WP2 workshop Busselton

Contributions by Mark Jessell & (*map2loop* & *dh2loop*), Ranee Joshi (*map2loop, multiscale* & *dh2loop*), Vitaliy Ogarko (*map2model* code), Kavitha Madiah (*dh2loop*), Mark Lindsay (*Geomodeller* export code), Miguel de la Varga (*gempy* export code), Lachlan Grose (*LoopStructural* export code), Nuwan Suriyaarachchi (workshop debugging).

1. Installation
2. Web catalogue services for geophysics (Alex Ip)
3. Exporting and text parsing of drillhole data (Ranee Joshi/Kavitha Madaiah)
4. Multiscale modelling (Ranee Joshi)
5. Examine data in QGIS and via notebook (Mark Jessell)
6. Building your first model using *LoopStructural* (Mark Jessell/Lachlan Grose)
7. Building a model using *gempy* (Mark Jessell/Miguel de la Varga)
8. Building a model using *Geomodeller* (Mark Jessell/Mark Lindsay)

Where available, the best predictor for the 3D geology of the subsurface is often the information contained in a geological map. This information falls into three categories of geometric data: positional data such as the position of faults, intrusive and stratigraphic contacts; topological data, such as the age relationships of faults and stratigraphic units, and gradient data, such as the dips of contacts or faults. In a 3D workflow, we combine all of these direct observations with conceptual information, including assumptions regarding the subsurface extent of faults and plutons to provide sufficient constraints to build a 3D geological model. Typically these conceptual assumptions are communicated via geological cross-sections supplied with the map, however these are often based on limited or no data.

In the Loop Consortium we are developing algorithms that allow us to automatically deconstruct a geological map to recover the necessary positional, topological and gradient data as inputs to different 3D geological modelling codes.

This automation provides significant advantages as it:

- significantly reduces the time to first prototype models;
- clearly separates the primary data from the data reduction steps and conceptual constraints and
- provides a homogenous pathway to sensitivity analysis, Uncertainty Quantification and Value of Information studies from the original data, rather than a subset extracted for modelling purposes.

In this proof of concept, we use the 2016 1:500 000 State interpreted bedrock geology map of Western Australia (GSWA, 2016), the Western Australian Field Observation database (WAROX) and SRTM data supplied as an online service by Geoscience Australia as sources of the data needed to build a first-pass model of the region around the Rocklea Dome and Turner Syncline in the Hamersley Region of Western Australia. The area consists of upright refolded folds of Archean and

Proterozoic stratigraphy overlying an Archean basement cut by over 50 northwest–southeast trending faults that form a part of the Nanjilgardy Fault System.

1. Installation

1.1. Install ***VirtualBox*** for Windows/linux/MacOS Host

from here:

https://www.oracle.com/virtualization/virtualbox/

1.2. Copy test directory (~30GB) to your hard disk

Start up VirtualBox and select the Machine->Add menu item and select the ***M2L/M2L.vbox*** file you have copied over which is in the WP2 folder.

Click on the green arrow (start) 

This should start up a full Ubuntu Linux environment with all the python libraries all set up

1.3. Start Jupyter notebook

Now you can click on the terminal icon  to open up a bash shell and type:

```
>>jupyter-notebook pylibs
```

and click on the links to get to map2loop/notebooks

This should start a browser with a directory list. Please click on the **0 Test.ipynb** link and run the notebook by clicking on the Run icon twice. This should display a part of a geology map in the cell output below the code.

> Although not part of this workshop, the virtual machine also includes functional versions of notebooks that demonstrate other libraries and tools :
>
> **gempy + LoopStructural + SimPEG inversion + Alex Ip (GA) geophysics data utilities + Tomoslow + striplog + apsg + mplstereonet + 3-point problem solver + minQ XYZ from drillhole surveys**
>
> These all run in the conda environment called **loop**, except SimPEG that runs in an environment called **simpege** (type in **conda activate simpege** get change environments).

2. Web catalogue services for geophysics (Courtesy of Alex Ip)

2.1. Catalogue Service for the Web (CSW) server

Now we will go the Notebook in the **geophys_utils/examples** directory called **1_CSW_data_Discovery.ipynb** This notebook, developed by Alx Ip (then at GA), demonstrates where we would like to go in terms of accessing online data. It uses Catalogue Service for the Web (CSW) server that supplies access metadata for different (geophysical) GA data delivery services using a Geonetwork server with a keyword search facility. The default keywords are "NCI,grid,national geophysical compilation" but feel free to modify these terms to try other possibilities, or change Cells 5 & 6 to search for title words instead of keywords etc.

Ideally, there will be one CSW server for Loop that will maintain an international directory of input data for Loop, but even country-level and/or State/Province-level servers would be great!

2.2. Download and imaging data

Next we will go the Notebook in the **geophys_utils/examples** directory called **2_geophys_netcdf_grid_utils_demo.ipynb** This notebook, also developed by Alex Ip, demonstrated the direct download of gravity data from the GA servers.

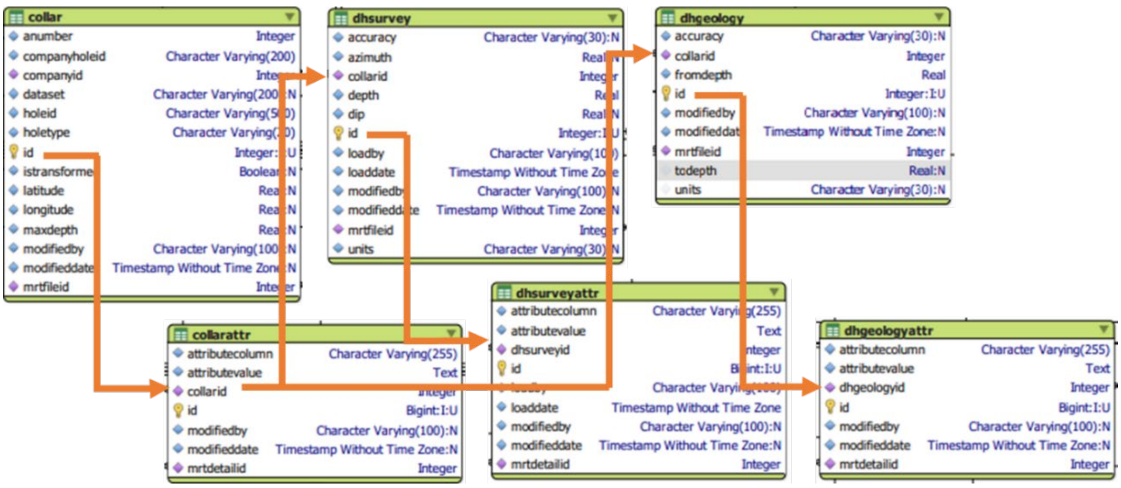
2.3. Download and imaging geology data

We will do more of this later on, but in the map2loop notebooks area you can see some examples of loading geological data at **98. Geological data input_tests.ipynb**

3. Exporting and text parsing of drillhole data (dh2loop *beta*)

Next, we will look into exporting and text parsing of drillhole data. We will go to the notebook in **dh2loop/notebooks** directory called **3_Exporting_and_Text_Parsing_of_Drillhole_Data.ipynb**.

This notebook demonstrates exporting collar, survey and lithology data from the GSWA WAMEX database and further data processing of these tables. WAMEX contains mineral exploration reports and data that have lapsed the period of confidentiality. Online access is free of charge at: https://www.dmp.wa.gov.au/Geological-Survey/Mineral-exploration-Reports-1401.aspx



*Simplified schema of the GSWA WAMEX database showing the links between the **collar, collattr, dhsurvey, dhsurveyattr, dhgeology and dhgeologyattr** tables. It could be noticed that each main table (i.e **collar**) is accompanied by a supplementary table (**collarattr**). The main table contains the unique id for a corresponding location (which could be an X,Y position, dowhhole depth or interval. The supplementary table contains the corresponding attribute column and attribute information for each id in the main table.*

3.1. Set-up

For this workshop, we will be accessing the data through a local subset copy instead of connecting directly to the Loop Postgis WAMEX database. For this, we will first define the paths to the database files and the thesauri to be used to extract and decode the collar, survey and lithology files. This is done in a file called ***dh2l_config.py.***

3.2. Exporting collar data

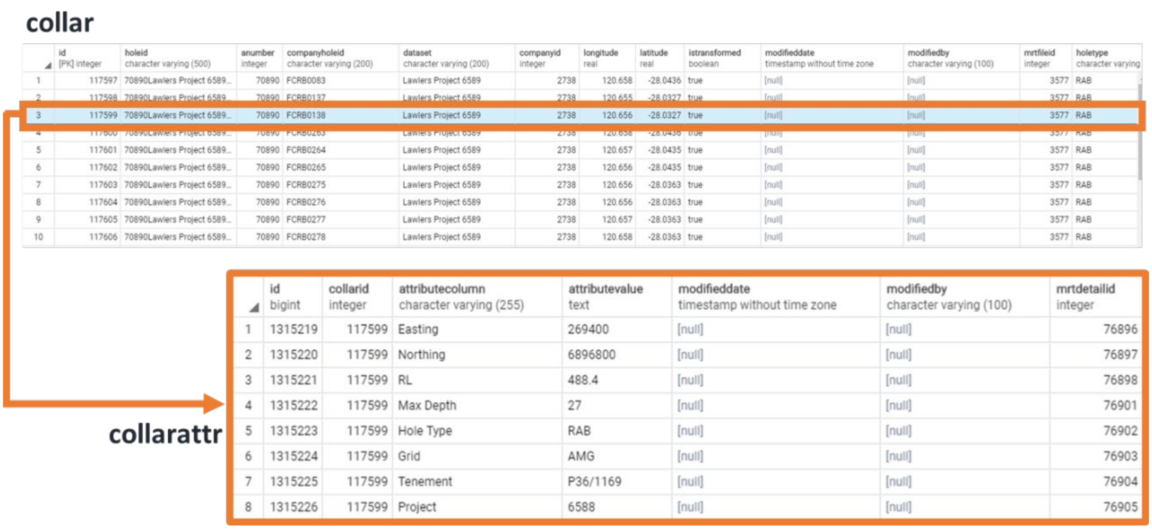
The database is structured to be in pairs of tables. The ***collar*** table contains the unique id for the hole and its location (longitude and latitude). The ***collarattr*** table contains the different attributes (**RL, MaxDepth**, etc) and their corresponding values. Since the attributes we would like to extract include **RL** and **MaxDepth** which are not standard terms across different holes, we will use an **RL_MaxDepth Thesaurus** to filter and select the rows of interest. The thesaurus consists of 357 terms for **RL** and 154 terms for **MaxDepth**. This thesaurus can be updated by simply adding new entries in the table (csv file for this case). The X,Y coordinates are also computed by projecting to Map Grid of Australia (GDA 94/MGA) (currently at Zone 50, but have to do it for different zones) .

The code also includes data quality checks such as dealing with:

- Multiple **MaxDepth** Values > Takes largest value
- Multiple **RL** Values>Takes value with most decimal places (considering taking from dtm in the future)

This outputs a table with:

- CollarID
- HoleId
- Longitude
- Latitude
- RL
- MaxDepth
- X
- Y



collar

	id [PK] integer	holeid character varying (500)	anumber integer	companyholeid character varying (200)	dataset character varying (200)	companyid integer	longitude real	latitude real	istransformed boolean	modifieddate timestamp without time zone	modifiedby character varying (100)	mrtfileid integer	holetype character varying
1	117597	70890Lawlers Project 6589...	70890	FCRB0083	Lawlers Project 6589	2738	120.658	-28.0436	true	[null]	[null]	3577	RAB
2	117598	70890Lawlers Project 6589...	70890	FCRB0137	Lawlers Project 6589	2738	120.655	-28.0327	true	[null]	[null]	3577	RAB
3	117599	70890Lawlers Project 6589...	70890	FCRB0138	Lawlers Project 6589	2738	120.656	-28.0327	true	[null]	[null]	3577	RAB
4	117600	70890Lawlers Project 6589...	70890	FCRB0263	Lawlers Project 6589	2738	120.658	-28.0436	true	[null]	[null]	3577	RAB
5	117601	70890Lawlers Project 6589...	70890	FCRB0264	Lawlers Project 6589	2738	120.657	-28.0435	true	[null]	[null]	3577	RAB
6	117602	70890Lawlers Project 6589...	70890	FCRB0265	Lawlers Project 6589	2738	120.656	-28.0435	true	[null]	[null]	3577	RAB
7	117603	70890Lawlers Project 6589...	70890	FCRB0275	Lawlers Project 6589	2738	120.656	-28.0363	true	[null]	[null]	3577	RAB
8	117604	70890Lawlers Project 6589...	70890	FCRB0276	Lawlers Project 6589	2738	120.656	-28.0363	true	[null]	[null]	3577	RAB
9	117605	70890Lawlers Project 6589...	70890	FCRB0277	Lawlers Project 6589	2738	120.657	-28.0363	true	[null]	[null]	3577	RAB
10	117606	70890Lawlers Project 6589...	70890	FCRB0278	Lawlers Project 6589	2738	120.658	-28.0363	true	[null]	[null]	3577	RAB

collarattr

	id bigint	collarid integer	attributecolumn character varying (255)	attributevalue text	modifieddate timestamp without time zone	modifiedby character varying (100)	mrtdetailid integer
1	1315219	117599	Easting	269400	[null]	[null]	76896
2	1315220	117599	Northing	6896800	[null]	[null]	76897
3	1315221	117599	RL	488.4	[null]	[null]	76898
4	1315222	117599	Max Depth	27	[null]	[null]	76901
5	1315223	117599	Hole Type	RAB	[null]	[null]	76902
6	1315224	117599	Grid	AMG	[null]	[null]	76903
7	1315225	117599	Tenement	P36/1169	[null]	[null]	76904
8	1315226	117599	Project	6588	[null]	[null]	76905

Link between collar and collarattr table.

3.3. Exporting survey data

The ***dhsurvey*** table contains a unique id (***id***) for a particular depth (***depth***) in a hole (***collarid***). It does contain a ***dip*** and ***azimuth*** column, however, most of the time this is empty.

To extract the azimuth and dip values, we use the ***dhsurveyattr*** table. This time, we use an **Azimuth_Dip Thesaurus** to filter and select the rows of interest. The thesaurus consists of 19 terms for azimuth and 8 terms for dip. The X, Y, Z location for each depth is computed using Minimum curvature method.

The code also includes data quality checks such as dealing with:

- Negative depth values> Takes absolute value
- Null/Non-numeric depth values> Does not include
- Exceeds MaxDepth (to be corrected)
- Azimuth values greater than 360> Does not include
- Dip values less than -90 or greater than 90> Does not include
- Dip values from 0 to 90 (to be corrected)

This outputs a table with:

- CollarID
- Depth
- Azimuth
- Dip
- X
- Y
- Z

dhsurvey

	id integer	collarid integer	depth real	units character varying (30)	accuracy character varying (30)	loaddate timestamp without time zone	loadby character varying (100)	modifieddate timestamp without time zone	modifiedby character varying (100)	mrtfileid integer	dip real	azimuth real
1	260889	117599	27	[null]	[null]	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	3587	[null]	[null]

dhsurveyattr

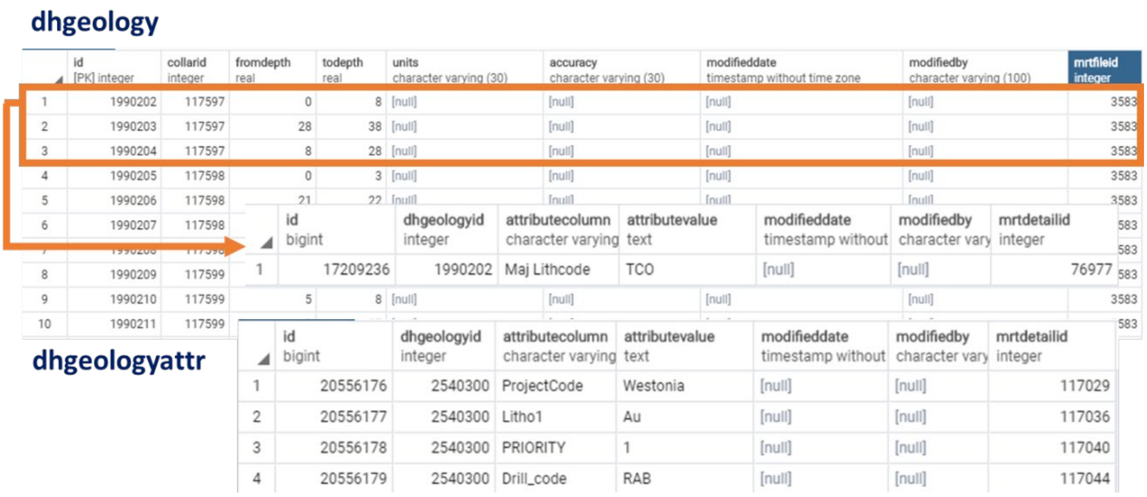
	id bigint	dhsurveyid integer	attributecolumn character varying (255)	attributevalue text	loaddate timestamp without time zone	loadby character varying (100)	modifieddate timestamp without time zone	modifiedby character varying (100)	mrtdetailid integer

	id [PK] bigint	dhsurveyid integer	attributecolumn character varying (255)	attributevalue text	loaddate timestamp without time zone	loadby character varying (100)	modifieddate timestamp without time zone	modifiedby character varying (100)	mrtdetailid integer
1	1600987	260887	Azimuth	270	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77012
2	1600988	260887	AMG Azimuth	270	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77013
3	1600989	260887	Dip	-60	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77014
4	1600990	260888	Azimuth	270	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77012
5	1600991	260888	AMG Azimuth	270	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77013
6	1600992	260888	Dip	-60	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77014
7	1600993	260889	Azimuth	270	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77012
8	1600994	260889	AMG Azimuth	270	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77013
9	1600995	260889	Dip	-60	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77014
10	1600996	260890	Azimuth	270	2011-05-19 12:40:43.88	MIGSDCL	[null]	[null]	77012

Link between dhsurvey and dhsurveyattr table.

3.4. Exporting lithology data

The ***dhgeology*** table contains a unique id (***id***) for a particular from-to interval (***fromdepth, todepth***) in a hole (***collarid***). Extracting lithology has a bit more steps as the database usually contains rock codes. Using ***dhgeologyyattr*** table and a **Litho Thesaurus** (Thesaurus 1, 48 terms). Thesaurus 1 is used to retrieve respective rock codes (**Company_Litho_Code**: i.e. MBH, Mb, ABM). Decoding the Company_Litho_codes requires integrating the **logging protocols (Thesaurus 2)** used by each company to decode these rock codes (i.e., High-Mg basalt, mafic rock after basalt, basalt).



*Link between **dhgeology** and **dhgeologyattr** table.*

To standardize the rock codes retrieved, the script developed in CET which uses the FuzzyWuzzy algorithm was applied. FuzzyWuzzy is a library of Python which is used for string matching fuzzy string matching is the process of finding strings that matches a given pattern using Levenshtein Distance to calculate the difference between sequences. We use the FuzzyWuzzy algorithm to standardize the lithologies.

Before parsing into FuzzyWuzzy, the decoded rock codes (***Company_Litho***) is inspected and cleaned of symbols, descriptors, ages (***cleanup_dictionary*****).**

The FuzzyWuzzy matching compares the decoded rock codes against a **lithology thesaurus** (Thesaurus 3). The **lithology thesaurus** compiles different rock names from the logs and rock databases. The processor used to do the matching is: fuzz.token_set_ratio. It tokenize strings, but split the tokens into groups: intersection and remainder before comparing. It is not as strict as a using an exact processor but stricter than a partial match. Having a comprehensive thesaurus allows this processor to work effectively.

The pseudocode for the FuzzyWuzzy matching:

```
scores=process.extract(Company_Litho, Litho_Dico, scorer=fuzz.token_set_ratio)
    sc in scores:
        if(sc[score]>bestmatch): #better than previous best match
            bestmatch =  sc[score]
            bestlitho= Litho_Dico[firstword]
            if(sc[litho]==Company_Litho[last]): #bonus for being last word in phrase
                bestmatch=bestmatch*1.01
        elif (sc[score]==bestmatch): #equal to previous best match
            if(sc[0]==words[last]): #bonus for being last word in phrase
                bestlitho=Litho_Dico[firstword]
                bestmatch=bestmatch*1.01
```

The pseudocode shows that we input the Company_Litho and parse it through a **lithology thesaurus** (Thesaurus 3). It takes the score for each iteration and if it is greater than the previous match, it stores the score and the first lithology listed in the corresponding **lithology thesaurus** entry (Thesaurus 3) as ***CET Litho***. A bonus is also added to the score if the Company_Litho's last word matches the thesaurus. Furthermore, if the match is less than a threshold we set (in this case, 80). The ***CET_Litho*** is classified as "unclassified _rock".

In order to link and upscale the drillhole information, a **hierarchical thesaurus** (Thesaurus 4) was also built. The dictionary involved three levels (***Level 1, Level 2, Level 3=CET_Litho***) that would upscale a list of 757 rock names to more general rock groups. For example, "basalt" is upscaled to "mafic_fine-grained crystalline" and further upscaled to "igneous rock".

Thesaurus 4
Hierarchical Thesaurus
Cleanup Dictionary + Thesaurus 3
Fuzzywuzzy matching
Thesaurus 2
Company-based Logging Systems
Thesaurus 1
Attribute column names

companyid	name	collarid	fromdepth	todepth	Company_Code	Company_Litho	CET_Litho	bestlithonum	bestmatch	Level_3	Level 2	Level 1
1311	WCP RESOURCES LTD	3984644	24.00	25.00	MBH	High-Mg basalt	basalt	409	101.0	basalt	mafic_fine-grained crystalline	igneous
1489	BUXTON RESOURCES LTD	904011	28.00	29.00	Mb	mafic rock after basalt	basalt	409	101.0	basalt	mafic_fine-grained crystalline	igneous
1621	KARARA MINING LTD	1233224	347.66	350.95	Mb	basalt	basalt	409	101.0	basalt	mafic_fine-grained crystalline	igneous
3049	GINDALBIE GOLD NL	872864	12.00	28.00	ABM	Metamorphosed high-Mg basalt	basalt	409	101.0	basalt	mafic_fine-grained crystalline	igneous

Upscaling lithological information using a hierarchical dictionary.

The start, midpoint and endpoint X, Y, Z location for each interval is also computed using minimum curvature method.

The code also includes data quality checks such as dealing with:

- Todepth is null> Add 0.1
- Fromdepth>Todepth > Reverse interval
- Exceeds MaxDepth (to be corrected)
- Overlapping intervals (to be corrected)

This outputs a table with:

- CompanyID
- CollarID
- FromDepth
- ToDepth
- Company_Litho Code
- Company_Litho
- Cet_Litho
- Score
- Level_1
- Level_2
- Level_3
- Xbt
- Ybt
- Zbt
- Mxy
- Myt
- Mzt
- Xet
- Yet
- Zet

This can also be exported as a VTK file which allows us to inspect the data visually.
Open Paraview to view the VTK file at **\dh2loop\data\export\DB_Lithology_Export.vtp**

4. Mutiscale modelling

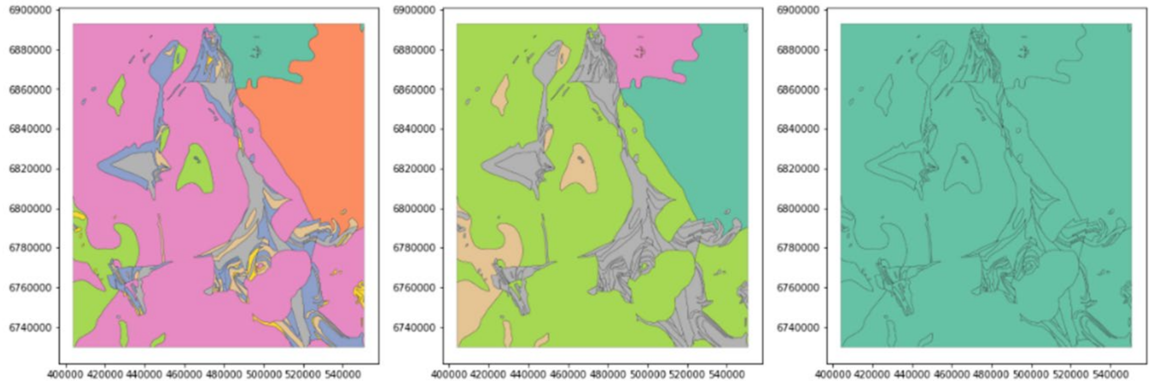
Current geologic modelling allows building a single model at a predetermined scale which is limited to that specific purpose and have an inherent risk to be used to make other assessments. This motivates us to research how to properly subsample geologic data to be able to automatically generate multiscale models that change as we try to answer different geological questions and as we visualize different scales. This notebook will show the proof of concept of hierarchical filters and vector simplification as subsampling parameters.

4.1. Set-up

To start, go to the **map2loop/notebooks** directory and click on **4. Multiscale Modeling .ipynb**

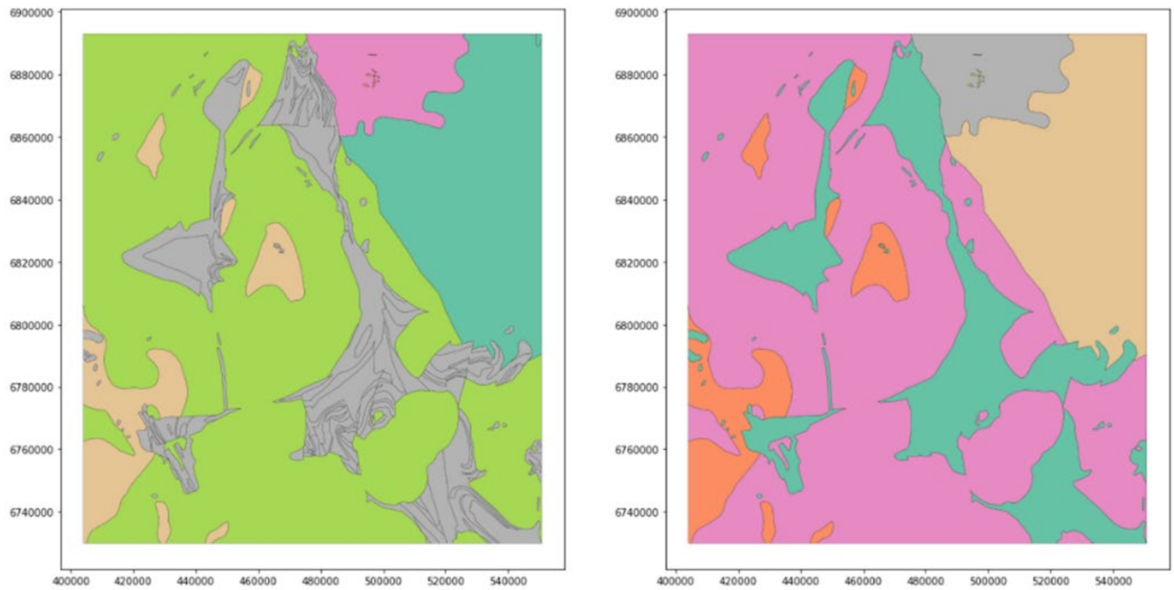
4.2. Hierarchical filters

An important step in subsampling is to identify the features relevant to the model. The answer to this varies on the purpose of our modelling. As a proof-of-concept for categorical filters, we tested filtering using hierarchical filters. This information is readily available in GSWA datasets. Each polygon is linked with the Explanatory Notes System which indicates to what, unit, formation, group and so on it belongs to.



By rock type | By rock unit | By supergroup

Making use of this information, we can automate generalizing the information we would like to keep in the modelling. The vectors are then simplified through aggregation and vertex reduction.



Grouped by Unit Name | Aggregated Vectors

4.3. Vector simplification

Multiple vector simplification algorithms were tested to identify which works well with geological information. The two most common are: Ramer-Douglas- Peucker and Visvalignam-Whyatt algorithms.

The Ramer-Douglas-Peucker (RDP) is the most well-known vector simplification method as it is easy to implement and its recursive nature lends to a hierarchical structure for multi-scale simplification. It is fast and efficient for data compression, eliminating redundant details, reducing the number of points used to represent them (Ramer, 1972; Douglas and Peucker, 1973). The algorithm begins by connecting the endpoints of a line with a trend line. The distance of each vertex to the trend line is then measured perpendicularly. Vertices closer to the line than the tolerance bandwidth error are eliminated. The line is then divided by the vertex farthest from the trend line, which makes two new trend lines. The remaining vertices are measured against these lines, and the process continues until all vertices within the tolerance are eliminated.

The Visvalignam-Whyatt (VW) algorithm is more intuitive, has less perceptible change and preserves shape more precisely. The principle of the algorithm is to select the vertices to delete (the less characteristic ones) rather than choosing the vertices to keep (in the Douglas-Peucker-Ramer algorithm). The selection of vertices to delete is an iterative process, and at each iteration, the triangles formed by three consecutive vertices are computed. If the area of the smallest triangle is smaller than an area tolerance threshold, the middle vertex is deleted, and another iteration starts (Visvalingam and Whyatt, 1990).

It was found that the Visvalignam-Whyatt algorithm manages to keep characteristic points/salient relevant vertices and outline of stratigraphic regions to capture and maintain certain spatial and topological features and remains consistent with the original vector at some level of uncertainty.

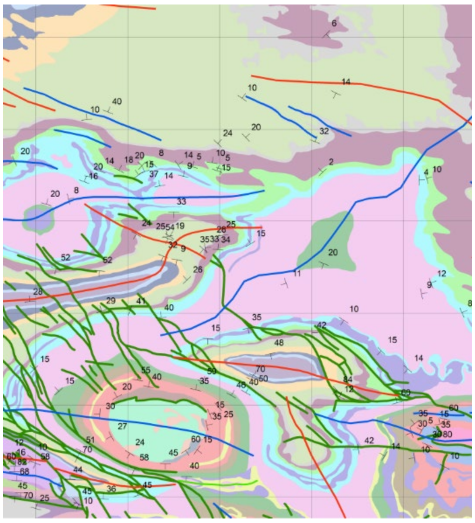
Since the Visvalignam-Whyatt algorithm is more suited for geological information, where the shapes geologists draw actually contain geological information and interpretation, the algorithm has been modified to preserve topological relationship and proximity to adjacent/neighboring polygons by keeping junctions between polygons and planar self-intersections.

The next steps in the proposed subsampling workflow involves identification of topology and stratigraphy, and extraction of contacts and orientation data through Map2Loop. We will look into this in the next notebooks.

5. Building a 3D model using *map2loop*-Examine model input data in QGIS

In order for map2loop to function, it needs three layers (geology polygons; fault/fold axial trace polylines and bedding measurement points). These GIS layers have to have a specific set of attributes. The fault layer can possibly be empty (not tested yet).

First we will examine these three layers in QGIS. Click on the QGIS icon and load the **test_data3** project.



1:500 000 Interpreted bedrock geology of the Rocklea Dome/Turner Syncline region of Western Australia showing the different datasets used to create the 3D model. Red lines are synclines, blue lines are anticlines, green lines are faults and structural symbols are the orientation of bedding. The region shown is approximately defined by the max/min lat/long coordinates [-22,-23,117,118].

Geology polygons:
-a. All polygons are watertight (no gaps or overlaps)
-b. Polygons stop on faults (or at least no gaps/overlaps where they coincide)
-c. Polygons have as attributes:
- -i. Object ID
- -ii. Stratigraphic code
- -iii. Stratigraphic group (multiple codes to a group)
- -iv. One of more fields that describe if sill, if igneous, if volcanic
- -v. Min_age field
- -vi. Max_age field (can be same as Min_age field, and can be simple numerical ordering, larger number is older)

Note on Stratigraphic Hierarchy: Different maps have different terminology for stratigraphic hierarchies, and modelling also packages vary. The ***map2loop*** code assumes a two-level stratigraphic hierarchy, internally referred to as Groups and Codes (or Units), which are equivalent to Series and Formations in ***Geomodeller*** and ***gempy.***

map2loop also allows you to define “supergroups” for some calculations so that these Groups are treated as a single entity (particularly for interpolation purposes). In the long term, a generalised third level to the hierarchy will be applied across the code.

Fault/Fold Axial Trace Polylines:
-a. Faults terminate on other faults (but don’t cross ?)
-b. Faults/Folds have as attributes:
- -i. Object ID
- -ii. Field that specifies if polyline is fault or fold axial trace
- -iii. Field that specifies type of fold axial trace e.g. syncline or anticline)
- -iv. Fault dip (Optional)

Bedding orientations:
-a. Assumes dip/dip direction data
-b. Orientations have as attributes:
- -i. Dip

-ii. Dip Direction
-iii. Field that specifies that measurement is of bedding plane

5.1. Understanding the information content of GIS layers

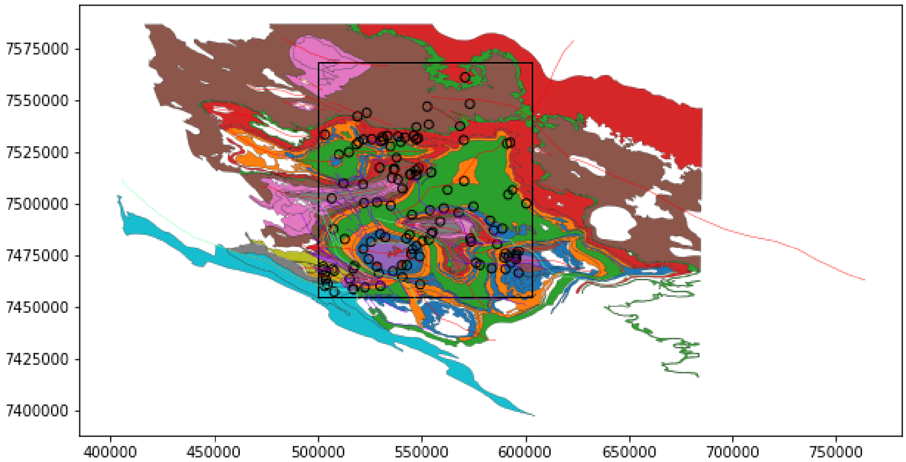
To get things started we need to think a bit more about how information is stored in a GIS layer. Each GIS layer consists of a set of geometric objects (raster, polygons, polylines or points). Shapefiles in particular cannot mix geometry types in a single file (Mapinfo tables can mix geometric objects but not rasters, but we won't use Mapinfo here, however if you do use Mapinfo as inputs they have to be one geometry type per table). For each geometric object (or cluster of objects: multipolygons, multilines, multipoints) we can store a row of information, like in an excel table (in fact you can open a shapefile*.dbf file in excel to see what it looks like).

The geometry information is simply an x,y location (points), or a series of x,y locations (polylines and polygons). These coordinates cannot be seen in the *.dbf file.

Alternatively, we can examine the contents of a GIS layer and its geometries via a python notebook:

Start **0. Data Examination.ipynb** by clicking on its name in the list

This notebook provides graphical and tabular representations of the three GIS layers we will be working with.



The tables allow us to see the field names and contents of the field for each geometry object in the file. For example in the geology layer we can see that the first row refers to a polygon with a **UNITNAME** of **Marra Mamba Iron Formation** with a **CODE** of **A-HAm-cib**.

	OBJECTID_1	OBJECTID	LITHSTRTNO	CODE	UNITNAME	GSWASTATUS	RANK	DESCRIPTN	PARENTCODE	PARENTNAME	ROCKTYPE1
0	1	1	2258	A-HAm-cib	Marra Mamba Iron Formation	Formal	Formation	Chert, banded iron-formation, mudstone, and si...	A-HAL-xci-kd	Hamersley Group, lower	sedimentary other chemical or biochemical

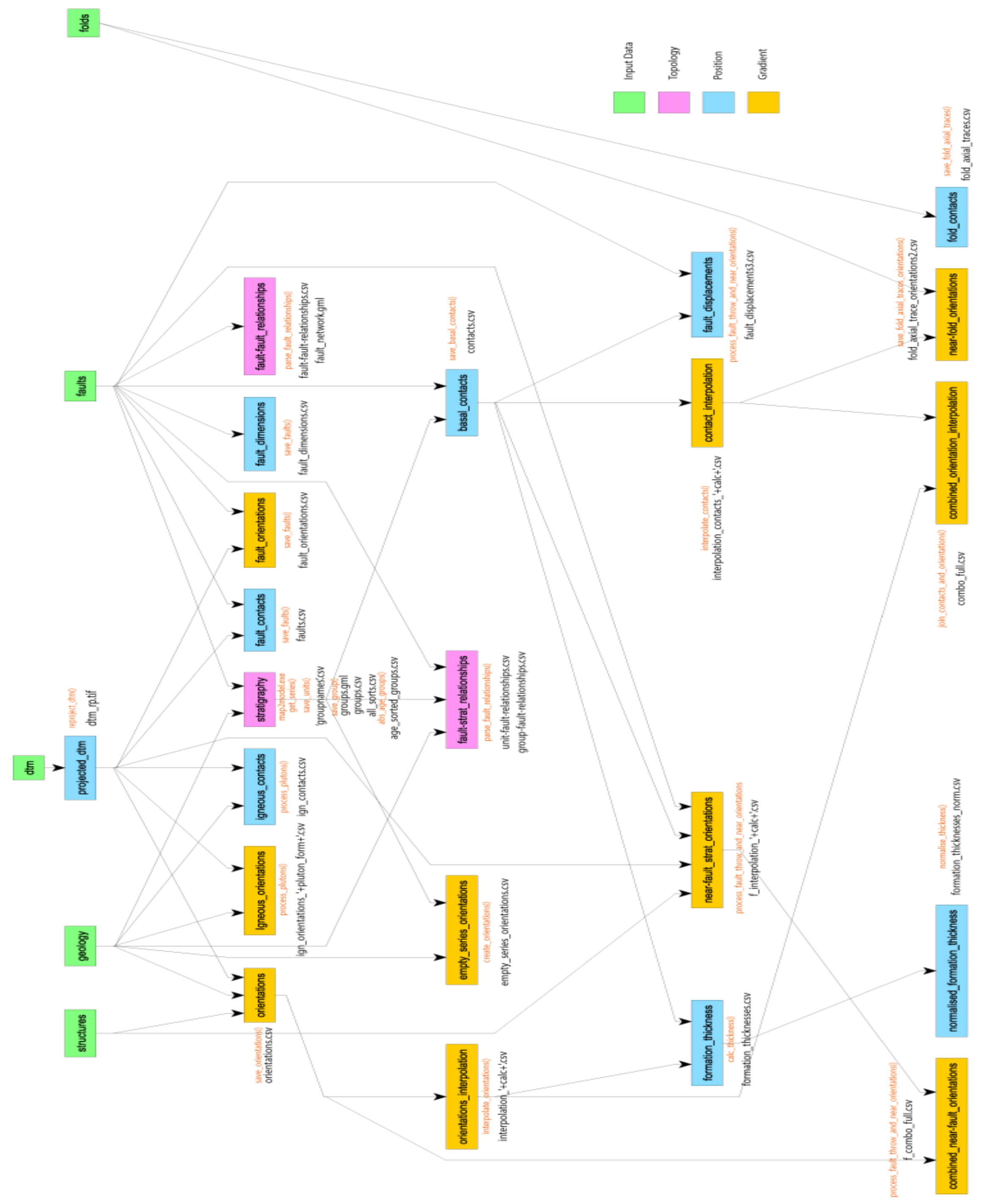
Obviously different maps with have different headings for the different data associated with a polygon, which is why we have the **c_l** (codes and labels) data object defined in the ***config.py*** file. In theory this provides enough flexibility to work with any GIS layer, but I am sure new codes will be needed down the track.

If we look at **OBJECTID** number **47**, we can see that the **ROCKTYPE1** field contains **igneous mafic intrusive** which is the information we use to determine if this is an intrusive unit.

Other things we check for are the text '**sill'** in the **DESCRIPTN** field. In the polyline layer we look for '**Fault'** in the **FEATURE** field and if it is a fold axial trace '**'syn'** in the **Type** field; in the structure layer we look for '**Bedding'** in the **FEATURE** field.

All of these codes and labels can be edited in the ***m2l_config.py*** file, or simply redefined in a cell after the ***m2l_config.py*** file has been read by the notebook.

The *map2loop* codes deconstruct map layers into a set of geometric information that can be broadly classed into **topological**, **positional** and **gradient** information. As it is currently written, the codes interact and extract information in a way that strangely resembles spaghetti:



Input Data
Topology
Position
Gradient
folds
faults
dtm
projected_dtm
dtm_rp.tif
geology
structures
fault-fault_relationships
fault-fault-relationships.csv
fault_network.gml
fault_dimensions
fault_dimensions.csv
fault_orientations
fault_orientations.csv
fault_contacts
faults.csv
stratigraphy
groups.csv
all_sorts.csv
age_sorted_groups.csv
igneous_contacts
ign_contacts.csv
igneous_orientations
orientations
orientations.csv
basal_contacts
contacts.csv
fault-strat_relationships
unit-fault-relationships.csv
group-fault-relationships.csv
empty_series_orientations
empty_series_orientations.csv
orientations_interpolation
fault_displacements
fault_displacements3.csv
contact_interpolation
near-fault_strat_orientations
formation_thickness
formation_thicknesses.csv
fold_contacts
fold_axial_traces.csv
near-fold_orientations
combined_orientation_interpolation
combo_full.csv
normalised_formation_thickness
formation_thicknesses_norm.csv
combined_near-fault_orientations
f_combo_full.csv

6. Building your first model using LoopStructural, Notebook:

1c. All in one Hamersley-LoopStructural.ipynb

The quality of the ***LoopStructural*** and ***gempy*** models is more a function of my ability to use those codes than their inherent qualities. I am sure models that more closely resemble the ***Geomodeller*** ones will be available soon! In addition, the density and distribution of input data is almost certainly not optimal for any of the three modelling packages.

The 3D models have 16 units/formations in 5 Groups/Series, there are 2 intrusions, and after faults less than 5km long are filtered out, there remain 58 limited-extent faults, with 36 fault-fault intersections. A few units/formations in the SW corner of the map are ignored, as there isn't enough orientation data to model them easily. Igneous intrusions are assumed to have domal geometries.

6.1. Timing

map2loop: For reference on an HP ProBook 440 G5 (16 cores, 32 GB Memory), the time needed to calculate all the inputs for ***LoopStructural***/***gempy***/***geomodeller*** takes about 8 minutes. This includes about 3 minutes worth of calculations that are only needed by ***LoopStructural*** (unit thickness) and 2 minutes that are only currently needed by ***LoopStructural***/***Geomodeller*** (fault throw and orientations near faults)

LoopStructural: One the input data has been calculated by ***map2loop***, the additional time to calculate the model (stratigraphic surfaces only) in ***LoopStructural*** is around 6 minutes.

gempy :One the input data has been calculated by ***map2loop***, the additional time to calculate a ***gempy*** model (stratigraphic surfaces only) is about 8 minutes.

Geomodeller : One the input data has been calculated by ***map2loop***, the additional time to calculate the project files for ***Geomodeller*** is 2 minutes, the time to calculate the potential-field in ***Geomodeller*** itself (stratigraphic surface, intrusions and faults) is 2 minutes, and the time for rendering of a low-resolution marching cube 3D surface models 7 minutes so 11 minutes in total.

6.2. Set up

First we need to define some basic parameters which will control the map deconstruction, which is done in a file called ***m2l_config.py.*** For our purposes, we will leave all these parameters as they are, but if you want to try out the code on a different map, or even another part of the same map, the terminology and logic can change, so some edits may be needed.

m2l_config.py **file**

The lines that you may want to vary are shown below, with others that define fixed and derived path names left off of this figure here, but in any case for now just leave it as is.

m2l_config.py

```
#ROI

step_out=0.1  #padding arounf dtm to ensure reprojected dtm covers target area (in degrees)
inset=0     #unused??

minx=500057  #region of interest coordinates in metre-based system (or non-degree system)
maxx=603028
miny=7455348
maxy=7567953
model_top=1200
model_base=-8200

#PATHS

local_paths=True      #flag to use local or WFS source for data inputs (True = local)

test_data_path='../test_data3/'

geology_file='hams2_geol.shp'  #input geology file (if local)
fault_file='GEOS_GEOLOGY_LINEARSTRUCTURE_500K_GSD.shp' #input fault file (if local)
structure_file='hams2_structure.shp' #input bedding orientation file (if local)
mindep_file='mindeps_2018.shp' #input mineral deposit file (if local)

#CRS

src_crs = {'init': 'EPSG:4326'}  # coordinate reference system for imported dtms (geodetic lat/long WGS84)
dst_crs = {'init': 'EPSG:28350'} # coordinate system for data

#CODES AND LABELS these refer to specific fields (codes) in GIS layer or database that contain the info needed for these calcs and text substrings (labels) in the contents of these fields
c_l= {
#Orientations
  "d": "DIP",             #field that contains dip information
  "dd": "DIP_DIR",        #field that contains dip direction information
  "sf": 'FEATURE',        #field that contains information on type of structure
  "bedding": 'Bed',       #text to search for in field defined by sf code to show that this is a bedding measurement
#Stratigraphy
  "g": 'GROUP_',          #field that contains coarser stratigraphic coding
  "c": 'CODE',            #field that contains finer stratigraphic coding
  "ds": 'DESCRIPTN',      #field that contains information about lithology
  "u": 'UNITNAME',        #field that contains alternate stratigraphic coding (not used??)
  "r1": 'ROCKTYPE1',      #field that contains  extra lithology information
  "r2": 'ROCKTYPE2',      #field that contains even more lithology information
  "sill": 'sill',         #text to search for in field defined by ds code to show that this is a sill
  "intrusive": 'intrusive',   #text to search for in field defined by ds code to show that this is an intrusion
  "volcanic": 'volcanic',     #text to search for in field defined by ds code to show that this is an intrusion
#Mineral Deposits
  "msc": 'SITE_CODE',         #field that contains site code of deposit
  "msn": 'SHORT_NAME',        #field that contains short name of deposit
  "mst": 'SITE_TYPE_',        #field that contains site type of deposit
  "mtc": 'TARGET_COM',        #field that contains target commodity of deposit
  "mscm": 'SITE_COMMO',       #field that contains site commodity of deposit
  "mcom": 'COMMODITY_',       #field that contains commodity group of deposit
  "minf": 'Infrastructure',   #text to search for in field defined by mst code that shows site to ignore
#Timing
  "min": 'MIN_AGE_MA',        #field that contains minimum age of unit defined by ccode
  "max": 'MAX_AGE_MA',        #field that contains maximum age of unit defined by ccode
#faults and folds
  "f": 'FEATURE',             #field that contains information on type of structure
  "fault": 'Fault',           #text to search for in field defined by f code to show that this is a fault
  "fold": 'Fold axial trace', #text to search for in field defined by f code to show that this is a fold axial trace
  "n": 'NAME',                #field that contains information on name of fault (not used??)
  "t": 'TYPE',                #field that contains information on type of fold
  "syn": 'syncline',          #text to search for in field defined by t to show that this is a syncline
#ids
  "o": 'OBJECTID',            #field that contains unique id of geometry object
  "gi": 'GEOPNT_ID'           #field that contains unique id of structure point
}

#DECIMATION

orientation_decimate=0  #store every nth orientation (in object order) 0 = save all
contact_decimate=10     #store every nth contact point (in object order) 0 = save all
fault_decimate=5        #store every nth fault point (in object order) 0 = save all
fold_decimate=5         #store every nth fold axial trace point (in object order) 0 = save all

#INTERPOLATION

gridx=50          #x grid dimensions (no of points, not distance) for interpolations
gridy=50          #x grid dimensions (no of points, not distance) for interpolations
scheme='scipy_rbf'    #interpolation scheme
dist_buffer=5         #buffer distance for clipping points by faults (in metres or same units as dst_crs)
intrusion_mode=0      # 1 all intrusions exluded from basal contacts, 0 only sills
use_interpolations=False   # flag to sue interpolated orientations or not.

#ASSUMPTIONS

pluton_dip=45         #surface dip of pluton contacts
pluton_form='domes'   #saucers: \__+_+__/  batholith: +/      \+  domes: /  +  +\  pendant: +\______/+
fault_dip=90          #surface dip of faults
```

This ***m2l_config.py*** also file specifies the directories where different types of files produced by ***map2loop*** will be stored, in this case:

test_data3	all files related to project
test_data3/data	input files assuming local files are processed
test_data3/graph	outputs from map2model c++ code
test_data3/dtm	Digital Terrain Model files
test_data3/output	outputs which will be used by modelling packages
test_data3/tmp	temporary files used during calculations

For a summary of each file produced by ***map2loop***, see **Appendix 1** and to see compact Pseudocode for the ***map2loop*** functions used by these notebooks see **Appendix 2**.

6.3. Topology (strat)

The first stage of the deconstruction of the map involves extracting stratigraphic information from the geology polygons (taking into account the effects of faulting). To do this we run the cells labelled 1a to 1k(in order!). This code:

1a Loads various libraries and defines paths for projection info

1b Loads the ***config.py*** file and creates a bounding box for the model and allows the user to override some parameters

1c Test access to online data (we download the DTM data from an online server, and can optionally download the geological data from a server as well)

1dIf using online data override some parameters with new values (but we are not)

1e Load and display geology map and overlay with the bounding box

1fSave geology polygons in WKT format (**test_data3/tmp/hams2_geol.csv**)

1g1 Load and save mineral occurrences (not used by this notebook) and **1g2** orientation information in WKT format (**test_data3/tmp/hams2_structure.csv**)

1h Load and plot fault/fold axial trace polylines

1i Save fault/fold axial trace polylines in WKT format (**test_data3/tmp/GEOS_GEOLOGY_LINEARSTRUCTURE_500K_GSD.csv**)

1j Save out parameter control file for Vitaliy Ogarko's***map2model***c++ code (**m2m_cpp/Parfile**)

1k Call ***map2model.exe*** binary

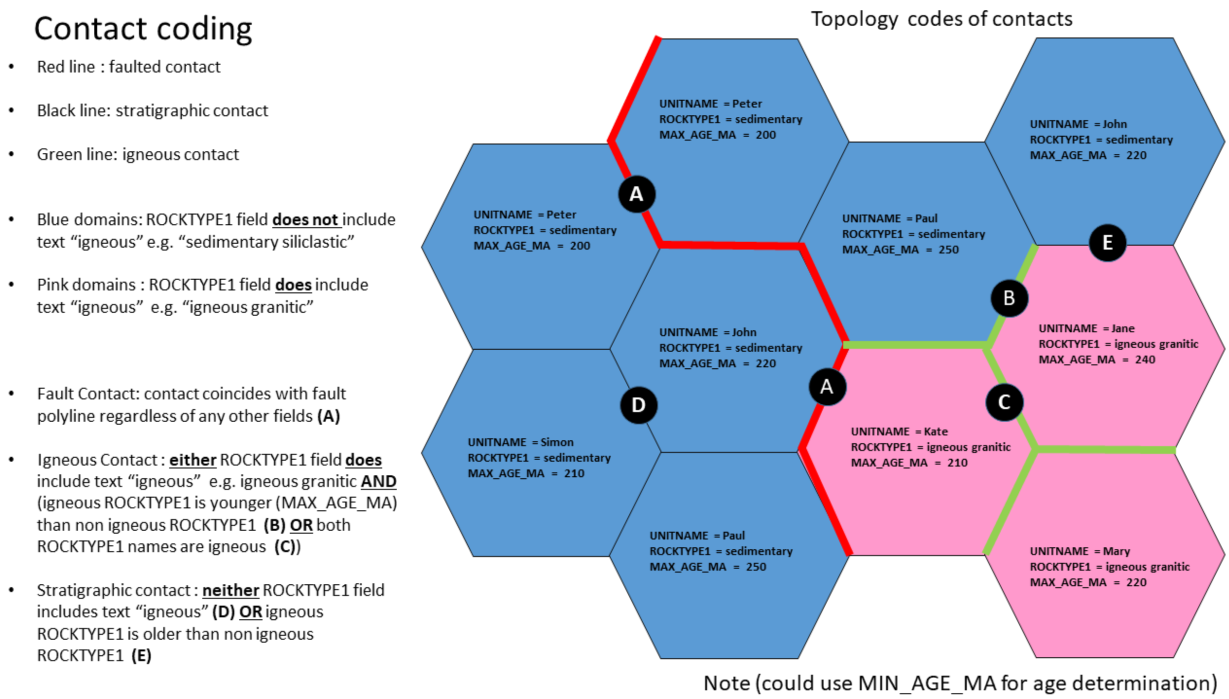
At this stage, the ***map2model*** code will take the geology and fault layers and extracts three types of topological information, based on the type of contact (see figure below):

a) A series of graphs of neighbour relationships between adjacent geology polygons taking into account the relative ages of the two polygons and ignoring boundaries coincident with faults. (**test_data3/graph/graph_*.gml**) that can be read in using ***yEd***.

b) fault-stratigraphy relationships to build a relationship file showing which faults cut which units (**test_data3/graph/unit-fault-intersection.txt**).

c) It also calculates fault-fault relationships to establish which faults are truncated by other faults (**test_data3/graph/fault-fault-intersection.txt**).

d) Finally it does some simple mineral system analysis by determining where different mineral deposits (in this case Fe, Cu, Au) are situated relative to fault, intrusive and stratigraphic contacts (**test_dat3/graph/graph_all_XX.gml**) that can be read in using ***yEd***.



If we run cell **1l** now, it displays a network diagram at the Code/Unit level.

One of the outputs from these calculations is a set of GML files (Graph Modeling Language) which we can visualise in the free-but-not-open-source ***yEd*** visualisation package. To do this start up ***yEd*** and select open and go to the testdat3/graph directory. There are eight GML files:

a. ***graph_all_NONE.gml*** (all relationships displayed)
b. ***graph_fault_NONE.gml*** (only polygons which are adjacent across faults)
c. ***graph_partial_NONE.gml*** (local topology around one specified polygon)
d. ***graph_igneous_NONE.gml*** (only polygons with igneous contacts) and
e. ***graph_strat_NONE.gml*** (stratigraphic (not fault or intrusive) contacts only)
f. ***graph_all_XX.gml*** (three graphs showing topological relationship between deposits (**Fe, Au, Cu**) and different types of contacts)

Open the ***graph_strat_NONE.gml*** file and visualise it by selecting the menu **Layout->OneClick Layout** (needs internet connection) or **Layout->Orthogonal->Classic** if you have no internet connection. This graph shows all of the stratigraphic relationships found locally in the map area. Arrows show the older to younger relationship (double-headed arrows means the relative age is unknown). In this graph the colours show what other types of contact (igneous or fault) also occur for each contact. Feel free to try out other layout types.

It is worth noting that ***yEd*** can produce a graph from a simple 2-column excel file with each pair of entries on a line representing an edge in a graph.

Fault-fault and fault-strat relationships will be calculated later on.

6.4. Position (topo)

Now we run some cells that process some of the positional information needed to build a model:

2aBasic initialisation

2bDefine area of interest in different formats for different calculations

2c Download ~600m SRTM data for area from a Geoscience Australia server (alternatively could get it at ~900m resolution from a server in Hawaii for the whole world). Since the default projection for this data is in WGS84 geodetic lat/long we will reproject this to a metre-based projection system here as well (defined by the EPSG code by the **dst_crs** variable. (**test_data3/dtm/dtm.tif** and **dtm_rp.tif**)

2d Load stratigraphic information

2e Load geology polygons, fault/fold axial trace polylines and orientation data points.

2f Downloaded data contains any polygon or polyline that even partially is within area, so now we clip them to the bounding box, and only retain orientation data related to bedding (**test_data3/tmp/faults_clip.shp, folds_clip.shp, geol_clip.shp, structure_clip.shp**).

2g Combine stratigraphic information from topology analysis with extra information from maps to save out one arbitrarily selected ordering of groups (**test_data3/tmp/groups.csv**).

6.5. Gradient (bedding orientations)

Now we save out the bedding information from the orientation layer:

3a First we get the bedding data (filtering out those with dip=0 as they are probably misleading), and add the Z value from the reprojected dtm, and do a spatial join with the geology polygons to get the stratigraphic code (intrusive polygons are ignored here and will be dealt with later). All polarities are assumed to be normal, and although this could be calculated from comparison of bed dip direction and younging direction, this has not yet been coded. (**test_data3/output//orientations.csv**).

3bNow we cycle through those stratigraphic Groups for which no orientation data is available and add arbitrary data just to keep some modelling systems happy (this is a complete fudge and needs to be redone). (**test_data3/output/empty_series_orientations.csv**)

6.6. Position (strat and igneous contacts, faults)

Now we save out some more positional information:

4a First we identify those parts of each polygon which represent contacts with stratigraphically older polygons (**test_data3/tmp/all_contacts.csv** and**test_data3/tmp/contacts.csv**)

4bNow we remove from the resulting polylines those contacts coincident with faults and save out as a shapefile with no decimation. (**test_data3/tmp/basal_contacts.shp**)

4c Then we decimate the polylines and save out as csv files with x,y,z and stratigraphic Codes (**test_data3/output/contacts4.csv)**

4d Faults are decimated and saved out to a csv file with x,y,z and Fault name. A second file stores the fault length and elliptical extent information, and a third file saves the orientation of each fault. Along each fault offsets in stratigraphic contacts are estimated, local interpolated estimates of stratigraphic contact orientations are calculated and together these provide estimates of true displacement assuming a down-dip slip vector. Simultaneously we store the near-fault stratigraphic orientation data along each side of the fault so that we don't run into problems with empty fault domains. (**test_data3/output/fault_orientations.csv, faults.csv and fault_dimensions.csv**)
4e For all intrusive contacts (except sills) we extract the contact positions and define as inward or outward dipping and normal or reverse polarity depending on the assumed nature of the pluton (saucers, domes, batholiths, pendants). At the moment, all plutons are assumed to have the same form, but this could be modified in the code in the future.(**test_data3/output/ign_contacts.csv**and**ign_orientations_domes.csv**)

6.7. Gradient (interpolated orientations and near-fold axial trace orientations)

Next we process the orientation and basal contact polylines to produce gridded estimates of orientations across the model (we have already actually done this in step 4e but not in a grid pattern). This is a crude approximation of co-kriging, which could be implemented in the future.

5a All bedding orientation data is interpolated to a grid using an RBF function as the l_o,m_o,n_o direction cosines.**(test_data3/tmp/interpolation_scipy_rbf.csv**and**interpolation_*.csv**)

5b All basal contact segments are interpolated to the same grid using an RBF function as the l_c,m_c direction cosines (**test_data3/tmp/raw_contacts.csv, interpolation_contacts_scipy_rbf.csv and interpolation_contacts_*.csv**)

5cThe l_c,m_c,n_o direction cosines are normalised and combined so that the interpolated azimuth respects the local contact orientations and the dips come from the interpolated bedding orientation data. (**test_data3/tmp/combo_full.csv**)

5d Estimate Unit thickness by drawing normal to local contacts and looking for intersections with stratigraphically next higher Unit. If found calculate apparent thickness then estimated true thickness by using local interpolated orientation data. A normalised version for each unit is also calculated by dividing values by median unit thickness. (**test_data3/output/formation_thicknesses.csv, formation_summary_thicknesses.csv** and **formation_thicknesses_norm.csv**)

5e Fold axial traces are decimated and appropriate bed dips are calculated either side of the fold axial trace to reinforce the local fold geometries. (**test_data3/output/fold_axial_traces.csv** and**fold_axial_trace_orientations.csv**)

6.8. Data cleansing

In cell **6a** current position and gradient data are checked to make sure that orientations and contacts don't exist for units that are not in the stratigraphy and v.v. (**test_data3/output/orientations_clean.csv** and **contacts_clean.csv**)

6.9. Topology (fault-strat and fault-fault relationships)

In cell**7a** files saved out by ***map2model*** are reprocessed to produce matrices of fault-Unit, fault-Group relationships and fault-fault relationships for faults exceeding a certain length. Cyclic relationships removed (A truncates B; B truncates C; C truncates A). The fault-fault relationships produce a file ***test_data3/tmp/fault_network.gml*** that can be loaded into ***yEd*** to visualise the graph. The menu **Tools->Centrality Measures** allows the faults to be visualised according to how many other faults are in connection with them (like ordering of a stream network). (**test_data3/output/unit-fault-relationships.csv, group-fault-relationships.csv and fault-fault-relationships.csv**)

6.10. Building your first model with *LoopStructural*

The ***map2loop*** codes have produced 20 output files that together will be used as inputs to the 3D modelling codes. The different modelling systems use different subsets of these output files

All the subsequent cells in this notebook are based on code provided by Lachlan Grose (Monash Uni),and should produce a 3D model of the stratigraphic surfaces using ***LoopStructural*** visualised using the ***lavavu*** library. This ignores faults and intrusions, awaiting further code from Lachie!

7. Building a model using *gempy*

This notebook provides a workflow for the deconstruction of a geological map to provide inputs to the ***gempy*** modelling system (developed by Miguel de la Varga at RWTH Aachen) which then calculates a 3D geology models based on the inputs.

To keep things simple, the notebooks using ***gempy*** only builds the stratigraphic surfaces (i.e. we don't provide them with the information for faults or intrusions).

Each cell in the notebook performs a task related to the extraction of data from the map, the combination of these data to produce new information, or to the model construction itself.

The first cells initialise libraries and load the same config file so the file names and paths are established. Since we have already calculated all the files we need, to build a ***gempy*** model we simply have to call the model construction cells based on code provided by Miguel de la Varga (RWTH Aachen), found in the notebook **1d.Pre-calculated Hamersley-gempy.ipynb** and visualised using the vtk library. This also ignores faults and intrusions for now

8. Building a model using *Geomodeller* (needs Geomodeller!)

The first cells initialise libraries and load the same config file so the file names and paths are established. Since we have already calculated all the files we need, to build a ***Geomodeller*** model we simply have to call the model construction parts of the code, found in the notebook **1e. Pre-calculated Hamersley-Geomodeller.ipynb**. This first produces a taskfile, and we then use this to produce a full input project for ***Geomodeller***, so we then need to load the model from ***Geomodeller*** (see ***Geomodeller*** Manual for details). This model includes 58 faults, and a couple of intrusions (but no sills).

Appendix 1 *map2loop* outputs:

map2loop outputs:

Topology

content	filename	created by	example notebook
Various stratigraphic topology graphs	*/graph/*.gml	map2model cpp code in Notebook 1	1
Group-level stratigraphic relationships	*/tmp/groups.csv	m2l_topology. save_group	1
Formation-level stratigraphic relationships	*/tmp/*_groups.csv	m2l_topology. save_units	1
Summary strat relationships	*/tmp/all_sorts.csv or all_sorts_clean.csv	m2l_topology. save_units	1
Fault-fault relationship table	*/output/fault-fault-relationships.csv	m2l_topology. parse_fault_relationships	1
Fault-fault relationship graph	*/output/fault_network.gml	m2l_topology. parse_fault_relationships	1
Fault-unit relationship table	*/output/unit-fault-relationships.csv	m2l_topology. parse_fault_relationships	1
Fault-group relationship table	*/output/group-fault-relationships.csv	m2l_topology. parse_fault_relationships	1

Digital Terrain Model:

content	filename	created by	example notebook
dtm in lat long wgs83	*/dtm/dtm.tif	m2l_utils.get_dtm	1
georeferenced dtm	*/dtm/dtm_rp.tif	m2l_utils.reproject_dtm	1

Position:

content	filename	created by	example notebook
Contact info with z and formation	*/output/contacts4.csv or contacts_clean.csv	m2l_geometry. save_basal_contacts	1
Fault trace with z	*/output/faults.csv	m2l_geometry. save_faults	1
Basal contacts shapefile	*/tmp/basal_contacts.shp	m2l_geometry. save_basal_no_faults	1
Clipped geology map shapefile	*/tmp/geol_clip.shp	Notebook 1	1
Clipped fault & fold axial traces shapefile	*/tmp/faults_clip.shp	Notebook 1	1
Pluton contacts with z and formation	*/output/ign_contacts.csv	m2l_geometry. process_plutons	1
Local formation thickness estimates	*/output/formation_thicknesses_norm.csv and formation_summary_thickness.csv	m2l_geometry. calc_thickness and normalise_thickness	2
Fault dimensions	*/output/fault_dimensions.csv	m2l_geometry. save_faults	1
Fault displacements	*/output/fault_displacement3.csv	Notebook 6	6

Gradient:

content	filename	created by	example notebook
Bed dip dd data with z and formation	*/output/orientations.csv or orientations_clean.csv	m2l_geometry. save_orientations	1
Extra orientations for empty series	*/output/empty_series_orientations.csv	m2l_geometry. create_orientations	1
Fault orientation with z	*/output/fault_orientations.csv	m2l_geometry. save_faults	1
Clipped orientations shapefile	*/tmp/structure_clip.shp	Notebook 1	1

content	filename	created by	example notebook
Interpolated dip dip direction grid	*/tmp/interpolation_scipy_rbf.csv	m2l_interpolation. interpolate_orientations	1
Interpolated contact vector grid	*/tmp/interpolation_contacts_scipy_rbf.csv	m2l_interpolation. interpolate_contacts	1
Combined interpolation grid	*/tmp/combo_full.csv	m2l_interpolation. join_contacts_and_orientations	1
Pluton contact orientations	*/output/ign_orientations_*.csv	m2l_geometry. process_plutons	1
Near-Fault strat orientations	*/tmp/ex_f_combo_full*.csv	Notebook 6	6
Near-Fold Axial Trace strat orientations	*/output/fold_axial_trace_orientations2*.csv	m2l_geometry. save_fold_axial_traces_orientations	5

loop2model:

content	filename	created by	example notebook
gempy	Notebook creates 3D model itself	m2l_export. loop2gempy	1a, 1d
Basic vtk model thanks to gempy	*/vtk/*.vtp	gempy	1a
Geomodeller	m2l.taskfile	m2l_export. loop2geomodeller	1b, 1e
LoopStructural	Notebook creates 3D model itself	m2l_export. loop2LoopStructural	1c

Appendix 2 Pseudocode for *map2loop* functions

m2l_utils.py functions

_clip_line_poly
Parameters
clip_obj
shp
Pseudocode:
Create a single polygon object for clipping
Create a box for the initial intersection
Get a list of id's for each object that overlaps the bounding box and
subset the data to just those lines
Clip the data - with these data
Return the clipped layer with no null geometry values

_clip_multi_point
Parameters
clip_obj
shp
Pseudocode:
Explode multi-point features when clipping then recreate geom

_clip_multi_poly_line
Parameters
clip_obj
shp
Pseudocode:
Clip multi polygons

_clip_points
Parameters
clip_obj
shp
Pseudocode:
Clip points

clip_shp
Parameters
clip_obj
shp
Pseudocode:
Clip according to geometry type

ddd2dircos
Parameters
dip
dipdir
Pseudocode:
Converts dip, dip direction to three direction cosine arrays(l,m,n)

dircos2ddd
Parameters
l
m
n
Pseudocode:
Converts (l,m,n) direction cosine arrays to dip, dip direction

explode
Parameters
indf
Pseudocode:
for each polygon in multipolygon:
save as polygon in GeoDataFrame

get_dtm
Parameters
maxlat
maxlong
minlat
minlong
path_out
Pseudocode:
getdtm data from GA SRTM server and save as geotiff

get_dtm_bounds
Parameters
dst_crs
path_in
Pseudocode:
get bounds of a dtm from rasterio raster

get_dtm_hawaii
Parameters
maxlat
maxlong
minlat
minlong
path_out
Pseudocode:
getdtm data from Hawaiian SRTM server and save as geotiff

have_access
Parameters
url
hw
Pseudocode:
determine if http access is available for a URL

mod_safe
Parameters
a
b
Pseudocode:
```
if b == 0:
        return 0
else:
        return a modulo b
```

pairs
Parameters
lst
Pseudocode:
convert 1D list into paired list

pts2dircos
Parameters
p1x
p1y
p2x
p2y

Pseudocode:
    Calulate 2D direction cosines from two points

ptsdist
Parameters
p1x
p1y
p2x
p2y
Pseudocode:
    calculate distance between two points

reproject_dtm
Parameters
dst_crs
path_in
path_out
src_crs
Pseudocode:
    reproject raster using rasterio

save_clip_to_bbox
Parameters
dst_crs
geom
maxx
maxy
minx
miny
path
Pseudocode:
    Create polygin from points
    Convert polygon to GeoDataFrame
    Save GeoDatFrame to shapefile

tri_angle
Parameters
p1x
p1y
p2x
p2y
p3x
p3y
Pseudocode:
    Apical angle between three points, first point is at apex

value_from_raster
Parameters
dataset
locations
Pseudocode:
    if point is wthin bounds of raster:
        return closest raster value to point
    else
        return -999

m2l_topology.py functions

abs_age_groups
Parameters
c_l
geol
tmp_path
Pseudocode:
    for each polygon in GeoDataBase:
    if no data in Group field:
        Group data = unit data and replace spaces and hyphens
    else:
        replace spaces and hyphens
        build list of groups and associated info
    for each group:
        calculate max/min ages of Units within that group
    savecvs file with groups sorted by average of max/min age of group

get_series
Parameters
id_label
path_in
Pseudocode:
    load a stratigraphy with Groups from GML file
    for each node:
        if new group:
            add to group list
    return Group list, number of Groups, array of Group names

parse_fault_relationships
Parameters
graph_path
output_path
tmp_path
Pseudocode:
    load unit fault relationships from txt file
    load fault lengths from csv file of faults longer than given length as array
    for all faults in array:
        create unique list of faults
    for all faults from fault relationships file:
        tidy up fault name and save out master fault to csv file
        for every fault in unique list:
            if fault is in unique list and unit is in unit-fault relationships list for this master fault:
                save out '1' to csv file
            else:
                save out '0' to csv file

    load sorted stratigraphy from csv file
    load newly created unit-fault csv file
    for each Group:
        for each Unit:
            if Unit-fault relationship is true:
                Group-fault code = 1
    for each Group:
        for each fault:
            save Group-fault relationship codes to csv

    load fault fault relationships from txt file
    for each fault relationship row:
        make unique master list of faults

    create null Graph
    for each master fault:
        for each secondary fault:
            if master fault:

```
        for each fault:
          if faults being compared as same:
            save out '0' to cvs file
          else:
            for each second order fault for this row:
              if second order fault is in list of long
faults:
                save out '1' to csv file
                add edge to Graph
          if seconday fault found:
            save out '0' to cvs file
  save out Graph to GML file
```

save_faults_wkt

```
  Parameters
  c_l
  fault_file_csv
  sub_lines
  Pseudocode:
    for every polyline in GeoDataBase of polylines:
      save to csv file in WKT format
```

save_geol_wkt

```
  Parameters
  c_l
  geology_file_csv
  sub_geol
  Pseudocode:
    for every polygons in GeoDataBase of polylgons:
      save to csv file in WKT format
```

save_group

```
  Parameters
  c_l
  G
  geol
  glabels
  path_out
  Pseudocode:
    load geology polygons
    load age-sorted Groups
    for every edge in stratigraphy graph (input
parameter):
      if no value for Group in endnodes:
        Group=Code for each empty endnode
      if first endnode younger than second endnode:
        add edge to new graph

    for every edge in copy of new graph:
      for every edge in another copy of new graph:
        if edges in both directions:
          remove one of the edges from new graph

    calculate and save out all topological sorts of new
graph as csv file

    load sorted list of groups from csv file

    for each group:
      load units
      save out combined unit and group information to
csv file
```

save_mindep_wkt

```
  Parameters
  c_l
  mindep_file_csv
  sub_mindep
  Pseudocode:
    for every point in GeoDataBase of points:
      save to csv file in WKT format
```

save_Parfile

```
  Parameters
  c_l
  fault_file_csv
  geology_file_csv
  graph_path
  m2m_cpp_path
  maxx
  maxy
  minx
  miny
  structure_file_csv
  Pseudocode:
    save input parameter file for map2model c++ code
```

save_structure_wkt

```
  Parameters
  c_l
  structure_file_csv
  sub_pts
  Pseudocode:
    for every point in GeoDataBase of points:
      save to csv file in WKT format
```

save_units

```
  Parameters
  G
  glabels
  path_out
  Pseudocode:

    for every Group in Group list (input variable):
      for every node in copy of graph (input parameter):
        if Group node or not part of current Group:
          delete node from copy of graph
      calculate and save to Groupname csv all topological
sorts of Units in current Group
```

m2l_geometry.py Functions

bboxes_intersect

```
  Parameters
  bbox1
  bbox2
  Pseudocode:
    calculate if corner nodes of bounding box fall within
other bounding box
```

calc_thickness

```
  Parameters
  buffer
  max_thickness_allowed
  output_path
```

tmp_path
Pseudocode:
load basal contacts as vectors from csv file
load interpolated bedding orientations from csv file
load basal contacts as geopandas GeoDataFrame of polylines
load sorted stratigraphy from csv file
calculate distance matrix of all orientations to all contacts

for each contact line segment:
if orientations within buffer range to contact:
calculate average of all orientation direction cosines within range
calculate line normal to contact and intersecting its mid-point
for all basal contact polylines:
if polyline Group is one stratigraphically one unit higher:
if contact normal line intersects polyline:
if distance between intersection and contact mid-point less than 2 x buffer:
store info
from list of possible intersections, select one closest to contact mid-point
if closest is less than maximumum allowed thickness:
save thickness and location to csv file

create_basal_contact_orientations
Parameters
c_l
contacts
dist_buffer
dtm
output_path
structures
Pseudocode:
not currently used...

create_orientations
Parameters
c_l
dtm
geology
path_in
path_out
structures
Pseudocode:
load Groups from csv file
for each Group:
for each orientation:
replace null Groups with Code
build list of groups found in orientations

for each Group:
for each polygon from geology layer:
add to list of groups using those found in polygons

for each polygon from geology layer:
build list of Units using those found in polygons

for each Group:
for each polygon from geology layer:
if Group has no orientations and Group is not intrusive:
invent and save orientation that falls within polygon to csv file

extract_poly_coords
Parameters
geom
i
Pseudocode:
if shape is polygon:
extract exterior polygon and interior holes
else if shape is multipolygon:
extract exterior polygons and interior holes

return set of all polygons

normalise_thickness
Parameters
output_path
Pseudocode:
load formation thicknesses from csv file
get list of unique Unit codes
for each unique code:
calculate median and standard deviation of thicknesses for that code
save out info to csv file

old_save_faults
Parameters
c_l
dataset
fault_decimate
fault_dip
fault_min_len
path_fault_orientations
path_faults
Pseudocode:
not used...

process_plutons
Parameters
c_l
contact_decimate
dtm
geol_clip
local_paths
output_path
pluton_dip
pluton_form
tmp_path
Pseudocode:
load sorted groups from csv file
for each polygon in GeoDataBase of geology polygons:
if Group is empty:
Group=Code
for each Group:
calculate max/min ages for group
if polygon is intrusive but not sill:
create a new Group=Code
if new Group does not exist:
add to list of Groups

calculate list of neighbour polygons using intersection test
if neghbours exist:
for each neighbour polygon:
if neighbour intrusive but not sill or neighbour not intrusive and neighbour has an age (they all do!):
if polyline is linestring:
for each line segment in linestring:
if decimate test passes:
if line segment within dtm bounds:
save contact point to ign_contacts csv file with x,y,z and Unit and to dictionary
else:
save to all_contacts csv file

if decimate test passes:
calculate normal to contact line segment
save contact orientation to csv file with dip direction and polarity varied acccording to pluton_form

update groups2 csv file with new groups

save_basal_contacts
Parameters
c_l
contact_decimate
dtm
geol_clip
intrusion_mode
path_in
Pseudocode:
explode geology polgyons so interior holes become distinct polygons
for each polygon:
build list of polygons and their atributes
load sorted stratigraphy from csv file
for each polygon in list:
if not intrusive:
if polygon Code found in sorted stratigraphy:
for each polygon in list:
if two polygons are not the same:
if two polygons are neighbours:
if second polygon is not a sill:
add neigbour to list
if first polygon has neighbours:
for each neighbour:
if neighbour polygon Code found in sorted stratigraphy:
if neighbour older than first polygon:
calculate intersection of two polygons:
if intersection is a multilinestring:
for all line segments in linestring:
save out segment with x,y,z Code
build dictionary of basal contacts and dictionary of decimated basal contacts

return dictionary of basal contacts and dictionary of decimated basal contacts

save_basal_contacts_csv
Parameters
c_l
contact_decimate
contacts
dtm
output_path
Pseudocode:
for each polyline:
if polyline is multilinestring:
for each linestring in multilinestring:
for each segment in linestring:
save contact line segment to csv file with x,y,z,Code
else if polyline is linestring:
for each segment in linestring:
save contact line segment to csv file with x,y,z,Code

save_basal_no_faults
Parameters
c_l
dist_buffer
dst_crs
ls_dict
path_fault
path_out
Pseudocode:
load fault linestrings as GeoDataBase
create polygonal buffer aorund all faults
clip basal contacts to polygonal buffer
make copy of clipped contacts
for each clipped basal contact polyline:
if polyline is GEOMETRYCOLLECTION:
remove from copy of clipped basal contacts
else:
add to dictionary

build GeoDataFrame from remaining clipped basal contacts and save out as shapefile

save_contacts_with_faults_removed
Parameters
c_l
dataset
dist_buffer
dst_crs
ls_dict
ls_dict_decimate
path_fault
path_out
Pseudocode:
no longer used...

save_faults
Parameters
c_l
dataset
fault_decimate
fault_dip
fault_min_len
output_path
path_faults
Pseudocode:
load polylines as GeoDataFrame
for each polyline:
if polyline is a fault:
calculate distance between fault endpoints

if distance greater than minimum allowed:
  for each line segment in fault polyline:
    if passes decimate test:
      if apex of triangle of current three points is > 45 degrees:
        save fault segment to csv file with x,y,z,Fault name
  calculate azimuth defined by fault endpoints
  save azimuth, fault length etc to csv file

save_fold_axial_traces
Parameters
c_l
dataset
fold_decimate
path_fold_orientations
path_folds
Pseudocode:
  load polylines as GeoDataFrame
  for each polyline:
    for each line segment in polyline:
      if fold axial trace:
        if passes decimate test:
          save trace as x,y,z,Fold name,Fold sign to csv file

save_fold_axial_traces_orientations
Parameters
c_l
close_dip
dataset
dst_crs
fat_step
fold_decimate
output_path
path_folds
tmp_path
Pseudocode:
  load geology polygons as GeoDataFrame
  load interpolated contacts as array
  load polylines as GeoDataFrame
  for each polyline:
    for each line segment in polyline:
      if fold axial trace:
        if passes decimate test:
          calculate azimuth of line segment
          calculate points either side of line segment
          find closest interpolated contact
          if interpolated contact is sub-parallel to fold axial trace:
            save orientation data either side of segment and related x,y,z,Code to csv file

save_orientations
Parameters
c_l
dtm
orientation_decimate
path_out
structures
Pseudocode:
  for each point in GeoDataFrame:
    if not intrusive:
      if point within dtm bounds:
        save orientation data and x,y,z,Code to csv file

tidy_data
Parameters
inputs
output_path
pluton_form
tmp_path
use_fat
use_group
use_interpolations
Pseudocode:
  combine all wanted orientation files into one DataFrame
  combine all wanted contact files into one DataFrame
  for each Group:
    for each contact:
      if contact found for Group:
        build list of good Groups
      else:
        build list of bad Contacts
  for each Group:
    for each contact:
      if Group has known Units:
        add to list of good Groups
      else:
        add to list of bad Contacts
  for each Group:
    for each orientation:
      if orientation in good Group and in wanted Group:
        do nothing
      else:
        add to list of bad Contacts

  update master stratigraphy and save to csv file

  for each orientation:
    if orientation not in good Group:
      do nothing
    else:
      save out as cleaned orientation csv file

  for each contact:
    if contact not in good Group:
      do nothing
    else:
      save out as cleaned contact csv file

xxxpt_dist
Parameters
x1
x2
y1
y2
Pseudocode:
  not used...

m2l_interpolation.py Functions

call_interpolator
Parameters

calc
fault_flag
l
m
n
nx
ny
x
xi
y
yi
Pseudocode:
pass arrays to appropriate interpolation function

distance_matrix
Parameters
x0
x1
y0
y1
Pseudocode:
calculate distance between two sets of points

interpolate_contacts
Parameters
bbox
c_l
calc
dtm
fault_flag
geology_file
gridx
gridy
output_path
use_gcode
Pseudocode:
create grid of positions for interpolation, or use predefined list of points
for each linestring from basal contacts:
if passes decimation test:
for each line segment in linestring:
calculate direction cosines of line segment and save to file as csv with x,y,z,etc

interpolate direction cosines of contact segments

save interpolated contacts to csv files as direction cosines and azimuth info with x,y,z,etc

interpolate_orientations
Parameters
bbox
c_l
calc
fault_flag
gridx
gridy
output_path
structure_file
this_gcode
Pseudocode:
subset points to those wanted
create grid of positions for interpolation, or use predefined list of points
for each point from orientations:
calculate direction cosines of orientations

interpolate direction cosines of orientations

save interpolated orientations to csv files as direction cosines and dip,azimuth info with x,y,z,etc

interpolate_orientations_with_fat
Parameters
bbox
c_l
calc
gridx
gridy
output_path
structure_file
this_gcode
Pseudocode:
subset points to those wanted
create grid of positions for interpolation
for each point from orientations:
calculate direction cosines of orientations
for each point from fat orientations:
calculate direction cosines of fat orientations

interpolate direction cosines of combined orientations

save interpolated orientations to csv files as direction cosines and dip,azimuth info with x,y,z,etc

join_contacts_and_orientations
Parameters
bbox
c_l
combo_file
dst_crs
dtm_reproj_file
fault_flag
geology_file
lc
lo
mc
mo
no
output_path
xy
Pseudocode:
for each orientation in grid:
rescale contact direction cosines with z cosine of orientations
save out rescaled x,y direction cosines from contacts with z direction cosine from orientations and positional x,y,z,Code

plot
Parameters
grid
x
y
z
Pseudocode:
plot array as image

process_fault_throw_and_near_orientations

Parameters
bbox
c_l
dst_crs
dtm_reproj_file
output_path
scheme
tmp_path
use_gcode
use_gcode2
Pseudocode:
for each polyline:
if fault:
for each line segment:
build list of points either side of mid-point of line segment
spatially join points with geology polygons so have unit Code
for each point to left of fault segment:
if not last point in list:
if Code not same as previous in list and not sill:
add to left list of contacts
for each point to right of fault segment:
if not last point in list:
if Code not same as previous in list and not sill:
add to right list of contacts

for each left contact:
for each right contact:
if contact Codes are the same:
calculate distance between left and right contacts
add to list of distances

interpolate contacts at positions of successfully found distances and all fault positions
interpolate orientations at positions of successfully found distances and all fault positions
combine contacts and orientations at positions of successfully found distances and all fault positions

for each successfully found distances:
calculate true displacement based on interpolated orientation, apparent displacement and assumed vertical dispalcement vector
save apprent and estimated true displacements to csv file with x,y,z

save_contact_vectors
Parameters
bbox
c_l
calc
decimate
dtm
geology_file
tmp_path
Pseudocode:
for each basal contact polyline:
for each line segment:
if passes decimation test:
save to csv file with azimuth of contact, x,y,z

scipy_idw
Parameters
x
xi
y
yi
z
Pseudocode:
call scipy IDW calc

scipy_rbf
Parameters
x
xi
y
yi
z
Pseudocode:
call scipy RBF calc

simple_idw
Parameters
x
xi
y
yi
z
Pseudocode:
calculate all distances between grid points and observations
calculate wieghts based on inverse of distances
normalise weights
return sum of weights times observations for each grid point

m2l_export.py Functions

loop2gempy
Parameters
bbox
contacts_file
dtm_reproj_file
groups_file
model_base
model_top
orientations_file
test_data_name
tmp_path
vtk
vtk_pth
Pseudocode:
load dtm, orientation, contact info
calculate gempy model

loop2geomodeller
Parameters
bbox
compute_etc
dtm_file
output_path
save_faults
test_data_path
tmp_path
Pseudocode:
write out Geomodeller task file using outputs from map2loop

pass taskfile to geomodellerbatch.exe to generate project files
optionally write out second taskfile and pass taskfile to geomodellerbatch.exe to generate model

loop2LoopStructural
Parameters
bbox
contacts_file
orientation_file
thickness_file
Pseudocode:
load thickness, orientation, contact info
create LoopStructural model

solve_pyamg
Parameters
A
B
Pseudocode:
no idea