cfm-leapfrog

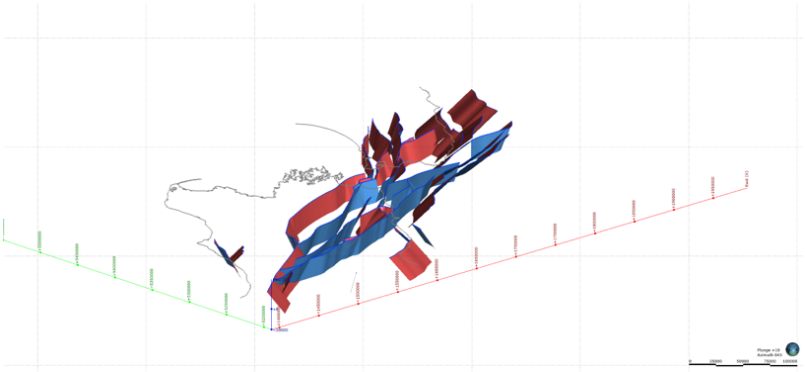
Release 1.0.0

Andy Howell and Camilla Penney

Jan 08, 2025

INTRODUCTION AND SETUP:

This module is designed to help create 3D models of complex fault systems, and works by projecting surface traces to depth. Leapfrog software should be used for mesh cutting operations.



CHAPTER

ONE

INTRODUCTION

Welcome. This online documentation is intended to walk users through the steps described in Howell et al. (in review). That paper focusses more on the rationale for each step in the workflow, whereas here we focus on the computational steps required to implement the workflow.

Please do get in touch if you have any questions or suggestions for improvement.

CHAPTER

TWO

PREPARING INPUTS FOR USE

2.1 Shapefile fields

The primary input to the workflow is a shapefile of similar (GeoJSON and GPKG also work). This shapefile includes the linework for surface traces, but should also include enough metadata to unambiguously constrain the fault geometry.

A minimal example of a shapefile with the required fields is provided in `tutorial_gis/central_nz_minimal_data.shp`.

The required fields are:

- `Name`: a unique identifier for each fault
- `Fault_ID`: a unique identifier (number) for each fault
- `Dip_pref`: the preferred dip of the fault in degrees
- `Dip_dir`: the dip direction of the fault in compass direction (e.g. "N", "NE", "E", "SE", "S", "SW", "W", "NW"). Note that two possible dip azimuths will be calculated based on the fault trace. Inclusion of this letter-based field allows the workflow to choose between these two possible azimuths, and also allows consistency checks (e.g., an error will be thrown if "E" is provided for an EW-striking fault).
- `SR_pref`: the preferred slip rate of the fault in mm/yr

The `SR_pref` field is optional, but if it is not provided, the automated method to estimate which faults terminate against each other (cutting hierarchy) will not work.

Note: Other fields can be included in the shapefile, but they will be ignored by the workflow.

2.2 Vertex spacing

The workflow requires that the fault traces are represented by a series of vertices. The spacing of these vertices is important for the accuracy of the calculations. We suggest a spacing of 1 km, but this can be adjusted based on the size of fault networks and complexity of the fault geometries. It would be possible to set this point spacing programmatically using python, but in our experience this iterative process is best carried out interatively in GIS software, for example using the Densify by interval tool in QGIS.

CHAPTER

THREE

INSTALLATION INSTRUCTIONS AND UPDATES TO CODE

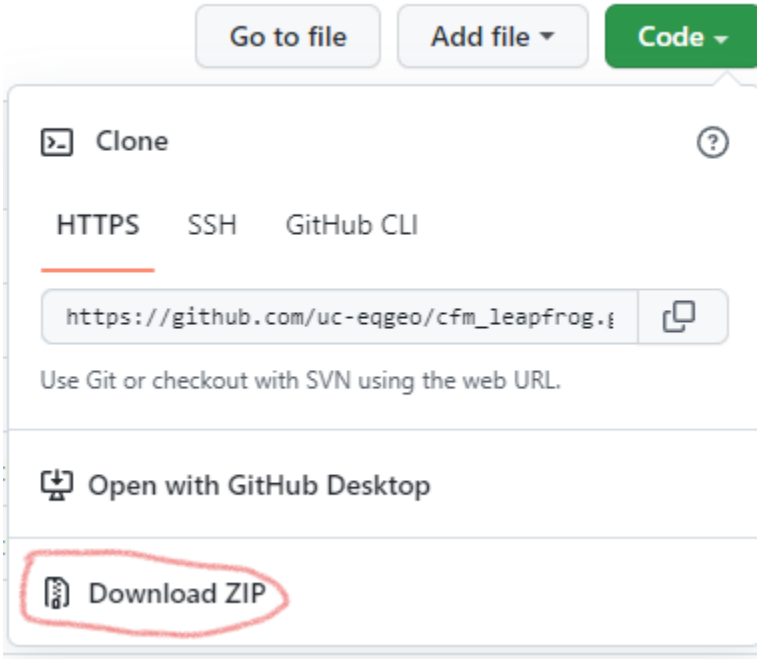
3.1 Prerequisites

You'll need some form of conda, for example Anaconda3 or miniforge.

Installation instructions for Anaconda can be found here.

Alternatively (and better in most ways), install miniforge.

You will also need to download the code from the GitHub repository (https://github.com/uc-eqgeo/cfm_leapfrog). If you are new to git, this is probably best to do using the download zip interface on GitHub. Make sure you unzip it if you do that.



3.2 Installation

1. Open Anaconda or Miniforge Prompt
2. Run `cd {PATH_TO_REPO}` and hit `Enter`, where `{PATH_TO_REPO}` is the location where you have downloaded the code to, for example `C:\Users\{USER}\Documents\projects\cfm_leapfrog`. In this case, `{USER}` is your username. Note that if you have downloaded to a drive that is different from your `C:` drive, you will need to change drives first; for example, type `D:` followed by `Enter`.

3. Run `conda env create` and hit `Enter`.
4. Run `conda activate leapfrog-fault-models` and hit `Enter`

3.3 Updating the code to a new version

In the early stages of this project, we will be updating the code frequently. To update to the latest version, follow these steps:

1. Open Anaconda or Miniforge Prompt
2. Activate the environment: `conda activate leapfrog-fault-models` followed by `Enter`
3. Enter the command `pip install git+https://github.com/uc-eqgeo/cfm_leapfrog/#subdirectory=tectonic_fault_mesh_tools` followed by `Enter`

CHAPTER

FOUR

RUNNING TUTORIAL EXAMPLES USING JUPYTER

4.1 Open Jupyter notebooks

1. Open Anaconda or Miniforge Prompt
2. Run `cd {PATH_TO_REPO}` and hit `Enter`, where `{PATH_TO_REPO}` is the location where you have downloaded the code to, for example `C:\Users\{USER}\Documents\projects\cfm_leapfrog`. In this case, `{USER}` is your username. Note that if you have downloaded to a drive that is different from your `C:` drive, you will need to change drives first; for example, type `D:` followed `Enter`.
3. Type `jupyter notebook` and hit `Enter`.

These actions should open a new tab in your web browser that looks something like this.

4. Click through `docs` followed by `tutorials` to enter the directory where the example notebooks are stored. 5. Click on a tutorial to open it in a new tab:


jupyter
Quit
Logout
Files
Running
Clusters
Select items to perform actions on them.
Upload
New
0
/ docs / tutorials
Name
Last Modified
File size
..
seconds ago
tutorial_gis
2 minutes ago
tutorial_images
23 minutes ago
define_connected.ipynb
5 minutes ago
3.97 kB
central_gt1_5_connected__suggested.csv
2 minutes ago
1.33 kB

CHAPTER

FIVE

DEFINING CONNECTED FAULT SYSTEMS

This notebook demonstrates how to identify which fault segments in a shapefile should be connected up into large fault systems.

We'll begin by importing relevant modules

```
# Import modules
from fault_mesh.faults.leapfrog import LeapfrogMultiFault
import os
import numpy as np
import geopandas as gpd
```

5.1 Setting useful parameters

There are a few parameters that are worth setting here:

1. Coordinate system via EPSG code (2193 for New Zealand) — this is optional but useful if you wish to visualize contours in GIS software.
2. Trimming gradient (α_{trim}) for clipping depth contours — see Section 3.3 of Howell et al. (in review)
3. Dip multiplier and strike multiplier for calculating Θ_{change} — see Section 3.3 of Howell et al. (in review)

```
# Set coordinate system (optional) EPSG code
# If not desired, set to None
epsg = 2193

# Trimming gradient (alpha) and strike and dip multipliers for trimming depth contours␣
↪of multi-segment faults
trimming_gradient = 1.
dip_multiplier = 1.
strike_multiplier = 0.5
```

5.2 Reading in faults

First, you need to read in your GIS representation of faults. In this example, we use a subset of faults from the New Zealand Community Fault Model (Seebeck et al., 2022).

```
# Read in fault data from shapefile
fault_data = LeapfrogMultiFault.from_shp("tutorial_gis/central_nz_minimal_data.shp",␣
↪remove_colons=True, epsg=epsg, trimming_gradient=trimming_gradient,
```

(continues on next page)

(continued from previous page)

```
                                        dip_multiplier=dip_multiplier, strike_
↪multiplier=strike_multiplier)
```

5.3 Finding connections between segments

Now your data are read in, you need to set the distance tolerance. This tolerance is the minimum horizontal distance between two fault traces that is allowed to count as a connection.

```
dist_tolerance = 200.
```

The next cell uses python module networkx to find segment traces that are within the specified distance tolerance of each other.

```
fault_data.find_connections(verbose=False)
```

```
Found 156 connections
Found 142 connections between segment ends
```

It is now necessary to write out these connections for manual editing and review. The file will be written out into the same directory as this Jupyter notebook. It will have a prefix supplied by you and the suffix "_suggested.csv".

```
fault_data.suggest_fault_systems("central_gt1_5_connected")
```

This will create a CSV file that looks like this:

	A	B	C	D	E
1	Alfredton South combined	Alfredton North	Alfredton South	Dreyers Rock	
2	Needles combined	Alpine Jacksons to Kaniere	Alpine Kaniere to Springs Junction	Alpine Springs Junction to Tophouse	Awatere
3	Cloudy 3 combined	Awatere Northeast 1	Awatere Northeast 2	Cloudy 1	Cloudy 2
4	Chancet - Campbell Bank combined	Campbell Bank	Chancet		
5	Wairarapa 2 combined	Carterton	Wairarapa 1	Wairarapa 2	Wairara
6	Clarence combined	Clarence Central	Clarence Northeast	Clarence Southwest	
7	Te Rapa combined	Hope Te Rapa	Te Rapa 1	Te Rapa 2	
8	Pahaua - Kekerengu Bank combined	Kekerengu Bank	Pahaua		
9	Waewaepa combined	Makuri - Waewaepa	Waewaepa		
10	Wellington Tararua 2 combined	Mohaka South	Wellington Pahiatua	Wellington Tararua 1	Wellingt
11	Ohariu combined	Ohariu	Ohariu South 1	Ohariu South 2	
12	Uruti combined	Uruti Basin	Uruti Ridge 2		
13	Wellington Hutt Valley combined	Wellington Hutt Valley 1	Wellington Hutt Valley 2	Wellington Hutt Valley 3	Wellingt

The name of the combined fault system is in the first column, and names of the faults that make up the connected system are in subsequent columns.

5.4 Making and incorporating manual edits

The automatically-generated fault system suggestions will (by design) include hyper-connected fault systems that need to be broken up. At this stage, the best way to break up these networks into smaller fault systems is to do it manually by editing the CSV file. An example below shows a new line added to the CSV representing the Hope Fault system – The Hope Fault is grouped with the Alpine and Kekerengu-Needles fault systems in the automatically-generated connections CSV. **Make sure you save this new file with a different name to avoid overwriting it!**

Alfredton combined	Alfredton North	Alfredton South	Dreyers Rock	
Alpine combined	Alpine Jacksons to Kaniere	Alpine Kaniere to Springs Junction	Alpine Springs Junction to Tophouse	Wairau
Hope combined	Hope Conway	Hope Hanmer SE	Hope Hanmer SW	Hope Hope River
Jordan - Kek - Needles combined	Jordan	Kekerengu 1	Needles	
Awatere - Vernon combined	Awatere Northeast 1	Vernon 1	Vernon 2	Vernon 3
Cloudy combined	Cloudy 1	Cloudy 2	Cloudy 3	
Campbell Bank - Chancet combined	Campbell Bank	Chancet		
Wairarapa combined	Wairarapa 1	Wairarapa 2	Wairarapa 3	Wairarapa Needles
Clarence combined	Clarence Central	Clarence Northeast	Clarence Southwest	
Te Rapa combined	Hope Te Rapa	Te Rapa 1	Te Rapa 2	
Kekerengu Bank - Pahaua combined	Kekerengu Bank	Pahaua		
Waewaepa combined	Makuri - Waewaepa	Waewaepa		
Wellington combined	Mohaka South	Wellington Pahiatua	Wellington Tararua 2	Wellington Tararua 3
Ohariu combined	Ohariu	Ohariu South 1	Ohariu South 2	
Uruti combined	Uruti Basin	Uruti Ridge 2		
Wellington Hutt Valley combined	Wellington Hutt Valley 1	Wellington Hutt Valley 2	Wellington Hutt Valley 3	Wellington Hutt Valley 4

Once you have made the necessary edits, read your new CSV to overwite the automatically-generated connected fault systems:

```
fault_data.read_fault_systems("./define_connections_data/central_gt1_5_connected_edited.
↪csv")
fault_data.generate_curated_faults()
```

CHAPTER

SIX

DEFINING A CUTTING HIERARCHY

Now you have defined your fault system (which you will later use to create meshes), it is necessary to specify which faults terminate against other faults. For example, it seems highly likely that the western end of the Hope Fault is truncated at depth by the Alpine fault. This complex mesh cutting is best achieved using dedicated software such as leapfrog (see below), but for cutting to be done automatically, it is best to first specify a *cutting hierarchy*.

6.1 Suggesting a hierarchy based on slip rate

A first pass at a hierarchy can be generated based purely on fault slip rate. Assuming that the fault data you have already read in have slip rates associated with them, this first pass is easy to make:

```
fault_data.suggest_cutting_hierarchy("central_gt1_5_hierarchy")
```

This operation simply orders the faults (or fault systems) in your model in descending order of slip rate. For connected fault systems that have segments with different slip rates, the maximum slip rate of any segment in that fault system is used to place the fault system in the cutting hierarchy.

6.2 Editing the cutting hierarchy

You can then edit this hierarchy by switching the order of lines in the file. For any pair of faults/systems that intersect, the fault closer to the bottom of the file will terminate against the fault closer to the top of the list. An example of a situation where editing is desirable is illustrated below. The maximum slip rate of the Jordan-Kekerengu-Needles Fault System (23 mm/yr) is faster than the corresponding maximum for the Hope Fault (15.8 mm/yr), but we wish to create a fault model where the Jordan Fault terminates against the Hope Fault. We effect this termination by moving `Hope combined` above `Jordan - Kek - Needles combined` in the CSV file. For similar reasons, we move the `Hanmer` Fault above `Hope Hanmer NW`.


Alpine combined
Jordan - Kek - Needles combined
Hope combined
Hope Hanmer NW
Wairarapa combined
BooBoo
Wellington Hutt Valley combined
Kakapo
Wellington combined
Awatere - Vernon combined
Awatere Southwest
Alfredton combined
Palliser - Kaiwhata
Clarence combined
Hope Taramakau
Mataikona
Riversdale
Riversdale North
Uruti combined
Porters Pass
Waewaepa combined
Campbell Bank - Chancet combined
Kekerengu Bank - Pahaua combined
Te Rapa combined
Madden Banks
Wharekauhau
Carterton
Hanmer
Browning Pass
Conway Trough East
Fidget
AmberBlyth
Awatere Northeast 2
Cloudy combined
Hawkswood
Maimai
Masterton
Northern Ohariu
Ohariu combined
Opouawe - Uruti
Pa Valley
Paparoa West
Papatea
Whareama Bank
Alpine combined
Hope combined
Jordan - Kek - Needles combined
Hanmer
Hope Hanmer NW
Wairarapa combined
BooBoo
Wellington Hutt Valley combined
Kakapo
Wellington combined
Awatere - Vernon combined
Awatere Southwest
Alfredton combined
Palliser - Kaiwhata
Clarence combined
Hope Taramakau
Mataikona
Riversdale
Riversdale North
Uruti combined
Porters Pass
Waewaepa combined
Campbell Bank - Chancet combined
Kekerengu Bank - Pahaua combined
Te Rapa combined
Madden Banks
Wharekauhau
Carterton
Browning Pass
Conway Trough East
Fidget
AmberBlyth
Awatere Northeast 2
Cloudy combined
Hawkswood
Maimai
Masterton
Northern Ohariu
Ohariu combined
Opouawe - Uruti
Pa Valley
Paparoa West
Papatea
Whareama Bank

We read in this new hierarchy as follows:

```
fault_data.read_cutting_hierarchy("./define_connections_data/central_gt1_5_hierarchy_
↪edited.csv")
```

CHAPTER

SEVEN

CREATE SHAPEFILES FOR MESH CREATION

The final pre-meshing step is the creation of files that can be combined with meshing software to create triangular mesh representations of faults. Although it is possible to construct these triangular surfaces in multiple software packages (for example, MOVE 3D), the following discussion is geared towards use with Leapfrog Geo software.

7.1 Create directories to hold shapefiles

For organisational reasons, it helps to have the different shapefiles in different directories

```
for dir_name in ["depth_contours", "traces", "footprints"]:
    if not os.path.exists(dir_name):
        os.mkdir(dir_name)
```

7.2 Write out shapefiles

```
for fault in fault_data.curated_faults:
    # Generate depth contours
    fault.generate_depth_contours(np.arange(2000, 32000., 2000.), smoothing=False)
    # Write contours to files
    fault.contours.to_file(f"depth_contours/{fault.name}_contours.shp")
    # Write traces
    fault.nztm_trace_geoseries.to_file(f"traces/{fault.name}_trace.shp")

# Write fault footprints
for fault in reversed(fault_data.curated_faults):
    fault.adjust_footprint()
    fault.footprint_geoseries.to_file(f"footprints/{fault.name}_footprint.shp")
```

CHAPTER

EIGHT

LEAPFROG 3D MESH GENERATION AND FAULT CUTTING

8.1 Import inputs

1. Right click on GIS Data, Maps and Photos – New Subfolder + create a subfolder with your project name, then subfolders of this named `traces`, `footprints`, `contours`.
2. Import the contours, footprints and traces into these folders by right clicking on the folder – import vector data
3. Import the grid to use as the depth cut off for faulting by right clicking on Meshes – import elevation grid

8.2 Run fault modelling

1. `File -> Fault System Modelling -> Rebuild Fault System`. This will ask you to select a json file (easiest to keep this in the folder where the project is) **"name"** needs to be the name of your first subfolder, **"faults_filename"** is the file with the fault hierarchy in it, "basement_name" is the location (in the leapfrog project) of the base mesh, **"uncut_zs"** is the depth range to make the initial uncut faults in, "trace buffer" … , "cut_zs" is the depth range to cut the fault surfaces to (in practice this will be the depth to the depth mesh but could use this if wanted a uniform surface), "resolution" is the resolution of fault interactions/terminations in m (based on some issues I've been having, might want to set this a bit smaller e.g. 100m), **"ignore_terminations"**, **"run_intersection_check"** runs a check on whether the faults intersect once they've been cut which is slightly less important now that we're making buffers around the areas where faults intersect, **"use_uncut_terminations"** can be used to truncate faults against the uncut version of their meshes rather than the cut version (which you generally won't want), **"filter_terminations_using_traces"** stops faults on opposite sides of an earlier fault from generating termination lines on each other.

```
{
    "name": "kaikoura_suggested",
    "faults_filename": "{name}_hierarchy151222_nonzero.csv",
    "basement_name": "Meshes/depths_smoothed",
    "uncut_zs": [
        -30000,
        0
    ],
    "trace_buffer": 20000.0,
    "cut_zs": [
        -30000,
        0
    ],
    "resolution": 1000,
    "ignore_terminations": true,
    "run_intersection_check": true,
    "use_uncut_terminations": false,
    "filter_terminations_using_traces": true
}
```

2. `File -> Fault System Modelling -> Create termination models` (seems to fail on some faults, but work if go to Projects – retry failed task after running)
3. File – Fault System Modelling – add termination boundaries
4. File – Fault System Modelling – Extract main fault parts. The final fault network will be in a folder called Meshes/{name}/extracted
5. Save the extracted meshes by right clicking on this folder – Export Meshes – Select All, Format: *.obj and choose the folder you want to save to
6. **In practice you'll want to check that this process gave you the faults you were actually expecting and that they've been cut/truncated properly so…**

8.3 Quality Control

1. Import traces into leapfrog scene (drag + drop the traces folder) - I've found it helpful to go through the extracted meshes and make sure they correspond to the surface traces + don't have intersections with other faults. The most common problems are that the trace of another fault doesn't quite truncate a fault so that the fault goes beyond its surface trace – might be solved by changing "resolution" to be smaller (see above), that a fault intersection doesn't get picked up or that a fault doesn't fully cut through another so you end up with a long slot in the fault rather than a truncation.
2. To check for fault intersections in the final model:
    1. rename Meshes/{name}/output to something else e.g. Meshes/{name}/output1
    2. rename Meshes/{name}/extracted to Meshes/{name}/output
    3. run File – Fault System Modelling – check Fault Mesh intersections
    4. Change back the folder names (just so you don't forget later) - this is a bit of a hack but Check Fault Mesh Intersections works on the folder "output"
3. If there are intersections which need adding in as terminations, there are two possibilities:

1. The fault which needs truncating already has a termination listed in the terminations folder:
    1. Go to the geological model for this fault – boundary – add lateral extent – from distance function – create new distance function – choose the intersection line as the object for this distance function – buffer – set to 100m.
    2. rerun “add termination boundaries” and “extract main fault parts” – might have to right click on the new termination in the GM – switch inside
2. The fault which needs truncating doesn’t have a termination:

    a. Copy the intersection line to the “terminations” folder

    b. Rename to the name of the fault being terminated

    c. rerun “add termination boundaries” and “extract main fault parts” (I can’t remember if you need to redo the GM section from the first case)

4. If a termination line needs editing (e.g. to bring it to the edge of a fault):
    1. Right click on the termination line – edit
    2. Select snap to mesh vertex
    3. Select 3D (in panel above scene)
    4. Click on the last vertex in the line + hold down ctrl as you move it where you want it to be. Alternatively click “draw new line” in the line above and draw a new polyline as a termination boundary.
    5. rerun “add termination boundaries” and “extract main fault parts”

CHAPTER

NINE

API REFERENCE

This page contains auto-generated API reference documentation[1].

9.1 fault_mesh

Module for the creation of depth contours and associated products to help create fault meshes.

Modules:

- faults - Main scripts for projecting surface traces to depth
- utilities - Functions for reading and writing files and

9.1.1 Submodules

fault_mesh.faults

A package for generating depth contours from a fault trace Modules:

- connected - A module for joining segments of a fault system into a larger object
- generic - Sets out classes and functions that are used by other modules
- leapfrog - A module associated with reading, generating, and combining DEMs

Submodules

fault_mesh.faults.connected

Module for connecting segments into a connected fault system

Classes

ConnectedFaultSystem	Class for a connected fault system

Module Contents

class fault_mesh.faults.connected.**ConnectedFaultSystem**(*overall_name*, *cfm_faults*, *segment_names=None*, *search_patterns=None*, *excluded_names=None*, *tolerance=100.0*, *smooth_trace_refinements=5*, *trimming_gradient=1.0*)

[1] Created with sphinx-autoapi

Class for a connected fault system

Parameters

- **overall_name** (*str*)
- **segment_names** (*list*)
- **search_patterns** (*Union[str, list]*)
- **excluded_names** (*Union[str, list]*)
- **tolerance** (*float*)
- **smooth_trace_refinements** (*int*)
- **trimming_gradient** (*float*)

fault_mesh.faults.generic

Classes to act as a general/multifault. Leapfrog classes inherit from these.

Classes

GenericMultiFault	Class to hold data for multiple faults, read in from shapefile (and hopefully also tsurfaces)

Functions

smallest_difference(value1, value2)	Finds smallest angle between two bearings
normalize_bearing(bearing)	change a bearing (in degrees) so that it is an azimuth between 0 and 360.
bearing_leq(value, benchmark[, tolerance])	Check whether a bearing (value) is anticlockwise of another bearing (benchmark)
bearing_geq(value, benchmark[, tolerance])	Check whether a bearing (value) is clockwise of another bearing (benchmark)
reverse_bearing(bearing)	180 degrees from supplied bearing
reverse_line(line)	Change the order that points in a LineString object are presented.
calculate_dip_direction(line)	Calculate the strike of a shapely linestring object with coordinates in NZTM,
root_mean_square(value_array)	Helper function to turn max and min to stdev for inclusion in XML.

Module Contents

fault_mesh.faults.generic.**smallest_difference**(*value1*, *value2*)

Finds smallest angle between two bearings :param value1: :param value2: :return:

fault_mesh.faults.generic.**normalize_bearing**(*bearing*)

change a bearing (in degrees) so that it is an azimuth between 0 and 360. :param bearing: :return:

Parameters

bearing (*Union[float, int]*)

fault_mesh.faults.generic.**bearing_leq**(*value*, *benchmark*, *tolerance=0.1*)

Check whether a bearing (value) is anticlockwise of another bearing (benchmark) :param value: :param benchmark: :param tolerance: to account for rounding errors etc :return:

Parameters

- **value** (*Union[int, float]*)
- **benchmark** (*Union[int, float]*)
- **tolerance** (*Union[int, float]*)

fault_mesh.faults.generic.**bearing_geq**(*value*, *benchmark*, *tolerance=0.1*)

Check whether a bearing (value) is clockwise of another bearing (benchmark) :param value: :param benchmark: :param tolerance: to account for rounding errors etc :return:

Parameters

- **value** (*Union[int, float]*)
- **benchmark** (*Union[int, float]*)
- **tolerance** (*Union[int, float]*)

fault_mesh.faults.generic.**reverse_bearing**(*bearing*)

180 degrees from supplied bearing :param bearing: :return:

Parameters
bearing (*Union[int, float]*)

fault_mesh.faults.generic.**reverse_line**(*line*)

Change the order that points in a LineString object are presented. Updated to work with 3d lines (has_z), September 2021 Important for OpenSHA, I think :param line: :return:

Parameters
line (*shapely.geometry.LineString*)

fault_mesh.faults.generic.**calculate_dip_direction**(*line*)

Calculate the strike of a shapely linestring object with coordinates in NZTM, then adds 90 to get dip direction. Dip direction is always 90 clockwise from strike of line. :param line: Linestring object :return:

Parameters
line (*shapely.geometry.LineString*)

fault_mesh.faults.generic.**root_mean_square**(*value_array*)

Helper function to turn max and min to stdev for inclusion in XML. :param value_array: Differences of values (e.g. sr_min and sr_max) from mean. :return:

Parameters
value_array (*Union[numpy.ndarray, list, tuple]*)

class fault_mesh.faults.generic.**GenericMultiFault**(*fault_geodataframe*, *sort_sr=False*, *remove_colons=False*, *dip_choice='pref'*, *check_optional_fields=True*)

Class to hold data for multiple faults, read in from shapefile (and hopefully also tsurfaces)

Parameters

- **fault_geodataframe** (*geopandas.GeoDataFrame*)
- **sort_sr** (*bool*)
- **remove_colons** (*bool*)

- **dip_choice** (*str*)
- **check_optional_fields** (*bool*)

static gdf_from_nz_cfm_shp(*filename*, *exclude_region_polygons=None*, *depth_type=None*, *exclude_region_min_sr=1.8*, *include_names=None*, *exclude_aus=True*, *exclude_zero=True*)

Read CFM shapefile

Parameters

- **filename** (*str*)
- **exclude_region_polygons** (*List[shapely.geometry.Polygon]*)
- **depth_type** (*str*)
- **exclude_region_min_sr** (*float*)
- **include_names** (*list*)
- **exclude_aus** (*bool*)
- **exclude_zero** (*bool*)

fault_mesh.faults.leapfrog

Classes that implement the Leapfrog fault model. Inherit from GenericFault and GenericMultiFault.

Classes

LeapfrogMultiFault	Class to hold data for multiple faults, read in from shapefile (and hopefully also tsurfaces)
LeapfrogFault	Represents either a whole fault (for simple faults) or one segment. Behaviours is slightly

Module Contents

class fault_mesh.faults.leapfrog.**LeapfrogMultiFault**(*fault_geodataframe*, *sort_sr=False*, *segment_distance_tolerance=100.0*, *smoothing_n=None*, *remove_colons=True*, *dip_choice='pref'*, *trimming_gradient=1.0*, *epsg=None*, *dip_multiplier=1.0*, *strike_multiplier=0.5*, *check_optional_fields=True*)

Bases: *fault_mesh.faults.generic.GenericMultiFault*

Class to hold data for multiple faults, read in from shapefile (and hopefully also tsurfaces)

Parameters

- **fault_geodataframe** (*geopandas.GeoDataFrame*)
- **sort_sr** (*bool*)
- **segment_distance_tolerance** (*float*)
- **smoothing_n** (*int*)
- **remove_colons** (*bool*)

- **dip_choice** (*str*)
- **trimming_gradient** (*float*)
- **epsg** (*int*)
- **dip_multiplier** (*float*)
- **strike_multiplier** (*float*)
- **check_optional_fields** (*bool*)

find_connections(*verbose=True*)

Find all connections between faults in the fault list using networkx :param verbose: print out information about individual connections :return:

Parameters
verbose (*bool*)

class fault_mesh.faults.leapfrog.**LeapfrogFault**(*parent_multifault=None*, *smoothing=5*, *trimming_gradient=1.0*, *segment_distance_tolerance=100.0*, *parent_connected=None*)

Bases: fault_mesh.faults.generic.GenericFault

Represents either a whole fault (for simple faults) or one segment. Behaviours is slightly

Parameters

- **parent_multifault** (LeapfrogMultiFault)
- **smoothing** (*int*)
- **trimming_gradient** (*float*)
- **segment_distance_tolerance** (*float*)

property is_segment

Records whether instance is a segment of a larger multi-segment fault like the Alpine Fault. :return:

property smoothing

n value to use in Chaikin's corner-cutting algorithm. :return:

property trimming_gradient

Factor that controls how much the ends of segment contours of a multi-segment fault are shortened to allow :return:

property parent

Return LeapfrogMultiFault instance that this fault is part of. :return:

depth_contour(*depth*, *smoothing=True*, *km=False*)

Generate contour of fault surface at depth below surface :param depth: In metres, upwards is positive :param smoothing: N for use with Chaikin's corner cutting :param km: If True, divide depth by 1000 :return: LineString or MultiLineString representing contour

Parameters

- **depth** (*float*)
- **smoothing** (*bool*)

extend_footprint(*end_i*, *other_end*, *other_segment*, *deepest_contour_depth=30000.0*, *search_line_length=150000.0*, *buffer_size=5000.0*, *fall_back_distance=40.3*)

Parameters

- **end_i** (*shapely.geometry.Point*) – End to extend
- **other_end** (*shapely.geometry.Point*) – Other end of segment
- **other_segment** (LeapfrogFault) – Other
- **deepest_contour_depth** (*float*)
- **search_line_length** (*float*)
- **buffer_size** (*float*)
- **fall_back_distance** (*float*)

Returns

fault_mesh.utilities

Test

Submodules

fault_mesh.utilities.cubit

Functions to make journal files for remeshing using cubit.

fault_mesh.utilities.graph

Submodule that uses networkx to find connections between segments in a fault map

Functions

to_graph(node_list)	Answer from https://stackoverflow.com/a/4843408

Module Contents

fault_mesh.utilities.graph.**to_graph**(*node_list*)

Answer from https://stackoverflow.com/a/4843408 :param node_list: :return:

Parameters

node_list (*List[List[str]]*)

fault_mesh.utilities.merging

Functions for merging segments that are nearly adjacent.

fault_mesh.utilities.smoothing

A convenience script for running many combinations of chunk_sizes and numbers_of_cores for a small sub-set of a catchment to help with the selection of an appropiate chink_size and number_of_cores before processing an entire catchment.

Functions

chaikins_corner_cutting(coords[, refinements])	Chaikin's corner cutting algorithm for smoothing a line.

Module Contents

fault_mesh.utilities.smoothing.**chaikins_corner_cutting**(*coords*, *refinements=5*)

Chaikin's corner cutting algorithm for smoothing a line.

CHAPTER

TEN

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

f

INDEX

R

S

T