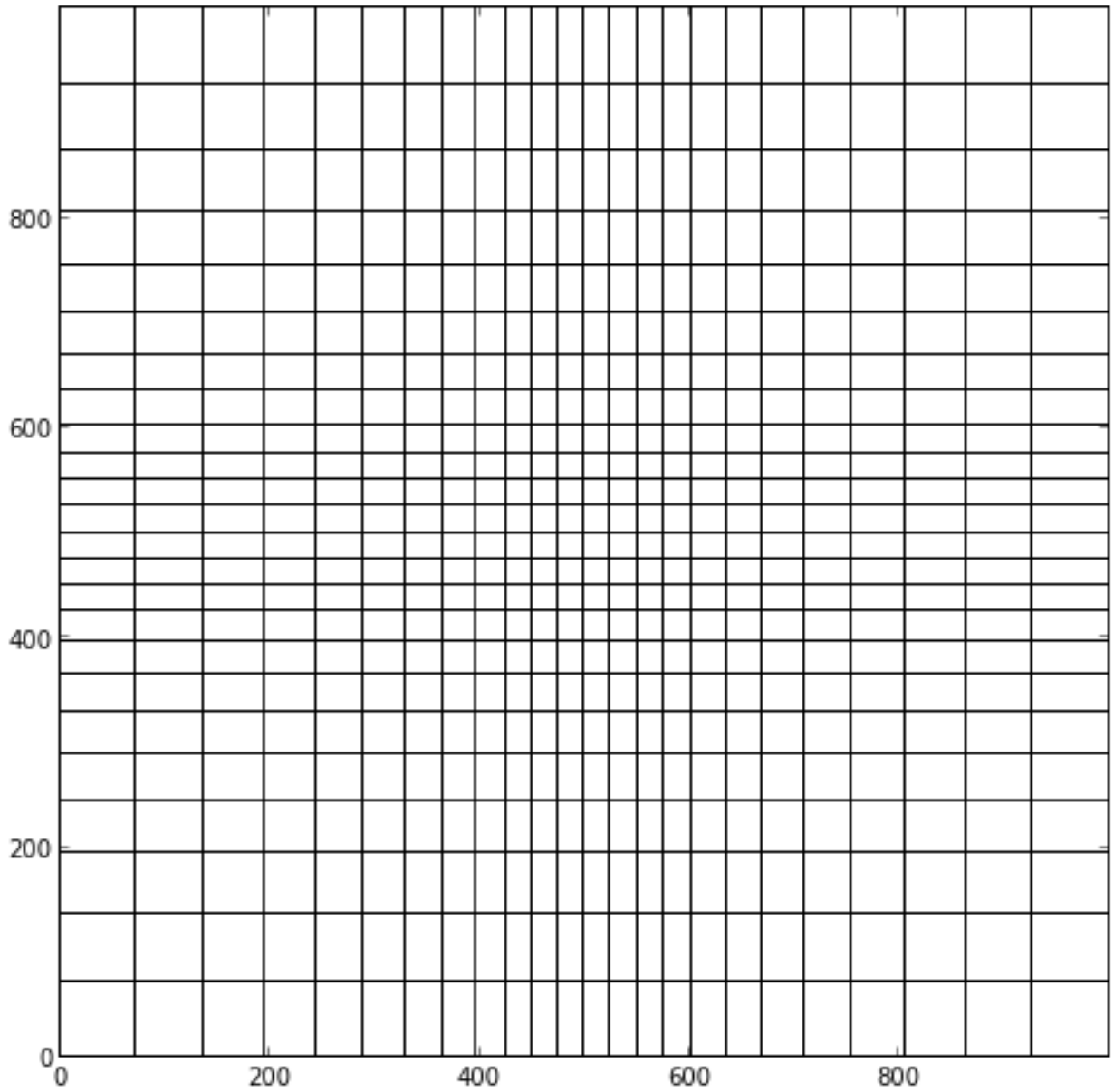
We now simply combine the boundary and core cells for the complete discretisation array:

```
delx = np.concatenate((dx_boundary[::-1], del_core, dx_boundary)) # first array reversed from large t
dely = np.concatenate((dy_boundary[::-1], del_core, dy_boundary))
```

A plot of the grid:

```
fig = plt.figure(figsize = (8,8))
ax = fig.add_subplot(111)
for dx in np.cumsum(delx):
    ax.axvline(dx, color = 'k')
for dy in np.cumsum(dely):
    ax.axhline(dy, color = 'k')

ax.set_xlim((0,sum(delx)))
ax.set_ylim((0,sum(dely)))
```

```
(0, 999.99999999999864)
```



In z-direction we will create a regular mesh:

```
nz = 20
delz = np.ones(nz) * G1.extent_z / nz
```

Ok, back to the geogrid package: we now assign the cell discretisation arrays to the geogrid object and populate the grid with geology ids determined from the Geomodeller model:

```
G1.define_irregular_grid(delx, dely, delz)
G1.update_from_geomodeller_project(geomodel)

G1.grid

array([[[ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        ...,
```

```
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.]],

       [[ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        ...,
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.]],

       [[ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        ...,
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.],
        [ 1.,  1.,  1., ...,  1.,  1.,  1.]],

       ...,
       [[ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        ...,
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.]],

       [[ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        ...,
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.]],

       [[ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        ...,
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.],
        [ 1.,  1.,  1., ...,  3.,  3.,  3.]]])
```

The simple plotting functions don't work for irregular/ rectilinear grids at to date (as imshow can only plot regular grids). Export to VTK would work, in principle.

What we can do, however, is create a SHEMAT nml file (for the old SHEMAT version) directly from the grid:

```
sys.path.append(r'/home/jni/git/tmp/PySHEMAT/PySHEMAT-master')
import PySHEMAT

S1 = PySHEMAT.Shemat_file(from_geogrid = G1, nml_filename = 'updated_model.nml')

create empty file
```

CHAPTER 5

pygeomod

5.1 pygeomod package

5.1.1 Submodules

5.1.2 pygeomod.geogrid module

5.1.3 pygeomod.geomodeller_xml_obj module

5.1.4 pygeomod.struct_data module

5.1.5 Module contents

Module initialisation for pygeomod Created on 21/03/2014

@author: Florian

CHAPTER 6

Indices and tables

- *genindex*
- *modindex*
- *search*

p

P