map2loop Package

map2loop is a map deconstruction library to provide inputs to Loop ([Loop3d.org](https://loop3d.org/)). The development of map2loop is lead by **Mark Jessell** at The University of Western Australia. **Yohan de Rose** at Monash University is now making it better. Standalone map2model cpp code from Vitaliy provides fault/fault and fault/strat relationships.

1. What it does:

- Combines information extracted from vector geology maps in various forms to support 3D geological modelling.

Outputs are simple csv files that should be readable by any 3D modelling system (I think), but specific examples are provided for

- https://github.com/Loop3D/LoopStructural LoopStructural,
- https://github.com/cgre-aachen/gempy gempy and
- https://github.com/cgre-aachen/pynoddy noddy.

Loop is led by Laurent Ailleres (Monash University) with a team of Work Package leaders from:

  - Monash University: Roy Thomson, Lachlan Grose and Robin Armit
  - University of Western Australia: Mark Jessell, Jeremie Giraud, Mark Lindsay and Guillaume Pirot
  - Geological Survey of Canada: Boyan Brodaric and Eric de Kemp

- This is very much a proof of concept code that is unlikely to work first time with anything but the example dataset provided, but if you would like to try it with your own data please feel free to contact me at mark.jessell@uwa.edu.au to discuss your plans.
- The fastest install path (thanks to Yohan de Rose) is via docker, go to development version at which provides both map2loop and LoopStructural:

  http://github.com/Loop3D/map2loop-2

- otherwise follow these instructions:

  https://github.com/Loop3D/map2loop/blob/master/m2l_LS_install.md

- Problems

Any bugs/feature requests/comments please create a new issue: https://github.com/Loop3D/map2loop/issues.

- Acknowledgements

2. Installation

You will need some flavour of conda (a python package manager, https://docs.anaconda.com/anaconda/install/index.html, as well as Python ≥ 3.6

2.1 Run

To just use map2loop, issue the following

```
conda install -c conda-forge -c loop3d map2loop -y
```

2.2 Development

If you want to tinker yourself/contribute, clone the source code with

```
git clone https://github.com/Loop3D/map2loop-2.git
```

Or get the source + example notebooks with

```
git clone https://github.com/Loop3D/map2loop-2.git
git clone --single-branch --branch yohan https://github.com/Loop3D/map2loop2-notebooks
```

Navigate into map2loop-2, and issue the following to install map2loop and its dependencies. _Note_: The 'develop' flag makes your source changes take effect on saving, so you only need to run this once

```
python setup.py develop
```

2.3 Building with Docker

Fair warning, we recommend conda to almost everyone. With great software development power comes great environment setup inconvenience. You'll need to download and install the [docker containerisation software](https://docs.docker.com/get-docker/), and the docker and docker-compose CLI.

Development

a. Clone this repo and navigate inside as per above

b. Run the following and click on the Jupyter server forwarded link to access and edit the notebooks

```
docker-compose up --build
```

c. To hop into a bash shell in a running container, open a terminal and issue

```
bash
docker ps


 Find the container name or ID and then run
```

```
bash
docker exec -it <container_NAMEorID> bash
# Probably -> docker exec -it  map2loop-2_dev_1 bash
```

2.4 Usage

Our notebooks at https://github.com/Loop3D/map2loop2-notebooks cover use cases in more detail, but here is an example of processing Loop's South Australia remote geospatial data in just 20 lines of Python.

First, lets import map2loop and define a bounding box. You can use GIS software to find one or use [Loop's Graphical User Interface](https://loop3d.github.io/downloads.html) for the best experience and complete toolset. Remember what projection your coordinates are in!

2.5 Known Issues and FAQs

- Developing with docker on Windows means you won't have GPU passthrough and can’t use a discrete graphics card in the container even if you have one.
- If Jupyter links require a token or password, it may mean port 8888 is already in use. To fix, either make docker map to another port on the host ie -p 8889:8888 or stop any other instances on 8888.

- Sometimes the submodules misbehave. Ensure you specify the recurse-submodules flag with an 's' as opposed to recurse-submodule, and double-check your .git directory is clean, see https://github.com/Loop3D/map2loop-2/issues/41

2.6 Links

https://loop3d.github.io/

https://github.com/Loop3D/LoopStructural

https://github.com/Loop3D/map2loop2-notebooks

3. map2loop outputs:

map2loop outputs:

content	filename
Various stratigraphic topology graphs	*/graph/*.gml
Group-level stratigraphic relationships	*/tmp/groups.csv
Formation-level stratigraphic relationships	*/tmp/*_groups.csv
Summary strat relationships	*/tmp/all_sorts.csv or all_sorts_clean.csv
Fault-fault relationship table	*/output/fault-fault-relationships.csv
Fault-fault relationship graph	*/output/fault_network.gml
Fault-unit relationship table	*/output/unit-fault-relationships.csv
Fault-group relationship table	*/output/group-fault-relationships.csv

Digital Terrain Model:

content	filename
dtm in lat long wgs83	*/dtm/dtm.tif
georeferenced dtm	*/dtm/dtm_rp.tif

Geometry:

content	filename
Contact info with z and formation	*/output/contacts4.csv or contacts_clean.csv
Contact info with tangent info	*/tmp/raw_contacts.csv
Fault trace with z	*/output/faults.csv
Basal contacts shapefile	*/tmp/basal_contacts.shp
Clipped geology map shapefile	*/tmp/geol_clip.shp
Clipped fault & fold axial traces shapefile	*/tmp/faults_clip.shp
Pluton contacts with z and formation	*/output/ign_contacts.csv
Local formation thickness estimates	*/output/formation_thicknesses_norm.csv and formation_summary_thickness.csv
Fault dimensions	*/output/fault_dimensions.csv

Fault displacements	*/output/fault_displacement3.csv
Fault strat & thickness-based displacements	*/output/fault_strat_offset3.csv
Near-Fault strat contacts	*/output/fault_tip_contacts*.csv

Orientations:

content	filename
Bed dip dd data with z and formation	*/output/orientations.csv or orientations_clean.csv
Bed dip dd data with calculated polarity	*/output/orientations_polarity.csv
Extra orientations for empty series	*/output/empty_series_orientations.csv
Fault orientation with z	*/output/fault_orientations.csv
Clipped orientations shapefile	*/tmp/structure_clip.shp
Interpolated dip dip direction grid	*/tmp/interpolated_orientations.csv
Interpolated contact vector grid	*/tmp/interpolated_contacts.csv
Combined interpolation grid	*/tmp/interpolated_combined.csv
Pluton contact orientations	*/output /ign_orientations*.csv
Near-Fold Axial Trace strat orientations	*/output/fold_axial_trace_orientations2.csv
Estimated contact orientations	*/output/contact_orientations.csv

loop2model:

content	filename
Gempy	Notebook creates 3D model itself
Basic vtk model thanks to gempy	*/vtk/*.vtp
LoopStructural	Notebook creates 3D model itself
noddy	Notebook creates 3D model, only of faults itself

Does not deal with sills yet.

Requirements

See dependencies.txt file

Simplified calculation schemes

3.1 Topology

a. Stratigraphic relationships

i. Adjacency relationships between neighbouring geological polygons based on formation and group

ii. Directed graph based on these relationships and relative age of formations and groups

iii. Edges attributed by type of contact (intrusive, stratigraphic, fault)

b. Fault relationships

i. Relative ages of faults longer than a specified length estimated from truncation relationships

ii. Directed graph based on these relationships

iii. Cyclic relationships removed (A truncates B; B truncates C; C truncates A)

c. Fault-stratigraphy relationships

i. Adjacency matrices of relative ages of faults longer than a specified length and formations and groups based on truncation relationships

3.2 Position information

a. Digital Terrain Model (DTM)

i. DTM downloaded for defined bounding box from SRTM server

ii. Re-projected to local EPSG-defined projection system

b. Basal contacts

i. Formation based on stratigraphic relationship (assigned to younger formation)

ii. X,Y from contact nodes with optional decimation

iii. Z from DTM

iv. Nodes that are defined by faults are removed

c. Igneous contacts

i. Formation based on intrusive unit

ii. X,Y from contact nodes with optional decimation

iii. Z from DTM

iv. Nodes that are defined by faults are removed

d. Faults

i. Fault name based on id of fault

ii. Optional removal of faults below a certain fault-tip to fault-tip distance

iii. X,Y from fault nodes with optional decimation

iv. Z from DTM

e. Fold axial traces

i. Fold axial trace name based on id of fold axial trace

ii. X,Y from fold axial trace nodes with optional decimation

iii. Z from DTM

f. Local formation thickness

i. X,Y from basal contact nodes

ii. Z from DTM

iii. Thickness from distance from normal to local contact orientation to stratigraphically next upper contact polyline in the taking into account the local orientation of bedding estimated from the interpolation of basal contacts and primary orientation data

iv. Normalised formation thickness calculated for each node based on division by median of thicknesses for each formation

g. Local fault displacement

i. X,Y from fault contact nodes

ii. Z from DTM

iii. Displacement calculated by finding distance between equivalent stratigraphic contacts either side of the fault

3.3 Gradient information

a. Primary dip/dip direction

i. Orientations of bedding, but filter out dip = 0

ii. X,Y from primary data with optional decimation

iii. Add Z from DTM

iv. Add geology polygon formation info

b. Fault orientations

i. Normal to fault tips for azimuth

ii. X,Y from midpoint between fault tips

iii. Dip as user-defined conceptual constraint

c. Near-Fold Axial Trace orientations

i. X,Y step out normal to fat from local nodes of fold axial trace polyline with optional decimation

ii. Add Z from DTM

iii. Dip direction from local normal to fat and sign of fold axis

iv. Dip arbitrarily set by user

d. Near-fault orientations

i. X,Y step out normal to fault from local nodes of fault polyline with optional decimation

ii. Add Z from DTM

iii. Dip and dip direction from interpolation of basal contacts and primary orientation data

iv. Add geology polygon formation info

e. Empty series orientations

f. Igneous contacts

i. X,Y from local nodes of igneous contact polyline with optional decimation

ii. Add Z from DTM

iii. Dip and polarity arbitrarily defined by user

iv. Dip direction from local normal to igneous contact interpolation of basal contacts

v. Add geology polygon formation info

3.4 Minimum map2loop inputs

3.4.1 EPSG coordinate reference system for input data (metre-based projection like UTM)

3.4.2 Max/min coordinates of area of interest

3.4.3 Geology polygons:

-a. All polygons are watertight -b. Polygons stop on faults -c. Polygons have as attributes:

-i. Object ID -ii. Stratigraphic code -iii. Stratigraphic 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 (bigger number is older))

3.4.4 Fault/Fold Axial Trace Polylines:

-a. Faults terminate on other faults but do not cross -b. Faults/Folds have as attributes:

-i. Object ID -ii. Field that determines if polyline is fault or fold axial trace -iii. Field that determine type of fold axial trace e.g. syncline or anticline) -iv. Faults can have dip/dip direction info

3.4.5 Bedding orientations:

-a. Assumes dip/dip direction or dip/strike data -b. Orientations have as attributes:

-i. Dip -ii. Dip Direction or strike

4. Using *map2loop* with your own or standard datasets

In order to deconstruct a map with *map2loop*, we need to undertake the following steps:

a. Ensure that we have sufficient correctly formatted data, with sufficient information to allow the calculations to work.

b. Once we have these data, we then need to define a set of parameters that inform *map2loop* about this data, and the details of the calculations to undertaken. Use the notebook **Utility 1 - Config file generator.ipynb** to make this slightly less painful.

c. Finally, once we have the data and *map2loop* control parameters sorted, we are ready to create a small python script to test the system.

5. Minimum *map2loop* data requirements:

5.1 Vector Geospatial File Data Formats:

'DXF': 'raw', 'CSV': 'raw', 'OpenFileGDB': 'r', 'ESRIJSON': 'r', 'ESRI Shapefile': 'raw', 'GeoJSON': 'rw', 'GeoJSONSeq': 'rw', 'GPKG': 'rw', 'GML': 'raw', 'MapInfo File': 'raw'

r=read, a=append, w=write

- geology polygons with stratigraphic code and rock type info (required)
- fault polylines (required)
- bed dips as points in dip, dip direction (required)
- mineral deposit layer (optional)
- fold axial trace layer (optional)

5.2 Geology Polygons (or Multipolygons):

- no gaps or overlaps between polygons, nodes from neighbouring polygons coincide (we have code to fix errors when the mismatch is smaller than the minimum node spacing).
- Stratigraphic Coherency: - Ideally the map should consist of polygons which all have the same stratigraphic heirarchical level, e.g. all formations, all groups, all members etc. This is often not the case, and it makes unravelling the stratigraphy more complex, as the parent and child may appear in

the same map, so the age sorting needed to build the model becomes ambiguous. National or state-level stratigraphies, and even stratigraphies described on map sheet legends are by definition simplifications of the the local system extracted by *map2loop*. *map2loop* uses the local stratigraphy first and then uses the regional stratigraphy (if available) as a guide to reduce the uncertainty.

- **Attributes with (data type) and {code} (see section 2.2):**
    - Fine Scale Stratigraphic coding, e.g. formation name (str) {'c'}
    - Coarser Scale stratigraphic coding, such as its parent, e.g. group (str) {'g'}
    - Alternate Coarser Scale stratigraphic coding, e.g. group (str) {'g2'}
    - Relative or absolute maximum age of Fine Scale Stratigraphic Code (float) {'max'}
    - Relative or absolute minimum age of Fine Scale Stratigraphic Code (float) {'min'}
    - Litho code to help determine if rock is intrusive (str) {'ds'}
    - Litho Code to help detemrine if this is a sill-like body (str) {'sill'}
    - Unique ID of polygon (str) {'o'}

5.3 Fault Polylines:

- single polyline per fault (no multipolylines)
- nodes on faulted boundaries coincident with geology polygon nodes
- **Attributes:**
    - Text that identifies polyline as a fault (str) {'fault'}
    - Dip of fault (str or float) {'fdip'}
    - Dip Direction of Fault (str or float) {'fdipdir'}

5.4 Bedding Points:

- single points (no multipoints)

Attributes:

- Dip (float) {'d'}
- Dip Direction or Strike (float) {'dd'}
- Code to show what convention is used: Strike RHR, or DD at the moment (str) {'otype'}
- Code to show what type of foliation: Bedding, S1 etc. (str) {'sf'}
- Code to say this unit is overturned (str) {'bo'}

5.5 Mineral Deposits: (Optional)

- single points (no multipoints)

Attributes:

- Site Code (str) {'msc'}
- Name of deposit (str) {'msn'}
- Type of feature: open pit, occurence, abandoned etc., (str) {'mst'}
- Code to show what main commodity: Fe, Iron, etc. (str) {'mtc'}
- Code to show what main commodity class: Industrial, Metal, etc. (str) {'mcom'}

5.6 Fold Axial Trace Polylines: (Optional)

- single polyline per trace (no multipolylines)

Attributes:

- Text that identifies polyline as a fold axial trace (str) {'feature'}
- Code that defines fold as syncline (str) {'syn'}

6) Defining *map2loop* parameters:

There are four types of parameters we need to define in order to use *map2loop*:

- **a) The paths or URLs** that tell *map2loop* where the information layers (GIS files or online sources) are stored. These are passed to the *map2loop* update_config() method.
- **b) The names of fields and some text flags** that tell *map2loop* where specific information can be retrieved from these layers. These are stored in an hjson format text file the path of which is passed to the *map2loop* Project() method.
- **c) A data output path, bounding box and Coordinate Reference System** information to define the extent of the model and where to put it. This is passed to the *map2loop* update_config() method
- **d) Parameters that control the specific functioning** of *map2loop*: what to calculate, what decimation factors to apply to the augmented data outputs, what resolution interpolations to use etc. These is passed to the map2looop project.run() method.

6.1 paths

These is passed to the *map2loop* Project() method.

Examples:

Remote WFS layers: See Example 1 Notebook

```
proj=Project(geology_file = 'http://geo.loop-gis.org/geoserver/loop/wfs?service=WFS&version=1.0.0&request=GetFeature&typeName=loop:geol_500k&bbox={}&srs=EPSG:28350',
           fault_file='http:// etc.',
           structure_file='http:// etc.',
           mindep_file='http:// etc.,
           metadata="http://anyurl.org/mydata.hjson',
           remote=True)
```

where remote=True signifies that WFS-served data will be accessed.

Standard Australia State Geological Surveys datasets, we have predefined the paths for all data and the following code is sufficient: See Example 2 Notebook

```
proj = Project(
              loopdata_state="WA", # choice between 'WA','QLD','NT','NSW','VIC','SA', 'TAS'
              )
```

Local GIS layers: see Example 3 notebook

```
proj=Project(geology_file="source/geology_polygons.shp",
             fault_file="source/fault_polylines.shp",
             fold_file="source/fold_polylines.shp",
             structure_file="source/bedding_points.shp",
             mindep_file="source/mindep_points.shp",
             metadata="source/meta.hjson",
             dtm_file="./source_data/terr50_gagg_gb_all.tif",
             remote=False)
```

where remote=False signifies that local GIS files will be accessed. Paths can be relative or absolute, or even a URL, however for URLs, the components of the shapefile or TAB file have to be zipped up.

6.2 Layer field codes:

You will need to create or modify an *hjson* format file that provides the names of fields and some text flags that tell *map2loop* where and what specific information can be retrieved from these layers. These are stored in an hjson format text file the path of which is passed to the *map2loop* Project() method. The easiest way to get started is to use a jupyter notebook allows you to reduce errors by providing a primitive GUI for creating an *hjson* config file and associated python script, named: **Utility 1 - Config file generator.ipynb**. Alternatively if you are brave you can edit the values to the right of the colon in each row of an existing *hjson* file. For example to specify that the field in the geospatial layer that contains bedding dip information is called **MYDIP**, replace the appropriate code in the *hjson* file below with:

"d":"MYDIP",

Some verification is carried out by *map2loop* to ensure the required parameters have been defined. In the following section *field* refers to a field name in a geospatial layer; *text* refers to some text in the contents of a field for a specific geometric object. You shouldn't use the same field for different codes as this may cause problems.

```
{
    # 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
    # text to search for in field defined by sf code to show that this is a bedding measurement
    "bedding": "Bed",
    # flag to determine measurement convention (currently "strike" or "dip direction")
    "otype": "dip direction",
    "bo": "TYPE",  # field that contains type of foliation
    # text to search for in field defined by bo code to show that this is an overturned bedding measurement
    "btype": "overturned",
    # Stratigraphy----------------------------
    "g": "GROUP_",  # field that contains coarser stratigraphic coding
    # field that contains alternate coarser stratigraphic coding if "g" is blank
    "g2": "SUPERSUITE",
    "c": "UNITNAME",  # field that contains finer stratigraphic coding
    "ds": "DESCRIPTN",  # field that contains information about lithology
    # field that contains alternate stratigraphic coding (not used??)
    "u": "CODE",
    "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
    # text to search for in field defined by r1 code to show that this is an intrusion
    "intrusive": "intrusive", # text to search for in field defined by ds code to show that this is an volcanic (not intrusion) "volcanic": "volcanic",
    # 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
    # text to search for in field defined by mst code that shows site to ignore
    "minf": "Infrastructure",
    # 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
    # text to search for in field defined by f code to show that this is a fault
    "fault": "Fault",
    "ff": "FEATURE",  # field that contains information on type of structure
    # text to search for in field defined by f code to show that this is a fold axial trace
    "fold": "Fold axial trace",
    "fdip": "DIP",              # field for numeric fault dip value
    # text to search for in field defined by fdip to show that this has no known dip
    "fdipnull": "0",
    "fdipdir": "DIP_DIR",        # field for text fault dip direction value
    # flag for text fault dip direction type num e.g. 045 or alpha e.g. southeast
    "fdipdir_flag": "alpha",
    "fdipest": "DIP_EST",        # field for text fault dip estimate value
    # text to search for in field defined by fdipest to give fault dip estimate in increasing steepness
    "fdipest_vals": "gentle,moderate,steep",
    # field that contains information on name of fault (not used??)
    "n": "NAME",
    "t": "TYPE",  # field that contains information on type of fold
    # text to search for in field defined by t to show that this is a syncline
    "syn": "syncline",
    # ids----------------------------
    "o": "OBJECTID",  # field that contains unique id of geometry object
    "gi": "GEOPNT_ID",  # field that contains unique id of structure point
    "deposit_dist": 500
}
```

6.3 ROI, Projection, output paths

A data output path which points to a new or existing directory (a new directory will be created if needed), bounding box and Coordinate Reference System information to define the extent of the model. This is be passed to the *map2loop* update_config() method

```
proj.update_config(
                  out_dir='./model-test',
                  overwrite='overwrite',
                  bbox_3d={
                       "minx": 500000,
                       "miny": 7490000,
                       "maxx": 545000,
                       "maxy": 7520000,
                       "base": -4800,
                       "top": 1200,
                    },
                  proj_crs={'init': 'EPSG:28350'},
                  quiet='none'
                )
```

- where bbox coordinates are in CRS defined by proj_crs
- where overwite can be 'overwrite', 'true'
- where quiet controls whether we allow or block print statements and matplotlib figures. Use 'none' to quiet nothing, 'all' to quiet everything, 'no-figures' to disable plots and allow text output. Defaults to 'none'

6.4 Full list of update_config flags:

Project flags:

- **out_dir** Path to write output files to. :type out_dir: string
- **overwrite** Allow overwriting the given out_dir if it exists, false, true or in-place, defaults to false :type overwrite: string, optional
- **bbox_3d** 3D bounding box of coordinates and base/top values defining the area, defaults to { "minx": 0, "maxx": 0, "maxx": 0, "maxy": 0, "base": -10000, "top": 1200, } :type bbox_3d: dict, optional
- **dtm_crs** Set the projection of the dtm, defaults to {'init': 'EPSG:4326'} :type dtm_crs: dict, optional
- **proj_crs** Set the projection of the input data, defaults to None :type proj_crs: dict, optional
- **step_out** How far to consider outside the re-projected dtm, defaults to None :type step_out: int, optional
- **quiet** Allow or block print statements and matplotlib figures, 'None' to quiet nothing, 'all' to quiet everything, 'no-figures' to disable plots and allow text output. Defaults to 'None' :type quiet: string, optional
- **clut_path** Path to custom map colours file :type clut_path: string, optional
- **model_engine** Which modelling engine to use and set associated flags for, defaults to loopstructural :type model_engine: string, optional
- **run_flags** Global dictionary that defines custom params such as decimation and minimum fault length, see below :type run_flags: dict, optional
- ****kwargs**

6.5 Calculation control parameters

These control the specific functionality of *map2loop*: what to calculate, what decimation factors to apply to the augmented data outputs, what resolution interpolations to use etc. These are passed to the *map2looop* project run() method:

proj.run()

This method performs the data processing steps of the *map2loop* workflow, and can be modified by including the following parameters [defaults](data type):

- **aus**: Indicates if area is in Australia for using ASUD, the Australian Stratigraphic Units Database to redfine stratigraphic relationships. Should only be True in Australia, and when the finest stratigraphic level is the ASUD standard Formation name. [True] (bool)
- **close_dip**: Dip to assign to all new fold axial trace orientations. If -999 then the nearest interpolated dip for that supergroup will be used instead. [-999] In degrees (int)
- **contact_decimate**: Save every nth contact data point. 0 means save all data. [5] (int)
- **contact_dip**: Dip to assign to all new basal contact orientations. If -999 then the nearest interpolated dip for that supergroup will be used instead. [-999] In degrees (int)
- **contact_orientation_decimate**: Save every nth contact orientation point. 0 means save all data. [5] (int)
- **deposits**: Mineral deposit names for focused topology extraction. ["Fe,Cu,Au,NONE"] Topological analysis of faults and strat will only be carried out relative to these deposit type. NONE must always be one of the types (str)
- **dist_buffer**: Buffer for processing basal contacts. Basal contact vertices less than this distance from the fault will be ignored. [10] In metres. (int)
- **dtb**: Path to depth to basement grid. Geotif of depths in the same projection system as everything else. [''] (str)
- **fat_step**: How much to step out normal to the fold axial trace. Distance in metres. [750] In metres. (int)
- **fault_decimate**: Save every nth fault data point along fault tace. 0 means save all data. [5] (int)
- **fault_dip**: default fault dip [90] In degrees (int)
- **fold_decimate**: Save every nth fold axial trace data point. 0 means save all data. [5] (int)
- **interpolation_scheme**: What interpolation method to use of scipy_rbf (radial basis) or scipy_idw (inverse distance weighted). ['scipy_rbf'] (str)
- **interpolation_spacing**: Interpolation grid spacing in meters. Used to interpolation bedding orientations [500] In metres or if a negative value defines fixed number of grid points in x & y (int)
- **intrusion_mode**: 1 to exclude all intrusions from basal contacts, [0] to only exclude sills. [0] (int)
- **max_thickness_allowed**: when estimating local formation thickness [10000] in metres. (int)
- **min_fault_length**: Min fault length to be considered. In metres. [5000] In meters. (int)
- **misorientation**: [30] Maximum misorientation in pole to great circle of bedding between groups to be considered part of same supergroup (int)
- **null_scheme**: How null values present in the depth to basement geotif. ['null'] (str)
- **orientation_decimate**: Save every nth orientation data point. 0 means save all data. [0] type int
- **pluton_dip**: default pluton contact dip [45] In degrees (int)
- **pluton_form**: Possible forms from domes, saucers or pendant. ['domes'] (str)
- **thickness_buffer**: How far away to look for next highest unit when calculating formation thickness [5000] In metres. (int)
- **use_fat**: Use fold axial trace info to add near-axis bedding info [True] (bool)
- **use_interpolations**: Use all interpolated dips for modelling [True] (bool)

6.6 Calculation workflow parameters

- **seismic_section**: Add data from a single seismic section (paths hardwired for the moment) [False] (bool)
- **cover_map**: Add data from a depth to basement grid (paths hardwired for the moment) [False] (bool)
- **near_fault_interpolations**: Add stratigraphic info near faults [False] (bool)
- **fold_axial_traces**: Add dip info either side of fold axial trace to enhance second order folds [False] (bool)
- **stereonets**: Calculate stereonets to define supergroups [True] (bool)
- **formation_thickness**: Calculate formation thickness [True] (bool)
- **polarity**: Calculate bedding polarity (doesn't work!!) [False] (bool)
- **strat_offset**: Calculate stratigraphic offset across faults [True] (bool)
- **contact_dips**: Add fixed or interpolated dips to contacts [True] (bool)

Individual workflow parameters can be overwritten AFTER the call to proj.update_config() as follows:

```
proj.workflow['contact_dips'] = False
```

7. Example minimum code:

An example minimum code to run *map2loop* with mostly default settings might look like this (and see the notebook **Example 3 - Local Source Data.ipynb**):

```
from map2loop.project import Project

proj=Project(geology_file="source/geology_polygons.shp",
            fault_file="source/fault_polylines.shp",
            fold_file="source/fold_polylines.shp",
            structure_file="source/bedding_points.shp",
            mindep_file="source/mindep_points.shp",
            metadata="source/meta.hjson"
            dtm_file="http://services.ga.gov.au/gis/services/DEM_SRTM_1Second_over_Bathymetry_Topography/MapServer/WCSServer?",
            )

proj.update_config(
                    out_dir='./model-test',
                    bbox_3d={
                        "minx": mbbox.total_bounds[0], #500000,
                        "miny": mbbox.total_bounds[1], #7490000,
                        "maxx": mbbox.total_bounds[2], #545000,
                        "maxy": mbbox.total_bounds[3], #7520000,
                        "base": -4800,
                        "top": 1200,
                        "local': True
                    },
                    proj_crs={'init': 'EPSG:28350'}
                    )

proj.run()
```

8. Pseudocode for key calculations

```
save_basal_contacts

           explode geology polgyons so interior holes become distinct Polygons
           for each Polygon:
                  build list of Polygons and their atributes modelling
           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_no_faults

             load fault linestrings as GeoDataBase
             create Polygonal buffer aorund modell 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_fold_axial_traces_orientations

            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
```

```
interpolate_contacts

             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

             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
```

```
join_contacts_and_orientations

           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
```

```
calc_thickness

            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
```